UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

CLASIFICACIÓN DE COMENTARIOS HACIA FUTBOLISTAS EN INSTAGRAM

Autor: Jorge Vinagre Triguero

Director: Dr. Santi Seguí Mesquida

Realizat a: Departament de Matemàtiques i Informàtica

Barcelona, 13 de juny de 2023

Resum:

Actualment s'han implementat múltiples models d'aprenentatge automàtic per a l'anàlisi de sentiments que tenen la capacitat de classificar text segons si són positius o negatius, tant paraules individuals com oracions complexes. Tot i això, els models amb més taxa d'encert han requerit una alta capacitat computacional per poder classificar el text en qüestió i també per anar actualitzant-se constantment amb més exemples.

En aquest cas, s'intenta classificar la polaritat de comentaris ofensius a les xarxes socials, concretament, a Instagram i dirigits cap a futbolistes professionals.

Per això, els objectius d"aquest estudi s"han definit primerament com la recol·lecció de dades de forma autònoma, i la creació d"un conjunt de dades per després entrenar models.

Seguint aquest fil, els objectius següents són investigar sobre les diferents metodologies, tecnologies i models de la biblioteca d'aprenentatge automàtic a Python anomenada scikit-learn. Finalment, després de fer una comparativa entre els 5 models seleccionats, s'escollirà un d'aquests models per determinar la polaritat dels comentaris extrets prèviament mitjançant la classificació de sentiments (sentiment analysis).

Malgrat el baix nivell de coneixement del qual es disposava personalment a l'àmbit del Processament del Llenguatge Natural al principi, i la manca de capacitat computacional, els resultats del model els puc donar com a satisfactoris, ja que s'està obtenint una classificació coherent basada en una justificació fundada.

No obstant això, si la planificació inicial hagués estat més encertada, els resultats podrien haver millorat i si aquestes dades pretenen ser usades en un altre projecte, el model hauria de ser entrenat en una màquina amb més capacitat computacional mitjançant la qual es pugui entrenar el model durant més temps amb mètodes més avançats, com algun dels que avui dia estan considerats com a part de l'estat de l'art en aquest àmbit.

Resumen:

Actualmente se han implementado múltiples modelos de aprendizaje automático para el análisis de sentimientos que tienen la capacidad de clasificar texto según si es positivo o negativo, tanto palabras individuales como oraciones complejas. Sin embargo, los modelos con más tasa de acierto han requerido una alta capacidad computacional para poder clasificar el texto en cuestión y también para ir actualizándose constantemente con más ejemplos.

En este caso, se intenta clasificar la polaridad de comentarios ofensivos en redes sociales, concretamente, en Instagram y dirigidos hacia futbolistas profesionales.

Por ello, los objetivos de este estudio se han definido primeramente como la recolección de datos de forma autónoma, y la creación de un conjunto de datos para luego entrenar modelos.

Siguiendo este hilo, los siguientes objetivos son investigar sobre las diferentes metodologías, tecnologías y modelos de la biblioteca de aprendizaje automático en Python llamada scikit-learn. Finalmente, después de hacer una comparativa entre los 5 modelos seleccionados, se escogerá uno de estos modelos para determinar la polaridad de los comentarios extraídos previamente mediante la clasificación de sentimientos (“sentiment analysis”).

A pesar del bajo nivel de conocimiento del cual se disponía personalmente en el ámbito del Procesamiento del Lenguaje Natural al principio, y la falta de capacidad computacional, los resultados del modelo los puedo dar como satisfactorios, ya que se está obteniendo una clasificación coherente basada en una justificación fundada.

No obstante, si la planificación inicial hubiera estado más acertada, los resultados podrían haber mejorado y si estos datos pretenden ser usados en otro proyecto, el modelo debería ser entrenado en una máquina con mayor capacidad computacional mediante la cual se pueda entrenar el modelo durante más tiempo con métodos más avanzados, como alguno de los que hoy en día están considerados como parte del estado del arte en este ámbito.

Abstract:

Currently, multiple machine learning models have been implemented for sentiment analysis that have the ability to classify text according to whether it is positive or negative, both individual words and complex sentences. However, the models with the highest hit rates have required high computational power to classify the text in question and also to be constantly updated with more examples.

In this case, the aim is to classify the polarity of offensive comments on social networks, specifically on Instagram and directed towards professional footballers.

Therefore, the objectives of this study have been defined firstly as the autonomous collection of data and the creation of a dataset to then train models.

Following this thread, the next objectives are to investigate the different methodologies, technologies and models of the Python machine learning library, scikit-learn. Finally, after making a comparison between the 5 selected models, one of these models will be chosen to determine the polarity of the comments previously extracted by sentiment classification (“sentiment analysis”).

Despite the low level of personal knowledge available in the field of Natural Language Processing at the beginning, and the lack of computational capacity, the results of the model can be considered satisfactory, since a coherent classification based on a well-founded justification is being obtained.

However, if the initial planning had been more accurate, the results could have been improved and if these data are intended to be used in another project, the model should be trained on a machine with higher computational capacity by which the model can be trained for a longer time with more advanced methods, such as some of those that are nowadays considered as part of the state of the art in this field.

Agradecimientos:

Me gustaría agradecer en primer lugar a mi pareja, por su ayuda, porque me ha estado apoyando y animando en todo lo relacionado con este trabajo y haciéndolo más ameno desde el primer momento hasta el último. En segundo lugar, a mis padres que han hecho que este proceso sea más fácil de llevar. También a mis amigos que me han aportado ideas y motivación durante el proyecto. Por último, agradecer a mi tutor de TFG Santi Seguí que contribuyó a que este trabajo cumpla las expectativas y pueda ser entregado correctamente.

Acrónimos:

BERT “Bidirectional Encoder Representations from Transformers”.

BoW “Bag of Words”.

ELMo “Embeddings from Language Models”.

GloVe “Global Vectors for Word Representation”.

LSTM “Long Short-Term Memory”.

ML “Machine Learning”.

NLP “Natural Language Processing”.

ReLU “Rectified Linear Unit”.

TF-IDF “Term Frequency - Inverse Document Frequency”.

VADER “Valence Aware Dictionary for sEntiment Reasoning”.

Índice:

1 Introducción

Para hacer una introducción al problema que será abordado, es realizada una contextualización de todo lo relacionado con éste, para después ser expuestas las motivaciones que me han llevado a realizar el presente trabajo y, para acabar, quedarán definidos los objetivos a ser cumplidos.

1.1 Contextualización

El Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés) es un subcampo de la inteligencia artificial y la lingüística que se encarga de estudiar cómo los ordenadores pueden entender, interpretar y generar lenguaje humano de manera automática.

A continuación, se adjunta una imagen descriptiva de la interacción entre los diferentes subcampos de la inteligencia artificial, que, como es de esperar interseccionan entre ellos creando así la capacidad de potenciarse unos a otros.

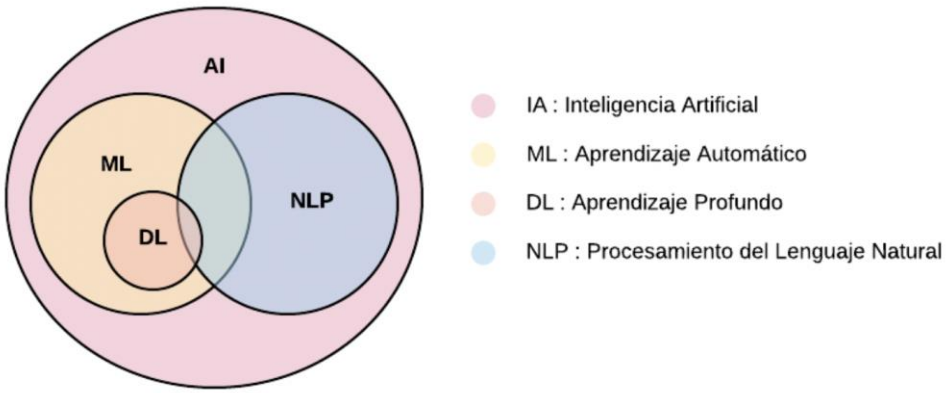


Figura 1: *Diagrama donde se muestran las relaciones entre subcampos*

En los últimos años, el NLP ha avanzado significativamente gracias al aprendizaje profundo (Deep Learning), que ha permitido crear modelos de lenguaje más complejos y precisos para tareas como el análisis de sentimientos, la traducción automática o el reconocimiento de voz, entre otras.

