1.**Descargar y exploración del corpus (ProyectF1.ipynb)**

Para este proyecto he cogido el dataset **All_Beauty_5.json**, que contiene las siguientes columnas y explicadas:

- **overall**: Calificación general del producto, expresada en estrellas (típicamente de 1 a 5, donde 5 es la mejor calificación).
- **verified**: Indica si la reseña fue verificada. True significa que Amazon ha confirmado que la persona que escribió la reseña realmente compró el producto. False significa que no se ha podido verificar la compra.
- **reviewTime**: Fecha en la que se escribió la reseña, en formato de texto.
- **reviewerID**: Identificador único del usuario que escribió la reseña. Este ID es anónimo y no revela la identidad real del usuario.
- **asin**: Identificador único del producto (Amazon Standard Identification Number). Este código es específico de cada producto en Amazon.
- **style**: Información adicional sobre el producto, como el tamaño, el sabor, el color, etc. Esta información suele estar en formato de diccionario (clave-valor).
- **reviewerName**: Nombre del usuario que escribió la reseña (puede ser un nombre real o un seudónimo).
- **reviewText**: Texto completo de la reseña del usuario.
- **summary**: Un resumen corto o título de la reseña, proporcionado por el usuario.
- **unixReviewTime**: Fecha de la reseña en formato Unix timestamp. Este es un número que representa la cantidad de segundos transcurridos desde el 1 de enero de 1970 (UTC). Es útil para realizar cálculos y ordenamientos por fecha.
- **vote**: Número de usuarios que encontraron útil la reseña. Este campo puede contener valores numéricos o NaN (Not a Number) si la reseña no ha recibido votos.
- **image**: Información sobre si la reseña incluye imágenes. En este ejemplo, esta columna solo contiene valores NaN, lo que sugiere que ninguna de las reseñas mostradas incluye imágenes.

  En resumen, este DataFrame contiene información sobre las reseñas de productos, incluyendo calificaciones, textos de reseñas, información del producto y del usuario, y datos sobre la utilidad de las reseñas

En mi análisis se va a basar en la columna ***reviewText***, que tiene los textos de las reseñas. Sin embargo, puede haber valores nulos (NaN), por lo que realizo un preprocesamiento para eliminarlos.

Análisis Exploratorio

1. Cardinalidad del vocabulario

La cardinalidad del vocabulario es simplemente el número de palabras únicas en el corpus. Para obtenerla, concateno todas las reseñas, las tokenizo y luego las convierto en un conjunto para eliminar duplicados.

En este DataSet nos da **9234 palabras**

2. Distribución de reseñas por número de estrellas

Veo cuantas reseñas hay por calificación (de 1 a 5 estrellas) usando la columna ***overall***.

Compruebo que hay más números de estrellas del número 5, con un 88.22%, las demás son insignificantes como vemos en el gráfico.

3. Nº de reviews positivas y negativas

Las reseñas positivas las considero aquellas con más de 3 estrellas y las negativas aquellas con 3 o menos estrellas. Uso apply() para crear una nueva columna ***sentimiento***.

Y el gráfico nos muestra que hay 4976 positivas y 288 negativas.

4. N-grams más frecuentes

Es una secuencia de n palabras, aquí cálculo los bigramas (n=2), y trigramas (n=3) más frecuentes.
Lo que puede ayudarme a entender las combinaciones mas frecuentes de palabras.

Aquí se muestra:

```
los 20 2-gramas mas frecuentes son:

('i', 'have'): 736
('this', 'is'): 687
('of', 'the'): 582
```

```
('it', 'is'): 580
('i', 'love'): 579
('my', 'hair'): 559
('is', 'a'): 537
('in', 'the'): 510
('love', 'this'): 435
('this', 'product'): 419
('and', 'i'): 387
('i', 'was'): 350
('and', 'the'): 328
('love', 'the'): 309
('i', 'am'): 293
('is', 'the'): 289
('my', 'skin'): 286
('and', 'it'): 284
('for', 'a'): 265
('for', 'the'): 253
los 20 3-gramas mas frecuentes son:
```

```
('this', 'is', 'a'): 216
('i', 'love', 'this'): 161
('i', 'love', 'the'): 155
('this', 'is', 'the'): 123
('a', 'lot', 'of'): 115
('i', 'have', 'been'): 109
('i', 'use', 'it'): 100
('is', 'a', 'great'): 87
('pre', 'de', 'provence'): 87
('it', 'is', 'a'): 86
('i', 'have', 'used'): 82
('hard', 'to', 'find'): 78
('i', 'would', 'recommend'): 73
('this', 'product', 'is'): 71
('my', 'hair', 'is'): 70
('have', 'been', 'using'): 69
('one', 'of', 'the'): 67
('this', 'is', 'my'): 66
('it', 'makes', 'my'): 64
('to', 'find', 'it'): 63
```