Además, el NLP se está aplicando cada vez más en el procesamiento de grandes cantidades de datos de texto, lo que está impulsando su uso en campos como la minería de datos, el análisis de redes sociales o la clasificación automática de documentos, entre otros. Está haciendo que grandes empresas como Google, Amazon, Facebook y Microsoft están invirtiendo fuertemente en esta tecnología para mejorar la experiencia de usuario y la eficiencia en sus productos y servicios.

Como he mencionado anteriormente en este proyecto se investigarán 5 modelos para realizar la tarea de análisis de sentimientos que se utilizará para identificar, extraer y cuantificar las emociones y opiniones expresadas en un texto, en este caso comentarios hacia jugadores de fútbol. El objetivo principal es determinar el tono general del texto, ya sea positivo, negativo o neutral, en relación con un tema específico. De esta forma, en futuras implementaciones se podrán detectar comentarios ofensivos de forma automática mejorando la convivencia en redes [1].

Para realizar el análisis de sentimientos, se utilizan técnicas de minería de texto y aprendizaje automático. En general, se emplean dos enfoques principales: el enfoque basado en reglas y el enfoque basado en aprendizaje automático. El primero se basa en reglas predefinidas, como listas de palabras positivas o negativas, y utiliza estas reglas para clasificar el texto en categorías de sentimientos. El segundo enfoque se basa en el entrenamiento de modelos de aprendizaje automático utilizando conjuntos de datos etiquetados para identificar patrones en el texto y predecir la polaridad de nuevos textos [2].

Sea como sea el enfoque que queramos darle al uso de NLP, se sigue el siguiente procedimiento en cuanto al tratamiento de datos de modo general.

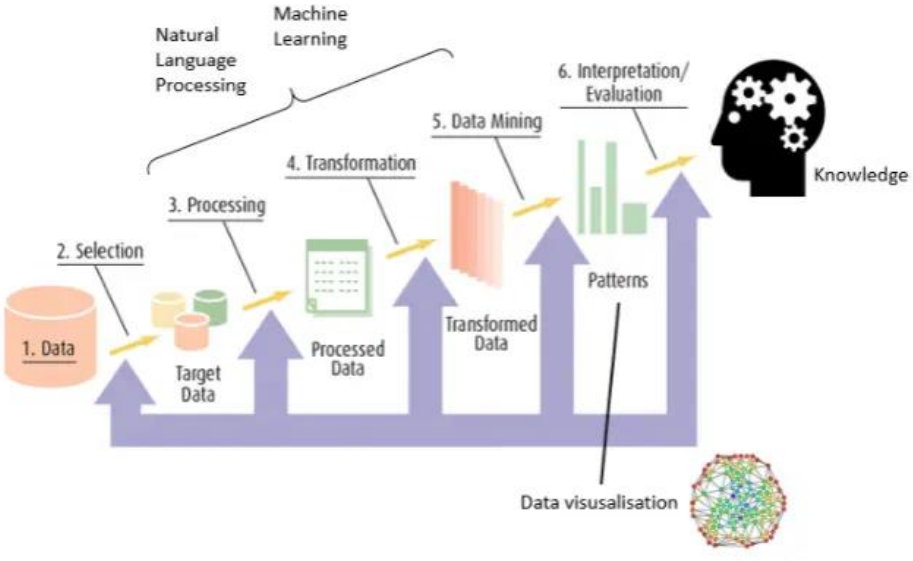


Figura 2: *Dinámica de procesamiento y modelización de datos de NLP*

1.2 Objetivos y motivación

Los objetivos que han sido propuestos para el presente trabajo están definidos por:

- En primer lugar, recopilar datos relevantes para crear un dataset que permita el análisis de sentimientos. Este dataset se utilizará para desarrollar y evaluar los diferentes modelos a implementar.
- En segundo lugar, la investigación de 5 diferentes modelos de la scikit-learn y la implementación de estos modelos de aprendizaje automático mediante los cuales se pueda determinar la polaridad de comentarios de instagram dirigidos a futbolistas profesionales.

En términos de motivación para la realización de este trabajo debe ser destacado principalmente:

- El interés por el funcionamiento de NLP y los diferentes modelos de redes neuronales.
- Haber trabajado con grandes fuentes de datos anteriormente y tener curiosidad por ver cómo pueden ser tratadas.
- El interés por ver las aplicaciones de modelos NLP en casos reales en los que sea pueda sacar partido a su utilidad.
- Esperanza que estos modelos creados sean escalables y aplicables a casos reales en los que mejore la convivencia en redes. Por ejemplo, utilizarlos como filtro en los comentarios de redes sociales evitando o restringiendo ciertos comentarios ofensivos y haciendo así mejorar la convivencia entre usuarios de una misma comunidad.

En resumen, como colofón a lo expresado anteriormente, debe ser comentado que este proyecto ha sido desarrollado de forma paralela a el aprendizaje de programación enfocada a web, durante la jornada laboral. A causa de esto y al conocimiento adquirido durante el curso universitario, junto con eventos externos que han permitido ver realmente las aplicaciones reales de diferentes tipos de tecnologías y como pueden ser combinadas para crear algo mayor, entre otros, espero que este trabajo pueda ser ampliado y continuado en un futuro. Esto también sería considerado como una de las grandes motivaciones para el desarrollo de este proyecto.

1.3 Planificación - Diagrama de Gantt (2022)

Actividades	2022															
	Septiembre				Octubre				Noviembre				Diciembre			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Elección del tema																
Diseño y planificación																
Introducción - doc																
Búsqueda dataset prefabricado																
Creación dataset																
Limpieza dataset																
Estado del arte 1/2 - doc																
Estado del arte 2/2 - doc																

Actividades	2023																							
	Enero				Febrero				Marzo				Abril				Mayo				Junio			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Investigación modelos																								
Materiales - doc																								
Métodos - doc																								
Implementación vectorizadores y preprocesado																								
Creación dataset - doc																								
Vectorizadores y train_split - doc																								
Implementación modelos																								
Implementación modelos - doc																								
Finalización memoria																								

1.4 Costes

En este apartado, se especifican los costes que ha requerido la realización de este trabajo. En este caso, al tratarse de una investigación y una implementación sobre la que hay mucha información, lo esencialmente necesario se podía encontrar sin ningún coste adicional en la red. Pese a esto, se han utilizado herramientas web con una suscripción gratuita que estaba limitada a un período de tiempo, pero al haber realizado la tarea dentro de este período, el coste ha sido finalmente gratuito.

El único coste remarcable, sería el gasto de luz durante la realización de este trabajo en su totalidad. En menor medida, durante la documentación del proyecto y en mayor medida durante la implementación, ya que el programa podía estar calculando hasta 1 día debido a la gran cantidad de datos a procesar y a la capacidad computacional del pc.

Estructura del trabajo:

El presente documento ha sido estructurado según el siguiente esquema:

1. Introducción: De manera introductoria al problema que ha sido abordado, se realiza una contextualización de todo lo relacionado con éste, para después ser expuestas las motivaciones para la realización del presente trabajo y se definirán los objetivos propuestos. Finalmente, se adjuntará un diagrama de Gantt para mostrar la organización del trabajo en el tiempo y se comentarán los costes.

2. Estado del arte: Es expuesto el último nivel de desarrollo, en el momento presente, de las metodologías con las cuales pretende ser solucionado el problema a tratar, mediante el estudio y exposición de los artículos científicos que fueron escritos sobre la utilización de modelos NLP para la clasificación de comentarios ofensivos.

3. Materiales y métodos: En esta sección son descritos la totalidad de los materiales que han sido utilizados para la realización del presente estudio, seguido de las metodologías y procedimientos que se han realizado. Esta exposición se realiza para que pueda ser valorada la idoneidad de la investigación y validez de los resultados.

4. Implementación: En esta sección será descrito el procedimiento utilizado para implementar el modelo, basándose en las metodologías y materiales disponibles, que se han mencionado en el apartado 3, para la clasificación de los comentarios ofensivos encontrados en la red social “Instagram” tal y como ha sido expuesto anteriormente.

5. Resultados y discusión: En esta sección estarán reportados los resultados de la investigación que se ha realizado mediante los materiales y tecnologías vistos anteriormente. Los resultados serán expuestos de forma completamente objetiva, con el objetivo de que pueda ser completado el objetivo inicial o, por contra, que deban ser consideradas nuevas perspectivas sobre cómo enfocar el problema a través de las cuales pueda ser realizada una nueva investigación.

6. Conclusiones: Aquí se exponen y reiteran los puntos principales de la investigación a modo de resumen, juntamente con las conclusiones que se han obtenido a través de un análisis de los resultados obtenidos anteriormente. Gracias a esta exposición, se llegará a nuevas consideraciones, tanto en forma de problemas que se han ido encontrando a lo largo del proyecto como de sus respectivas soluciones, juntamente con unas posibles futuras extensiones del presente trabajo.

2 Estado del arte

Uno de los principales retos del NLP es la propagación eficaz de los conocimientos o significados derivados de una parte de los datos textuales a otra. Por ejemplo, en una frase como "The animal did not cross the street because it was tired", para que el modelo entienda a qué se refiere la palabra "it" (el animal, no la calle), el significado procesado para la palabra "animal" debe ser recordado por el modelo y retomado con precisión en el momento necesario al tratar la palabra "it".

Las herramientas de NLP han evolucionado desde las redes neuronales recurrentes “vanilla” (programas o sistemas operativos que no han sufrido cambios, adaptaciones o actualizaciones con respecto a su versión original) o RNN, hasta las Long Short-Term Memory Networks (LSTM) o redes de memoria a largo plazo, pasando por las arquitecturas del modelo "Transformer" y sus diversas variantes, así como algunos modelos nuevos y revolucionarios.

2.1 Preprocesamiento del texto

El preprocesamiento de texto es una tarea importante en NLP que consiste en transformar el texto en un formato que facilite el funcionamiento de los algoritmos que implementan los modelos de aprendizaje. Esto se consigue, principalmente, mediante la eliminación de las palabras de parada o stopwords, stemming, lemmatization y algoritmos de TF-IDF o expresiones regulares entre otros.

Una de las primeras aproximaciones a este problema es utilizar “Bag of Words” (BoW), que consiste en “tokenizar” los documentos, transformándolos en listas de palabras. Este tipo de aproximación desecha el orden de las palabras y otros factores, y le da importancia, únicamente, al número de ocurrencias de cada una de las palabras. Es decir, genera un diccionario (clave, valor) en el que a cada palabra le asocia su multiplicidad. Después, se aplicaría un algoritmo de ML que utiliza los datos generados por BoW para clasificar los documentos. Uno de los problemas de BoW (sin contar que no tiene en cuenta las relaciones semánticas, la dimensión de los datos, etc.), son los documentos largos. Esto se debe a que este tipo de documentos elevan el número de ocurrencias de algunas palabras, pudiendo influir en la precisión del algoritmo en cuestión.

Teniendo en cuenta este problema, para encontrar una solución a corto plazo sería utilizar TF-IDF (Term Frequency - Inverse Term Frequency), que normaliza la frecuencia del término en el documento (TF) multiplicándola por el logaritmo natural de la inversa de la frecuencia de los documentos en los que aparece el término (IDF).

Después de aplicar este algoritmo, debería ser aplicado un algoritmo de ML que utilice los datos generados por TF-IDF para clasificar los documentos. Uno de los problemas de TF-IDF es que, como BoW, tampoco tiene en cuenta relaciones semánticas y por

lo tanto se está perdiendo información útil para poder aprender de estas relaciones. Veremos soluciones a esto en los siguientes apartados.

2.2 Enfoques lingüísticos

Los enfoques lingüísticos se centran, en general, en aplicar una serie de reglas creadas por expertos lingüistas. Este tipo de enfoque tiene la ventaja de que, a diferencia de los enfoques basados en ML, no necesita grandes cantidades de datos [3].

Un ejemplo de estos modelos sería “Valence Aware Dictionary for Sentiment Reasoning” (VADER), que utiliza una combinación de métodos cualitativos y cuantitativos para construir una lista de características léxicas relacionadas con el análisis de sentimiento. Como podemos ver en este tipo de modelos no requieren un gran coste computacional, pero sí un gran coste humano. Es decir, los expertos tienen que generar el conocimiento fundamental que se le da al modelo (y esto se hace de forma manual)[3].

También, comentar, la existencia de herramientas con información lingüística como, por ejemplo, los recursos léxicos (“lexical resources”). Un ejemplo de recurso léxico sería “WordNet”, que es una gran base de datos léxica de relaciones semánticas entre palabras, la cual se puede utilizar para visualizar datos a la hora de la implementación como veremos más adelante, (está disponible en más de 200 idiomas, pero en su origen era del inglés). Este tipo de recursos contiene información semántica que puede ser utilizada para entrenar distintos modelos [4].

2.3 Enfoques basados en ML

Generalmente, este tipo de enfoques tiene una base estadística (como la inferencia estadística) [5].

Como hemos observado anteriormente, uno de los problemas de TF-IDF es que no tiene en cuenta relaciones semánticas. Con el objetivo de representar estas relaciones semánticas entre las palabras, podríamos utilizar alguna técnica de “word embedding”, cuya aplicación es representar cada palabra mediante una lista de valores. Por ejemplo, “word2vec” [6]. Este método consiste en, dada una palabra, fijar un tamaño de ventana y emparejarla con las palabras que se encuentran antes o después a una distancia máxima del tamaño de ventana escogido. Formando de esta manera, para una misma palabra (palabra objetivo), varias parejas con diferentes palabras (palabras de entrada). Las parejas de palabras, palabra entrada, palabra objetivo, son dadas a un algoritmo que trata de predecir la palabra objetivo a partir de la palabra de entrada. De esta manera, el algoritmo de ML aprende sobre las relaciones entre palabras.

En 2014, surgió un nuevo método para crear "word embedding" denominado "Global Vectors for Word Representation" (GloVe). GloVe utiliza una matriz de co-ocurrencia palabra-palabra del corpus de texto. Cada entrada en la matriz representa el número de veces que una palabra "j" aparece en el contexto de la palabra "i". El proceso de entrenamiento se realiza utilizando los datos de esta matriz, junto con algunas operaciones adicionales que se describen en detalle en [7].

Como hemos comentado, “word embedding” empezó a conseguir relacionar semánticamente las palabras. Aun así, continuaba teniendo el problema de no tener en cuenta el orden de las palabras ni el contexto en el que se encuentran. Es decir, es una representación independiente del contexto. LSTM es un tipo de red neuronal recurrente capaz de aprender dependencias a largo plazo que veremos explicada más adelante.

Otra manera de manejar las dependencias a largo plazo es utilizar un modelo “Transformer”, que apareció en 2017. Dando lugar a un modelo más paralelizable y con un tiempo de entrenamiento menor. Es muy bueno en problemas de secuencia a secuencia (“seq2seq”), como, por ejemplo, tareas de traducción. Este modelo puede hacer frente a problemas de dependencias a largo plazo mejor que el LSTM.