5. Nubes de palabras

Nos muestra las palabras más frecuentes de nuestro corpus, como por ejemplo vemos en la imagen prodruct, love, hair, shapoo,…

También he creado dos nubes, una con las palabras de sentimiento positivo y otra negativo. Y seguidamente lo mismo pero con palabras que no se repitan en ningún de los dos lados, y así se ve más claro .

6. Visualización en 2 dimensiones de algunos Word embeddings calculados con Word2Vec

Para visualizar los vectores de palabras, entreno con un modelo de Word2Vec usando las reseñas y luego las reduzco la dimensionalidad con t-SNE para proyectar las palabras en 2D.
Esto nos permite ver como se relacionan las palabras semánticamente entre si como vemos en el gráfico de colores.

También he usado **FastText** y **Bert**

FastText es otra variante que es similar a Word2Vec, pero en lugar de representar solo palabras completas como vectores, también toma en cuenta las sub-palabras (substrings) dentro de cada palabra. Esto es útil para palabras raras o fuera del vocabulario.

BERT es un modelo de lenguaje basado en *transformers* que produce representaciones contextuales de las palabras (es decir, las representaciones dependen de las palabras que lo rodean en el texto). Este código:

- Utiliza el tokenizer y el modelo BERT de la librería transformers de Hugging Face.
- Para cada palabra, obtiene su representación vectorial de BERT.
- Aplica *t-SNE* para reducir la dimensionalidad y visualizar las palabras en 2D.
- Finalmente, genera un gráfico de dispersión para visualizar la relación entre las palabras.

Además le he añadido un análisis de frecuencia con las palabras más comenes y vemos que las palabras más comunes son las que generalmente pertenecen a las palabras STOP-WORDs, que en preprocesado las quitaré.

He realizado también un gráfico de tendencias temporales, donde veo que a partir de 2013 aumentan las ventas.

Uno de los usuarios más activos, que para nuestro análisis no será muy útil.

2. Etapa de preprocesado de texto (ProyectoF2.ipynp)

Uso el modelo de spacCy.

Cargo el modelo de inglés de spaCy, para poder realizar las tareas como elmatización y eliminación de stopwords.

1. Funciones de Preprocesamiento
   - Corregir errores ortográficos

Para ello uso TextBlob, para corregir los errores ortográficos comunes en el texto. TextBlob sirve para detectar las palabras mal escritas y las reemplaza por las correctas.

- Expandir las contracciones
  Expande contracciones comunes en in inglés.

- Convertir números a palabras
  Paso los números en el texto a palabras, esto es útil para tratar de normalizar el texto y no perder información de números.

- Convertir a minúsculas
  Convierto todo el texto a minúsculas, lo que es útil para evitar que palabras en diferentes formas de mayúsculas sean tratadas como diferentes.

2. Función de Prepocesamiento Principal

   Esta es la función principal del preprocesamiento. Se realiza lo siguiente:

   - Strip(): Elimina espacios en blanco al inicio y final del texto
   - Convertir_minuscula(): Convierte todo el texto a minúsculas
   - Corregir_ortografia():Corregir errores ortográficos.
   - expandir_contracciones():Expande las contracciones
   - convertir_numeros_a_palabras():Elimina signos de puntuación.
   - nlp(texto):Tokeniza el texto usando spaCy.
   - lematización: Se toma la lema de cada palabra (su forma básica) y se eliminan stopwords y signos de puntuación innecesarios.

3. Verificación de Nan y Aplicación del Prepocesamiento
   Verifico si hay valores NaN(vacíos) en la columna ‘reviewText’ y luego los reemplazo por cadena vacía.