En octubre de ese mismo año, apareció BERT. BERT destrozó los récords previos en múltiples conjuntos de datos de NLP, como, por ejemplo, en el “Standford Question Answering Dataset”. El cual consiste, básicamente, en responder preguntas de acuerdo a un texto de Wikipedia (comprensión lectora).[8]

BERT utiliza la transferencia de aprendizaje. También utiliza la estructura de un “Transformer”, concretamente, la parte del codificador del “Transformer”.

Para hacerlo bidireccional, BERT no utiliza un modelo “backward” como LSTM o ELMo, sino que añade el ocultamiento de palabras (“word masking”). Esta técnica consiste en ocultar un porcentaje de las palabras de la entrada (en [8] este porcentaje es un 15 %) sustituyéndolas por el token “[MASK]”. Después, pasan la frase por la arquitectura y tratan de predecir únicamente las palabras ocultas. Esta aproximación es considerada como “Deep Bidirectional Transformer Encoder”.

2.4 Análisis de las diferentes arquitecturas NLP

Hoy en día contamos con diferentes metodologías y arquitecturas basadas en NLP que han ido evolucionando desde el siglo pasado hasta la actualidad y que han ido obteniendo cada vez más usos, a continuación se expondrán un subconjunto de las arquitecturas que son aplicadas con mayor frecuencia o que tienen más relevancia dentro del procesamiento natural de texto, siendo éste el tema en concreto en el cual se ha indagado [9].

2.4.1 Red neuronal recurrente (RNN)

Las redes neuronales recurrentes son variantes de las redes neuronales regulares totalmente conectadas que tienen memoria en sus modelos. Las RNN son recurrentes, ya que realizan la misma función para cada punto de datos de entrada, y la salida para una entrada dada depende de los datos procesados de la celda anterior. Después de producir la salida, se copia y se envía de nuevo a la red recurrente. Para tomar una decisión, tiene en cuenta la entrada actual y la salida que ha aprendido de la entrada anterior.

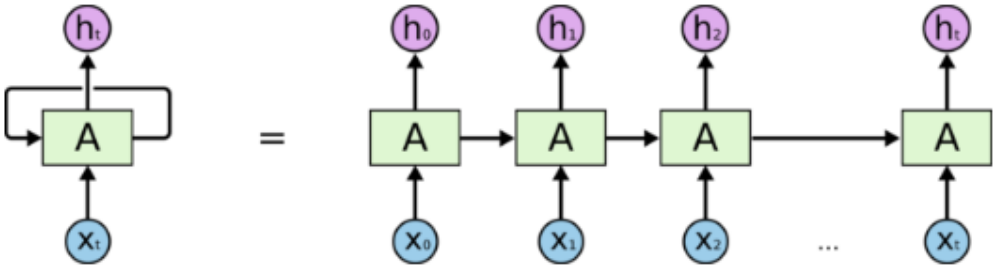


Figura 4: *Esquema de funcionamiento una RNN*

2.4.2 Long Short-Term Memory (LSTM)

LSTM es un tipo de red neuronal recurrente capaz de aprender dependencias a largo plazo. Es adecuada para clasificar, procesar y predecir series temporales con desfases de duración desconocida. Tiene tres puertas a través de las cuales se procesa la información: la puerta de entrada, la puerta de olvido y la puerta de salida, con dos elementos que se procesan en cada celda: los datos de la celda oculta y la salida de la celda.

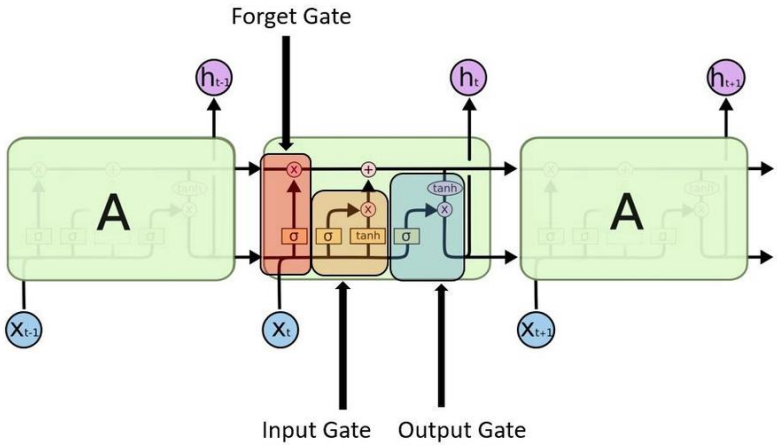


Figura 5: *Esquema de funcionamiento una red LSTM*

2.4.3 Attention Mechanism

El artículo "Attention Is All You Need", publicado por Google Brain y Google Research, presentaba técnicas para la computación paralela masiva en el hardware TPU (Tensor Processing Unit) de Google [10].

Este “transformer” utiliza este mecanismo para codificar información en su vector de palabras sobre el contexto relevante de una palabra determinada. En el siguiente caso antes de traducir al francés mira las representaciones de cada palabra que se ha procesado hasta el momento. Este proceso multiplica el rendimiento del modelo.

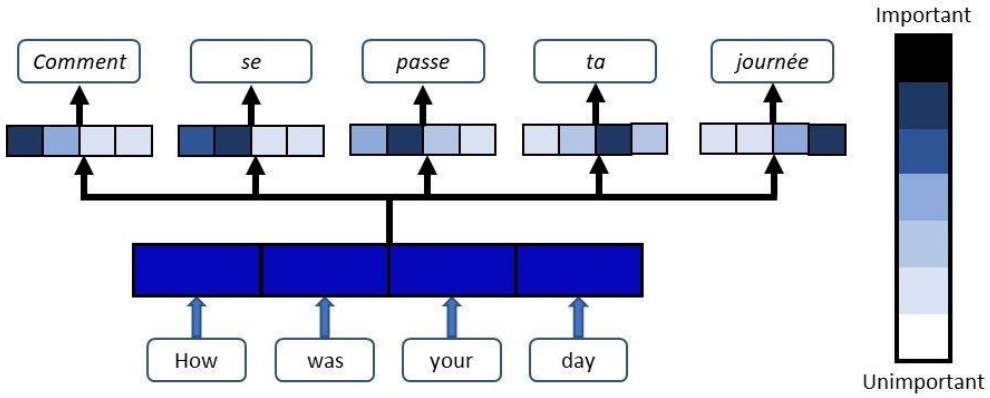


Figura 6: *Esquema de funcionamiento de Attention Mechanism*

2.4.4 Bidirectional Encoder Representations from Transformers (BERT) model

El modelo BERT es un tipo de modelo de transfer learning que puede utilizarse para diversas tareas fine-tuneando un modelo pre-entrenado. Más adelante, veremos más en profundidad el funcionamiento de este modelo. A continuación, vemos algunas tareas para las que se puede fine-tunear el modelo pre-entrenado BERT.