4. Aplicar Preprocesamiento al DataFrame

   Aplico la función de preprocesamiento a cada texto en la columna ‘reviewText’ y guardo el texto en una nueva columna llamada ‘reviewTextProcesado’.

5. Verificación y Guardado de Resultados
   Verifico que el preprocesamiento se realizó correctamente.
   Y finalmente, guardo el DataFrame procesado en un archivo CSV en la ruta indicada. (./datos_prepocesados.csv)

3. Etapa de entrenamiento y testeo de un modelo de análisis de sentimiento y Reporte de métricas y conclusiones

Está parte la he realizado en diferentes collabs y cada uno explicado con detenimiento y finalmente he construido un pipelline

(ProyectF3.ipynb)

Lo primero creo una columna llamada sentiment_label que contiene valores binarios(1 o 0):

etiqueta 1 es positiva

etiqueta 0 es negativa

Luego separo los datos de entrenamiento(X_train, y_train) y el conjunto de prueba (X_test, y_test), donde:

X: la columna de reviewTextProcesado, que contiene los textos de las reseñas procesadas

y: la columna sentiment_label, que contiene las etiquetas binarias de sentimiento.

El conjunto de entrenamiento representa el 75% de los datos y el conjuento de prueba el 25%. Uso train_test_split para dividir los datos de forma aleatoria, con una semilla (random_state = 42).

Verifico que no haya valores nulos en los dos conjuntos, y si los hubiera los reemplazo por cadenas vacías(‘ ‘)

Vectorización de Textos con TF-IDF

Estoy usando un Vectorizador TF-IDF, para convertir el texto de las reseñas en vectores númericos que los modelos puedan procesar.

TF-IDF, mide la importancia de una palabra dentro de un documento, en comparación con su frecuencia en otros documentos del conjunto.

Max_df=0.95: Las palabras que aparecen en mas de un 95% de los documentos serán ignoradas, ya que serán palabras demasiado comunes y no aportan mucho valor para la clasificación.

Min_df=3:Las palabras que aparecen en menos de 3 documentos también serán ignoradas, ya que pueden ser demasiado raras y no ser útiles para el modelo.

Transformo de Texto a Vectores TF-IDF. Una vez hecho esto tengo dos matrices de características X_train_ y X_test_

Ahora hago un anális de las 10 primeras palabras más revelantes según TF-IDF

He creado un diccionario que almacena el valor de IDF de cada palabra, que indica la relevancia de la palabra a través del conjunto de documentos. Luego para un conjunto de palabras definidas por mi, compruebo su valor IDF.

Si una palabra tiene valor IDF muy bajo implica que aparece con mucha frecuencia, lo que indica que no es muy revelante, y si es muy alto, es más importante y única en el conjunto de datos, y si sale OOV, es que no esta en nuestro conjunto de datos.

Palabra	IDF
happy	4.384137258916847
love	2.2232189063490937
product	2.212953632289499
shampoo	3.139939653280402
time	3.258529465906726
good	2.534390532038443
smell	2.6460305031811124
hair	2.720946392971186

and	OOV

****** X2 scores ******

Con esto, consigo que aparezcan en los resultados de este análisis las palabras más relevantes para la tarea de clasificación de sentimientos: switch, slight, comment, basic, terribly, large, decide, fairy…

******Modelo de Regresión Logística ******

Entreno un modelo de regresión logística en función de diferentes valores de un parámetro llamado c, que regula la regularización del modelo

Valores de c=[0.01, 0.05, 0.25, 0.5, 1, 10, 100, 1000, 10000]

Cuando C es pequeño el modelo está altamente regularizado y no tiene suficiente flexibilidad para ajustar los datos con precisión, lo que puede limitar su capacidad para hacer buenas predicciones.

Pero a medida que aumento c, la regularización disminuye, lo que permite al modelo ajustarse más a los datos de entrenamiento y mejorar su rendimiento.