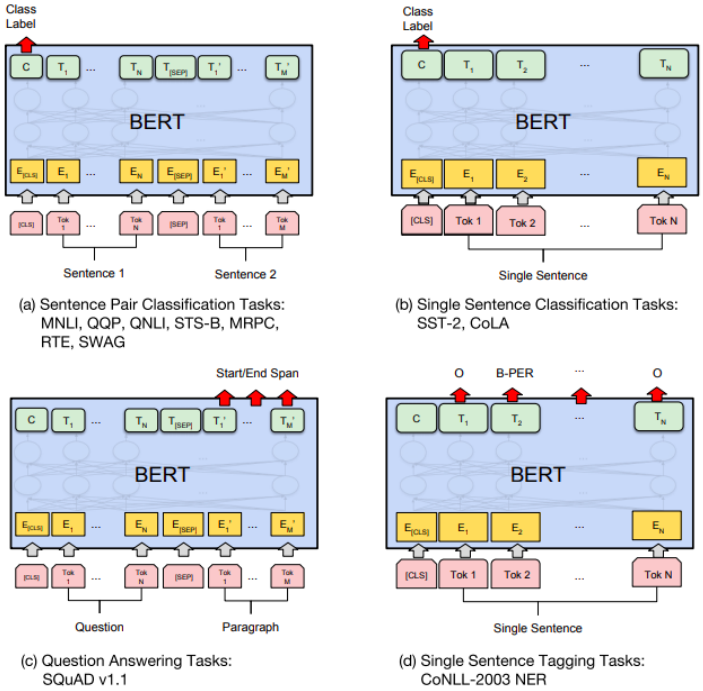


Figura 7: *Ejemplos de fine-tuning y esquema de funcionamiento del modelo BERT*

2.4.5 Transformer-XL

Se diseñó para utilizar y reutilizar los estados ocultos obtenidos en el modelo en segmentos anteriores para que sirvieran de memoria para el segmento actual sobre el que se está operando con el fin de construir una conexión recurrente entre los segmentos. Este enfoque permite aprender dependencias más allá de una longitud fija de palabras. Con la información pasada de segmentos anteriores, también se resuelve el problema de la fragmentación del contexto.

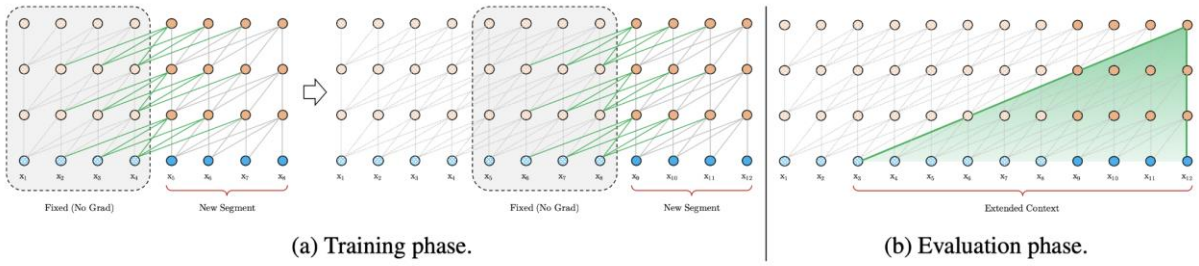


Figura 8: *Modelo Transformer-XL con un segmento de longitud 4*

2.4.6 Compressive Transformers

El Transformador Compresivo es una extensión del Transformer que, de forma similar, mapea las activaciones ocultas pasadas generadas en el modelo a un conjunto más pequeño de representaciones comprimidas, que son en realidad memorias comprimidas. Utiliza el mismo mecanismo de atención sobre su conjunto de memorias y memorias comprimidas, aprendiendo a consultar tanto su memoria granular a corto plazo como su memoria gruesa a largo plazo, y consigue resultados significativamente mejores.

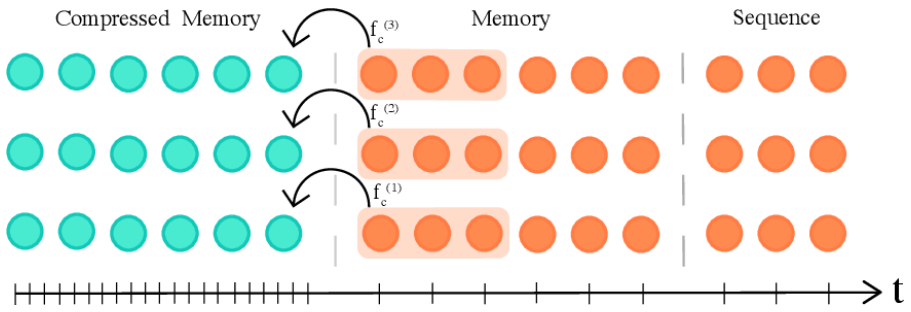


Figura 9: *Compressive transformer consultando su memoria granular y memoria comprimida*

2.5 Conclusiones

Habiendo realizado la enumeración y breve explicación de algunas de las arquitecturas basadas en NLP juntamente con sus usos, el presente trabajo estará basado en la investigación de 5 modelos de aprendizaje automático supervisado, seguido de la implementación de éstos y finalmente la elección de uno de ellos para afrontar el problema de clasificación de sentimientos. También se verá el modelo mencionado anteriormente, BERT, pero al haber explicado ya su funcionamiento y su sencilla implementación no se explicará en siguientes apartados de la memoria.

3. Materiales y métodos

En esta sección se describirán ordenada y detalladamente la totalidad de los materiales que se han utilizado para la realización del presente trabajo, en conjunto con las metodologías y procedimientos que se han llevado a cabo. Esta exposición se realiza para que pueda ser evaluada la genuinidad, la fiabilidad y validez de los resultados, además que, en un futuro, pueda ser replicada la investigación que se ha llevado a cabo en este proyecto.

3.1 Materiales

Aquí se expondrán de forma concreta aquellas herramientas de software, recursos de computación y la metodología necesaria para entender el funcionamiento de las tareas realizadas.

3.1.1 Data collection

La idea inicial para recolectar los datos que luego utilizaría para entrenar mis modelos estaba compuesta de dos partes. En la primera parte se recopilaron links de las 30 primeras imágenes de cada uno de los 25 jugadores de fútbol. En la segunda parte se utilizarían estos links para ir iterando por cada uno de los posts y extrayendo comentarios de usuarios para formar el dataset.

3.1.1.1 Apify

Inicialmente, comencé buscando la manera más eficiente para recolectar datos de Instagram y después de encontrar diferentes sitios web que permitían conseguir información me acabé decantando por la API de visualización básica de Instagram para desarrolladores que se puede encontrar en la página web de “Meta for developers”.

Me acabé decantando por esta opción ya que creí que sería la manera más eficiente y de mayor fiabilidad para recolectar datos. Al intentar realizar una implementación experimental a través de una función en código para extraer comentarios de esta red social y de seguir una serie de pasos en su guía, observé que requería tener una página web propia o formar parte de alguna organización para poder utilizarla, aunque me creara la cuenta luego intenté hacer llamadas desde mi notebook y me fallaba constantemente. Después de investigar e ir avanzando y retrocediendo constantemente y al no disponer de excesivo tiempo, decidí cambiar de estrategia para completar esta parte del proyecto.

Como alternativa al problema anterior, la herramienta de programación para recolectar datos que ha formado parte del objetivo de crear un conjunto de datos que

contiene comentarios extraídos de instagram ha sido Apify. Esta plataforma online proporciona herramientas para el web scraping de distintas direcciones web como es el caso de Instagram. Lo único que fue necesario hacer en esta web fue encontrar un endpoint adecuado para recibir la información que necesitaba, que en este caso era la información de 30 posts de instagram de cada jugador de fútbol y generar un token para poder realizar solicitudes desde mi código en Jupyter Notebook, en el fichero llamado `"Instagram_API_comments.ipynb"`.

A continuación, adjunto una captura de pantalla de ejemplo de uno de los correos electrónicos que enviaba Apify como justificante/información de que el proceso se ha llevado a cabo correctamente y adjuntando un archivo CSV con los resultados obtenidos tras hacer las solicitudes a la API desde mi código.

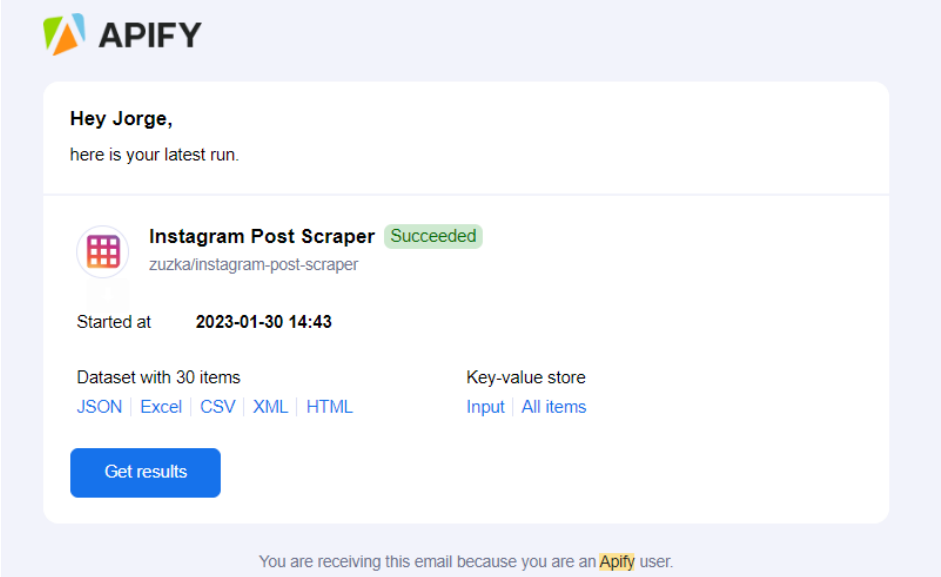


Figura 10: *Correo de confirmación de peticiones API con csv adjunto*

3.1.1.2 Selenium y BeautifulSoup

Durante mi período de prácticas en empresa aprendí a utilizar estas dos librerías para hacer web scraping y creí que para el trabajo a realizar eran suficientemente adecuadas. Por eso la segunda parte de la recopilación de datos consiste en, habiendo recibido la información de los 30 posts de cada jugador, seleccionar únicamente los enlaces de publicaciones de instagram (cada publicación tiene un enlace único formado por un conjunto de caracteres). Una vez recopilados los enlaces, hacer uso de la librería selenium para acceder a una instancia del navegador Google Chrome e iterar por cada uno de estos enlaces y gracias a la combinación con BeautifulSoup poder así extraer los comentarios de usuarios en estas publicaciones y almacenarlos en un dataframe de pandas, librería que ha sido utilizado en diferentes ocasiones durante este proyecto.

3.1.2 Preprocessing

Para este apartado se ha utilizado en gran parte el kit de herramientas de lenguaje natural, o más comúnmente NLTK. Es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural simbólico y estadísticos para el lenguaje de programación Python. NLTK incluye demostraciones gráficas y datos de muestra.

Se ha utilizado principalmente durante la implementación para dar el formato adecuado a un dataset de prueba, inicialmente, y al dataset creado durante este proyecto con la finalidad de aprender a utilizar estas funciones, pero como veremos más adelante en este proyecto al disponer de un dataset compuesto de comentarios que su longitud de texto no es muy extensa no se puede explotar esta funcionalidad tanto como me hubiera gustado.

3.1.3 Almacenamiento de datos y modelos con Pickle

Durante este proyecto se ha ido guardando el trabajo progresivamente en "checkpoints" por decirlo de alguna forma, tanto de datasets como de modelos. El motivo de esto es básicamente que, como se trabaja con cantidades significativamente grandes de datos y de procesos sobre estos datos, que suelen tener una duración extensa, decidí ir guardando el progreso.

Para esta tarea, he utilizado Pickle, un módulo de Python para la serialización y deserialización de objetos para guardar tanto los modelos como los datasets en formato ". pickle". Éstos los creaba durante el proceso de data collection en conjunto con la librería pandas para guardarlos como .csv.

3.1.4 Lenguaje de programación Python

La implementación del código fuente estará escrito en el lenguaje de programación Python. La elección de este lenguaje antes que otras opciones disponibles ha sido mayoritariamente debido a su estrecha relación con Machine Learning ya que se dispone de multitud de librerías y módulos mediante los cuales se permiten desarrollar modelos y programas dentro de este ámbito con mucha facilidad siendo especificadas únicamente las capas más altas de abstracción y delegando el uso de distintos algoritmos o métodos matemáticos de la misma librería. Otro de los motivos para escoger Python es la simplicidad sintáctica de este lenguaje que deriva en una más alta velocidad de código comparándolo con otros lenguajes [11].

3.1.4.1 PyTorch

Dentro de todo el abanico de librerías de programación para Python, durante la implementación del modelo BERT se ha elegido PyTorch, debido a, entre otros motivos, es una biblioteca basada en Torch que se adapta muy bien a este proyecto ya que dispone de diferentes aplicaciones dentro del ámbito de NLP y la documentación de esta biblioteca está actualizada hoy en día.

3.1.4.2 Transformers

Transformers es una biblioteca de Huggin Face que proporciona APIs para descargar y entrenar fácilmente modelos pre entrenados como es el caso de BERT. Esta biblioteca permite una integración entre Pytorch y TensorFlow que son las librerías utilizadas durante la implementación del modelo comentado.

3.1.4.3 Sci-kit learn

Scikit-learn es una biblioteca para Python que incluye varios algoritmos de clasificación, regresión y análisis. Para este proyecto se ha utilizado desde la selección y preprocesado de los datos y de los modelos hasta para el análisis y visualización de datos.

3.1.4.4 Pandas, matplotlib y tqdm

Pandas es una librería especializada en la manipulación y el análisis de datos. Durante la realización de este trabajo, desde el momento inicial, esta librería ha sido utilizada en el manejo de tablas de datos, durante el preprocesamiento de datasets e incluso para guardar estas tablas de datos como archivos .csv y de la misma forma leerlos posteriormente.

Matplotlib, por su parte, es utilizado para la visualización de los conjuntos de datos y análisis de éstos facilitando la comprensión.

Por último, tqdm ha sido una herramienta útil durante el proceso de data collection y durante el proceso de limpieza y traducción del dataset en el notebook:

(“*clean_translate_dataset.ipynb*”)

3.2 Metodologías

3.2.1 Deep learning

El Deep Learning es un subconjunto de aquellos algoritmos de Machine Learning a través de los cuales una inteligencia artificial puede ser entrenada para la realización de una tarea específica.

Este entrenamiento, al ser llevado a cabo mediante un sistema de redes neuronales de múltiples capas, puede ser completamente no supervisado por ningún ser humano en ningún momento, salvo que se produzca algún error inesperado y haya que gestionarlo, siendo delegadas al mismo algoritmo todas las tareas de predicción y análisis de los resultados.

3.2.2 Función de activación

Está función se ha visto durante la implementación del modelo BERT ya que para el modelo “tradicional” por llamarlo de alguna forma, de la librería de Sci-kit learn, no se ha requerido utilizar ya que el modelo viene dado con todas estas funciones en capas más bajas de abstracción a las cuáles no es necesario acceder.

La incapacidad de realización de cálculos complejos es tomada como una característica intrínseca de la naturaleza de una red neuronal completamente lineal, por la cual cosa se reduce considerablemente el abanico de posibilidades de ésta. La única manera de que una red neuronal sea capacitada para la realización de tareas y cálculos más complejos es la adición de componentes no lineales a ésta, como por ejemplo las funciones de activación. En este caso se ha utilizado la función softmax para BERT y las especificadas en el apartado de implementación para cada uno de los 5 modelos a estudiar.

3.2.3 Softmax

La función utilizada como capa final del modelo BERT implementado es Softmax que convierte un vector de K valores reales en un vector de K valores reales que suman 1. Es decir, que los valores de entrada pueden ser positivos, negativos, cero o mayores que uno, pero la función softmax los transforma en valores entre 0 y 1, de modo que puedan interpretarse como probabilidades. Si una de las entradas es pequeña o negativa, softmax la convierte en una probabilidad pequeña, y si una entrada es grande, entonces la convierte en una probabilidad grande, pero siempre permanecerá entre 0 y 1.