Accuracy for C=0.01: 0.9453717754172989
Accuracy for C=0.05: 0.9453717754172989
Accuracy for C=0.25: 0.9453717754172989
Accuracy for C=0.5: 0.9453717754172989
Accuracy for C=1: 0.9650986342943855
Accuracy for C=10: 0.988619119878604
Accuracy for C=100: 0.988619119878604
Accuracy for C=1000: 0.9893778452200304
Accuracy for C=10000: 0.9893778452200304

Ahora veo las **métricas** de matriz de confusión , el reporte de clasificación y el accuracy:

La matriz de confusion

[[59 13]

[1 1245]]

Lo que nos quiere decir:

59 veces el modelo predijo correctamente la clase 0

13 incorrectamente la clase 0

1245 veces el modelo predijo correctamente la clase 1

1 vez mal la clase 1

Accuracy score: 0.98937784

Significa que el modelo clasificó correctamente el 98,94% de las instancias en el conjunt de prueba de datos

Seguido he realizado una función ***predict_review_sentiment***, que predice el sentimiento de una reseña, es decir si es positivo o negativo, en función de un índice de revisión y un modelo entrenado.

Esta diseñado para mostrar cómo el modelo de regresión logística predice el sentimiento de reseñas de forma aleatoria. También maneja errores en caso de que las columnas no existan o si los índices no son válidos. La salida permite ver la calidad de las predicciones comparando con las etiquetas reales.

Review no. 15

Actual sentiment: 1

Prediction: 1

Review no. 42

Actual sentiment: 0

Prediction: 0

Review no. 87

Actual sentiment: 1

Prediction: 0

Review no. 133

Actual sentiment: 0

Prediction: 1

Review no. 189

Actual sentiment: 1

Prediction: 1

******Análisis de topicos con LDA*****

1. Entrenamiento del modelo LDA
   He definido un número óptimo de temas (5), basado en la distribución de probabilidad de los tópicos, y entrene el modelo LDA con 5 tópicos usando la función ***train_lda_model()***

2. Obtención de las palabras clave de cada Tópico
   Después he extraído las palabras más importantes para cada tópico, mostrando la probabilidad de aparición en cada tópico

3. Formato y Visualización de los Tópicos y Documentos.
   He creado una función *format_topics_sentences()*, que asigna el tema dominante de cada documento en función de la probabilidad más alta

4. Distribución de Documentos por Tópico
   Luego contabilice cuántos documentos corresponde a cada tópico, y el resultado me muestra cómo se distribuye los documentos entre los 5 tópicos entrenados.

5. Filtrado por palabras clave
   He realizado una búsqueda de documentos que contienen las palabras claves específicas (“shampoo”, “hair”), utlizando mascaras booleanas. Los documentos que cumplen con alguna de las palabras clave de interés fueron filtradas para posterior análisis.

6. Visualización de la Distribución de Tópicos
   He realizado un gráfico de barras para mostrar la distribución de documento por tópico, y así ayudar a visualizar cual tópico es el más representado y cual menos.

7. Exploración de un Tópico Especifico

8. Documento Representativo por Tópico

He comprobado que el proceso LDA ha sido exitoso para obtener los principales tópicos de un conjunto de documentos y entender como se distribuyen los documentos entre estos tópicos. Además pude realizar búsquedas de documentos con ciertas palabras clave y visualizar cómo los tópicos se distribuyen en mi conjunto de datos.

(ProyectF4.ipynb)

Aquí he probado los siguientes módelo LSTM, GRU y RNN + Word2vecEmbedding

Modelo LSTM

Primero como siempre,he preparado las etiquetas binarias, se ha preprocesado el testo de las reseñas, tokenizado y preparación de los datos para Keras, convirtiendo las reseñas en secuencias numéricas. Y realice padding a las secuencias para que todas tuvieran la misma longitud(500 palabras). También un manejo de desbalance de clases(Oversampling)

Empiezo el modelo LSTM:

Definido con los siguientes componentes:

Capa Embedding: Para convertir las secuencias numéricas en vectores de palabras densos de 32 dimensiones.
Capa LSTM: LSTM con 100 unidades, con dropout y recurrent_dropout para prevenir el sobreajuste.
Capa densa: Una capa densa con una sola neurona y función de activación sigmoid para clasificación binaria.

Entreno el modelo con 10 épocas, y usando un tamaño de batch de 32 y un 10 % de los datos de entrenamiento para validación
El modelo usa la función de perdida binary_crossentopy y el optimizador Adam.