Su fórmula es muy similar a la función sigmoidea que se utiliza para la regresión logística. La función softmax puede utilizarse en un clasificador sólo cuando las clases son mutuamente excluyentes como sería el caso.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (3)$$

Donde:

- $\vec{z}$ = Vector de entrada a la función softmax
- z_i = Elementos del vector de entrada a la función softmax
- e^{z_i} = Función exponencial estándar que se aplica a cada elemento del vector de entrada
- $\sum_{j=1}^{K} e^{z_i}$ =Término de normalización, garantiza que los valores de salida sumen 1 y estén entre 0 y 1.
- K = Número de clases del clasificador

Se verá alguna fórmula de activación más adelante, pero se explicará en relación con la implementación.

3.2.4 Term Frequency - Inverse Document Frequency (TF-IDF)

Para explicar un poco más a fondo el funcionamiento de TF-IDF, se seguirá la explicación de [12].

Para empezar, como ya se ha mencionado en el apartado 2, TF-IDF representa “Term Frequency - Inverse Document Frequency”. Asumiendo que disponemos de una colección de N documentos, y que el término t_i aparece en n_i de ellos. Entonces la frecuencia inversa de documento se puede calcular como:

$$\text{idf}(t_i) = log\frac{N}{n_i} \quad (4)$$

Si el término no está en la colección se producirá una división-por-cero. Por lo tanto, es común ajustar esta fórmula a $1+|\{d \in D : t \in d\}|$. Por otro lado, dado un término t_i, denotamos por tf(i) la frecuencia de t_i en el documento bajo consideración del artículo [12].

Finalmente, TF-IDF se define para un término t_i en un documento dado como el producto de TF por IDF. Es decir:

$$tfidf(t_i) = tf_i \cdot idf(t_i) \quad (5)$$

Con el factor $idf(t_i)$ conseguimos que los documentos largos no eleven la cuenta de las palabras, y de esa manera puedan influir en la precisión del algoritmo, como bien se señala en el apartado 2 de este trabajo.

3.2.5 Bidirectional Encoder Representations from Transformer (BERT)

Esta sección se centrará en describir de la manera más detallada posible la arquitectura del modelo BERT y daremos algunas nociones básicas de su funcionamiento. Para ello, se seguirán entre otras fuentes las referencias [8], [10].

Tal y como se explica en el artículo [8], BERT tiene dos pasos en su esquema de funcionamiento: preentrenamiento y reajuste ("fine-tuning"). El preentrenamiento consiste en entrenar al modelo sobre grandes corpus sin etiquetar. El reajuste consiste en inicializar el modelo con los parámetros del preentrenamiento, y ajustar estos parámetros utilizando conjuntos de datos etiquetados correspondientes a tareas específicas, en este caso, un conjunto de datos etiquetados con 0 o 1 dependiendo si el comentario es ofensivo o no.

Cada una de estas tareas específicas tiene modelos reajustados separados, aunque todos ellos son inicializados con los mismos parámetros del preentrenamiento tal y como se ha realizado en este proyecto:

```
"bert = AutoModel.from_pretrained('bert-base-uncased')"
```

Una de las características distintivas de BERT es que tiene una arquitectura unificada para las distintas tareas específicas. La diferencia entre la arquitectura de preentrenamiento y la de la tarea específica es mínima.

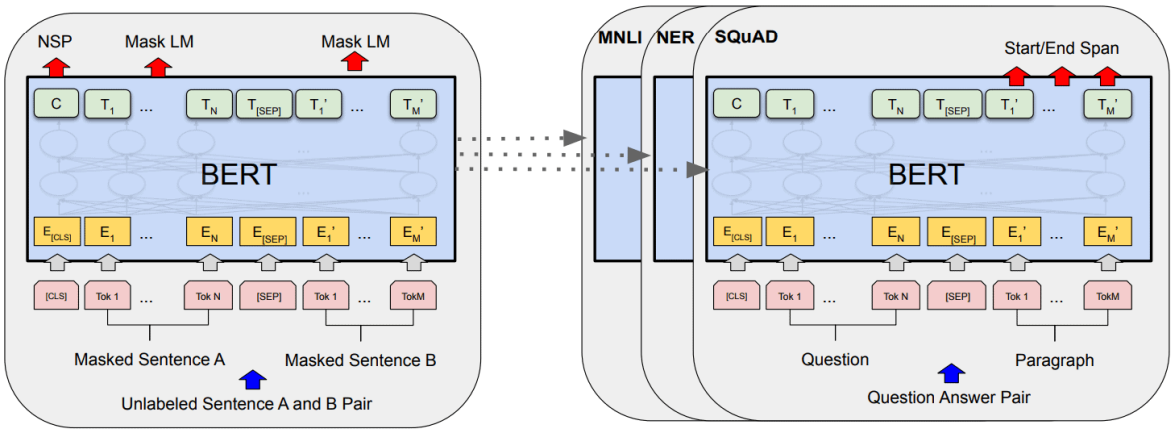


Figura 11: *Procedimientos de entrenamiento y reajuste*

3.2.5.1 Arquitectura del modelo

La arquitectura de BERT es la de un "Transformer" codificador bidireccional multicapa [8].

Y, dado que en el artículo referenciado [8] se refieren a para ver los detalles de la arquitectura del modelo diciendo que su implementación es casi idéntica, empezaré explicando en qué consiste esta arquitectura siguiendo la referencia original [10].

Las arquitecturas codificador-decodificador consisten en un codificador que mapea una secuencia de representaciones simbólicas a una secuencia de representaciones continuas (de uno en uno). En cada paso, el modelo es auto-regresivo, es decir, que utiliza los símbolos generados en los pasos previos como entrada extra para calcular el símbolo siguiente. Los "Transformer" siguen esta arquitectura general, utilizando capas totalmente conectadas ("fully connected layers") con mecanismos de auto-atención ("self-attention") y aplicados a cada posición ("position-wise"), tanto para el codificador como para el decodificador.

En este caso, se ha intentado crear una arquitectura del modelo BERT aprovechando que es "customizable" tal y como se puede ver en:

`"BERT-fine-tuning-and-evaluation.ipynb".`

3.2.5.2 Codificador

El codificador está compuesto por 6 capas idénticas. Cada una de estas capas, tiene 2 subcapas. La primera es un mecanismo de auto-atención multi-cabezal ("multi-head self-attention"). La segunda es una red neuronal prealimentada totalmente conectada aplicada a cada posición de manera separada pero idéntica ("position-wise fully connected feed-forward network"). Prealimentada quiere decir que las conexiones entre las neuronas no forman ciclos (bucles). Emplea una conexión residual alrededor de cada una de las subcapas, seguida de una normalización de capa [13].

3.2.5.3 Funcionamiento BERT

BERT representa una sola frase o una pareja de frases (pregunta, respuesta) como una secuencia de “tokens” de acuerdo con las siguientes características:

- Utiliza “WordPiece embeddings” [14].
- El primer “token” de la secuencia es “[CLS]”.
- Cuando hay dos frases, en la secuencia están separadas por el “token” “[SEP]”, y a cada “token” se le añade un “embedding” para indicar a qué frase pertenece.
- Para un “token” dado, su representación para la entrada se computa sumando los correspondientes “embeddings” del “token”, segmento y posición.