En el conjunto de prueba obtuve de precisión 94.5% en el conjunt de test y las prediciciones han sido convertidas a etiquetas binarias con un umbral de 0.5 para la función sigmoide.

Los resultados obtenidos:
Matriz de confusión:
[[0 58]
[0 996]]
Lo que nos dice que predijo correctamente las etiquetas 1(positivas para 996 casos)
Y para las etiquetas 0, predijo 0 para todos los casos de etiqueta verdadera era 0, lo que genera un falso negativo en 58 muestras.

Reporte de clasificación

	precision	recall	f1-score	support
0	0.00	0.00	0.00	58
1	0.94	1.00	0.97	996
accuracy			0.94	1054

el f1-score es bastante alto para la clase 1(0.97), lo que indica un buen equilibrio entre precisión y recall.

Nos advierte que no hubo predicciones correctas para la clase 0, lo que implica que el modelo esta sesgado hacia la clase 1, probablemente debido al desbalance en el conjunto de datos, a pesar del uso de sobremuestreo.

Conclusión:

Mi modelo con LSTM ha logrado buenos resultados para predecir reseñas positivas, pero necesita mejor clasificación de las reseñas negativas.

Modelo GRU con Dropout y regularizacioón L2

He implentado un modelo con la arquitectura GRU Bidireccional:

Embeddings: Utilizo un Embedding de tamaño 32 para representar las palabras en un espacio denso. Esto permite que el modelo aprenda representaciones semánticas de las palabras durante el entrenamiento.
Bidireccional: Al usar una capa GRU bidireccional, el modelo puede considerar la información del contexto tanto anterior como posterior de las palabras en una secuencia, lo que puede mejorar el rendimiento en tareas de texto.
Dropout y Regularización L2: Estas técnicas ayudan a evitar el sobreajuste al regularizar los pesos del modelo y "apagando" aleatoriamente algunas conexiones entre neuronas durante el entrenamiento.
Capa densa: Al final, agrego una capa densa con una activación sigmoide para clasificar las secuencias como positivas (1) o negativas (0).

El modelo se entrena duante 10 epocas y se observa que la precisión de validación es de casi el 100%, lo que indica que el modelo está aprendiendo muy bien y ajustando los pesos. En cuanto la pérdida vemos que también disminuye con cada época, lo que es una buena señal de que el modelo está aprendiendo de manera efectiva.

Resultados del modelo:

Precisión de prueba(Test accuracy): La precisión es del 98.86%, lo que es muy buen resultado.

Matriz de confusión

[[52 6]

[6 990]]

La clase 0(negativa), tiene 52 verdaderos negativos y 6 falsos positivos, lo que significa que el modelo a veces confunde algunas revisiones negativas con positivas

La clase 1(positiva), tiene 6 falsos negativos y 990 verdaderos positivos, lo que indica que el modelo es muy eficaz al predecir las reseñas positivas.

Reporte clasificación

precision recall f1-score support

0 0.90 0.90 0.90 58

1 0.99 0.99 0.99 996

accuracy 0.99 1054

macro avg 0.95 0.95 0.95 1054

weighted a vg 0.99 0.99 0.99 1054

Los f1-scores para ambos son bastantes altos(0.90 para la clase 0 y 0.99 para la clase 1). Lo que significa que el modelo tiene un buenequilibrio entre precisión y recall

Comparación con LSTM:

El modelo GRU parece estar funcionando mejor que el LSTM en términos de precisión y recall

Modelo RNN+Word2vecEmbedding

Primero hago el preprocesamiento como siempre.

Mi modelo construido tiene la siguiente forma:

Capa de Embedding: Se utiliza la matriz de embedding generada por Word2Vec. Aquí se usan los vectores densos que Word2Vec que ha aprendido.

LSTM Bidireccionales: Tienes dos capas LSTM. La primera tiene return_sequences=True, lo que permite pasar la salida al siguiente LSTM como secuencias. La segunda capa LSTM tiene return_sequences=False, lo que significa que solo se pasa un vector de salida, lo que es adecuado para clasificación binaria.

Capa densa de salida: Al final, usas una capa densa con activación sigmoide, que es ideal para problemas de clasificación binaria, ya que produce una salida entre 0 y 1.

En el entrenamiento uso Adam y binary_crossentropy como función de perdida. Luego el modelo se entrena durante 10 epocas con tamño de lote de 64. Y durante el entrenamiento, el modelo muestra la precisión tanto en el conjunto de entrenamiento como el de validación.

Resultado del modelo:

La precisión de prueba: 0.891640000

Matriz de confusión:

[[10513 1987]

[722 11778]]

El modelo está prediciendo correctamente tanto las clases 0 (negativas) como las clases 1 (positivas). El modelo tiene un alto número de verdaderos positivos (TP) para la clase 1 (11778) y bajos falsos negativos (FN) para la clase 1 (722), lo que indica que el modelo es eficiente en clasificar reseñas positivas.

Reporte de clasificación

	precision	recall	f1-score	support
0	0.94	0.84	0.89	12500
1	0.86	0.94	0.90	12500
Accuracy			0.89	25000
macro avg	0.90	0.89	0.89	25000
weighted avg	0.90	0.89	0.89	25000

F1-score es excelente para ambas clases, especialmente para la clase 1(positiva), con un f1 de 0.90 en cambas clases

Conclusión:

El modelo funciona muy bien, con una precisión final de 88.9%. Esto es un buen punto de partida para el modelo de clasificación de texto como este.

***

Finalmente he realizado un pipeline como los resueltos en clase

***

(PipelineFinal.ipynb)

Realiza una clasificación de texto para predecir el sentimiento (positivo o negativo) de reseñas de productos basándose en el texto preocesado.

1. He cargado las líbrerias y los datos, donde reviewTestProcesado contiene el texto limpio de las reseñas y overall es la clasificación del producto (en una escala de 1 a 5)
2. Prepocesamiento de los datos: Creo la etiqueta binaria de sentimiento utilizado la calificación overall, donde si es mayor o igual que 4 se etiqueta como 1(positivo) y 0 para calificaciones menores. También se verifica y rellena cualquier valor NaN de la

columna reviewTestProcesado con una cadena vacía para evitar problema.

3. División de datos de entrenamiento y prueba: entrenamiento el 80% y prueba 20%, aseguandome que la distribución de las etiquetas sean proporcional(stratify=labels)
4. Extracción de características(Features): defino una lista de diferentes métodos para extraer características de los textos de las reseñas:

   Unigramas: Se usa el CountVectorizer para obtener un vocabulario de unigramas (palabras individuales).

   Unigramas con max_df=0.95: Filtra palabras que aparecen en más del 95% de los documentos.

   Bigramas: Se extraen pares de palabras (bigrams).

   Trigramas: Se extraen tripletas de palabras (trigrams).

   TF-IDF: Se utiliza el TfidfVectorizer para ponderar las palabras en función de su frecuencia inversa de documentos, reduciendo el impacto de palabras comunes.
5. Clasificadores: se definen varios para evaluar el rendimiento:

   KNN: K-Nearest Neighbors con diferentes configuraciones (k=5 y k=3).

   Naive Bayes (GaussianNB).

   Random Forest (RF).

   Máquinas de soporte vectorial lineales (SVM).
6. Entrenamiento y evaluación:

   Para cada combinación de **método de extracción de características** y **clasificador**, el pipeline entrena el modelo con los datos de entrenamiento y luego evalúa el rendimiento en los datos de prueba.

   Normalización: Si el método de características no es TF-IDF, los vectores de características son normalizados para que cada documento tenga la misma escala.

   Métricas de evaluación: Para cada combinación, se calculan las siguientes métricas:

   - **Matriz de confusión**: Muestra el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.
   - **Reporte de clasificación**: Muestra las métricas de **precisión**, **recall**, y **F1-score** para cada clase (positivo y negativo).

Explicación de los resultados:

Los resultados muestran el rendimiento de los diferentes modelos con las distintas representaciones de características (unigramas, bigramas, trigramas, TF-IDF) para predecir el sentimiento de las reseñas:

Nomenclatura para la matriz de confusión: **TN**: Predicciones correctas de la calse 0, **FP**:Predicciones incorrectas de la clase 1, **FN**: Predicciones incorrectas de la clase 0, **TP**: Predicciones correctas de la clase 1.

1. **Con unigrama (ngram=1)**
    - **KNN**:
        - **Matriz de Confusión**:

          TN=31,FP=27,FN=0,TP=996

        - **Reporte**:
            - Precisión: 0.97, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.97
    - **Naive Bayes**:
        - **Matriz de Confusión**:

          TN=52,FP=6,FN=267,TP=729

        - **Reporte**:
            - Precisión: 0.99, Recall: 0.73, F1-Score: 0.84
            - Exactitud: 0.74
    - **Random Forest**:
        - **Matriz de Confusión**:

          TN=52,FP=6,FN=267,TP=729

        - **Reporte**:
            - Precisión: 0.99, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.99
    - **SVM**:
        - **Matriz de Confusión**:

          TN=49,FP=9,FN=1,TP=995

        - **Reporte**:
            - Precisión: 0.99, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.99

2. **Con bigramas (ngram=2)**:
    - **KNN**:
        - **Matriz de Confusión**:

          TN=32,FP=26,FN=4,TP=992

        - **Reporte**:
            - Precisión: 0.97, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.97
    - **Naive Bayes**:
        - **Matriz de Confusión**:

          TN=52,FP=6,FN=267,TP=729

        - **Reporte**:
            - Precisión: 0.99, Recall: 0.73, F1-Score: 0.84
            - Exactitud: 0.74
    - **Random Forest**:
        - **Matriz de Confusión**:

          TN=36,FP=22,FN=3,TP=993

        - **Reporte**:
            - Precisión: 0.98, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.98
    - **SVM**:
        - **Matriz de Confusión**:

          TN=33,FP=25,FN=5,TP=991

        - **Reporte**:
            - Precisión: 0.99, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.97

3. **Con trigramas (ngram=3)**:
    - **KNN**:
        - **Matriz de Confusión**:

          TN=4,FP=54,FN=0,TP=996

        - **Reporte**:
            - Precisión: 0.95, Recall: 1.00, F1-Score: 0.97
            - Exactitud: 0.95
    - **Naive Bayes**:
        - **Matriz de Confusión**:

          TN=58,FP=0,FN=747,TP=249

        - **Reporte**:

        - Precisión: 1.00, Recall: 0.25, F1-Score: 0.40
        - Exactitud: 0.29
    - **Random Forest**:
        - **Matriz de Confusión**:

          TN=5,FP=53,FN=2,TP=994

        - **Reporte**:
            - Precisión: 0.95, Recall: 1.00, F1-Score: 0.97
            - Exactitud: 0.95
    - **SVM**:
        - **Matriz de Confusión**:

          TN=5,FP=53,FN=2,TP=994

        - **Reporte**:
            - Precisión: 0.95, Recall: 1.00, F1-Score: 0.97
            - Exactitud: 0.95

4. **Con tf-idf**:
    - **KNN**:
        - **Matriz de Confusión**:

          TN=31,FP=27,FN=0,TP=996

        - **Reporte**:
            - Precisión: 0.97, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.97
    - **Naive Bayes**:
        - **Matriz de Confusión**:

          TN=49,FP=9,FN=270,TP=726

        - **Reporte**:
            - Precisión: 0.99, Recall: 0.73, F1-Score: 0.84
            - Exactitud: 0.74
    - **Random Forest**:
        - **Matriz de Confusión**:

          TN=47,FP=11,FN=0,TP=996

        - **Reporte**:
            - Precisión: 0.99, Recall: 1.00, F1-Score: 0.99

        - Exactitud: 0.99
    - **SVM**:
        - **Matriz de Confusión**:

          TN=49,FP=9,FN=1,TP=995

        - **Reporte**:
            - Precisión: 0.99, Recall: 1.00, F1-Score: 0.99
            - Exactitud: 0.99

<u>Conclusiones:</u>

- **El mejor desempeño se observa con el clasificador Random Forest y SVM**, donde ambos tienen una **alta precisión** y **recuperación** en la mayoría de los casos, con una **exactitud de 0.99**.

- **Naive Bayes** mostró un desempeño inferior, especialmente con el **bigramas** y **trigramas**, donde la **recuperación** fue baja, indicando que tuvo problemas para identificar correctamente las clases positivas.

- Los **unigramas** y **tf-idf** parecen ser las mejores características, ya que los clasificadores alcanzan altos puntajes en la mayoría de las métricas.