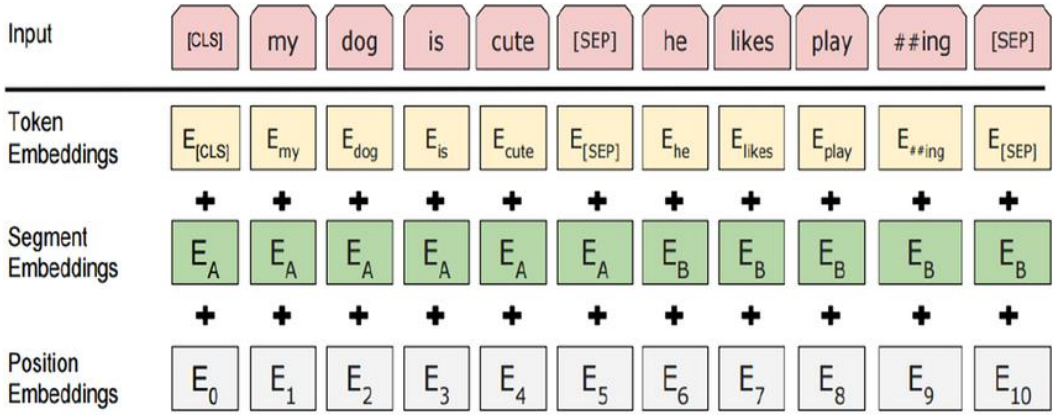


Figura 12: *Representación de entrada del modelo BERT*

3.2.5.3.1 Pre-entrenamiento

En este apartado se explicará en qué consiste el preentrenamiento siguiendo [8]. El preentrenamiento se divide en dos tareas:

- “Masked LM”: al estar hablando de un modelo bidireccional, y sin tener la intención de que cada palabra sea capaz de “verse a sí misma”, lo que se hace es ocultar (o enmascarar) un porcentaje aleatorio de “tokens” de la entrada para luego intentar predecirlos, normalmente, el 15 %. Para ello se sustituyen los “tokens” seleccionados: un 80 % de los casos por el “token” “[MASK]”, un 10 % por otro “token” aleatorio y un 10 % se deja el “token” que estaba.
- Predicción de la siguiente frase (NSP): disponiendo de dos frases A y B, para cada ejemplo de pre-entrenamiento, en un 50 % de los casos B es la frase que sigue a A y se etiqueta como IsNext, en el otro 50 % B es una frase aleatoria del corpus y se etiqueta como NotNext.

Para el pre-entrenamiento del modelo en inglés, se utilizan “BooksCorpus” (800M de palabras) y “Wikipedia” en inglés (2500M de palabras) como conjuntos de datos.

3.2.5.3.1 El reajuste

El reajuste o “fine-tuning” es relativamente sencillo gracias al mecanismo de autoatención del “Transformer”. Para cada tarea en específico, simplemente se ponen las entradas y salidas correspondientes y se ajustan los parámetros de principio a fin dependiendo del resultado que se quiera conseguir y en la mayoría de los casos basándose en la metodología de prueba y error.

El coste del reajuste, comparado con el de preentrenamiento, es muy bajo ya que, a niveles prácticos existen funciones parametrizables que permiten ajustar el modelo a nuestras necesidades en particular.

3.2.6 Métricas de evaluación

3.2.6.1 Matriz de confusión

La matriz de confusión está definida como una métrica de evaluación en la cual los resultados de la calidad del modelo se exponen en una matriz cuadrada de N elementos, donde N es el número de clases a predecir.

Para este caso, un modelo entrenado para la clasificación binaria, la matriz de confusión resultante únicamente consta de 2 filas y 2 columnas.

Esta tabla muestra la cantidad de observaciones clasificadas correctamente (verdaderos positivos y verdaderos negativos) y las que se han clasificado incorrectamente (falsos positivos y falsos negativos) por el modelo de clasificación.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 13: *Matriz de confusión*

Gracias a esta matriz de confusión se pueden extraer diversas métricas como se expondrá a continuación.

3.2.6.2 Métricas y curvas de evaluación

Teniendo en cuenta las siglas definidas en la figura 13 se explicarán cuáles son las métricas que se pueden obtener gracias a la matriz de confusión y cómo calcularlas.

- Precisión: proporción del número total de predicciones que eran correctas en relación con el total de predicciones:

$$\frac{VP+VN}{(VP+FP+VN+FN)}$$

- Sensibilidad (recall): proporción de casos positivos identificados correctamente:

$$\frac{VP}{(VP+FN)}$$

- Curva precision-recall: gráfica que muestra la relación entre la precisión y el recall.

- Especificidad: proporción de casos negativos identificados correctamente:

$$\frac{VN}{(VN+FP)}$$

- Valor F (F1-score): Medida entre precisión y sensibilidad:

$$\frac{2 \cdot (precisión \cdot sensibilidad)}{(precisión + sensibilidad)}$$

- Varianza: Sensibilidad del modelo a las fluctuaciones de los datos de entrenamiento.

- Sesgo: tendencia del modelo a cometer errores consistentes debido a suposiciones simplificadas o limitaciones en su capacidad de representar la complejidad de los datos

- Valor AUC: El área bajo la curva indica la capacidad de un modelo para distinguir entre clases positivas y negativas, es decir su rendimiento. Un valor de 1.0 representa un rendimiento perfecto y 0.5 indica una clasificación aleatoria.

- Curva ROC: Muestra la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos.

Para obtener tanto la matriz de confusión como algunas estas métricas, en el notebook de `"sklearn-model-performance.ipynb"` se utiliza las funciones de sklearn `"classification_report()"` y `"ConfusionMatrixDisplay()"`.

3.2.6.3 Validación cruzada

Para evaluar los modelos y algunos de los métodos de división del conjunto de entrenamiento y prueba etc… Se utiliza la validación cruzada en algunas funciones como `cross_val_score()` o `StratifiedKFold()` donde se busca evaluar el rendimiento del modelo en datos no vistos y obtener una estimación más robusta del rendimiento general del modelo en lugar de simplemente evaluarlo en el conjunto de entrenamiento.

Esto se consigue utilizando diferentes "folds" o pliegues donde cada pliegue es una división diferente de los datos de entrenamiento y se evalúa una vez por cada pliegue. Finalmente, se hace un promedio de los resultados obtenidos y se consigue una puntuación más realista al haber evaluado con más de una división de datos.

4 Implementación

En este apartado se repasarán los detalles de implementación del proyecto, desde el sistema operativo utilizado hasta los ficheros que han sido utilizados. La implementación del código fuente de este proyecto se ha escrito en el lenguaje de programación Python, al haber sido confeccionadas para este lenguaje múltiples bibliotecas de programación como abstracción de sistemas de inteligencia artificial NLP, con las cuales pueden ser desarrollados estos modelos, siendo delegadas a estas bibliotecas las implementaciones y optimizaciones de bajo nivel. Sin dejar de lado otras bibliotecas encargadas de la parte de data collection como han sido utilizadas en este caso las librerías de apify o selenium, por ejemplo.

4.1 Configuración del entorno de desarrollo

El sistema operativo sobre el que se ha implementado el desarrollo ha sido:

- Windows 11 Home
- Versión: 22H2
- Versión del sistema operativo: 22621.1555

En cuanto al entorno de desarrollo, se ha utilizado Anaconda con las siguientes versiones:

- anaconda: 23.3.1
- anaconda-client: 1.11.1
- anaconda-navigator: 2.4.0
- anaconda-project: 0.11.1

El lenguaje utilizado ha sido Python 3.10.11. La mayor parte del desarrollo se ha hecho sobre notebooks de Jupyter con las siguientes versiones:

- jupyter_client: 6.5.2
- jupyter_core: 5.2.0
- jupyter_server: 1.23.4

Algunas versiones de librerías clave utilizadas:

- pytorch: 11.8
- cuda_version: 11.8 (tenía 12.1, tuve que hacer downgrade para que hubiera compatibilidad entre versiones)

La totalidad del proyecto ha sido desarrollado en un entorno virtual con la versión de Python antes mencionada y los paquetes instalados listados en los respectivos requirements de los notebook implicados.

El orden de ejecución de los diferentes notebooks implementados durante el proyecto, si se quisiera seguir los mismos pasos realizados, sería el siguiente:

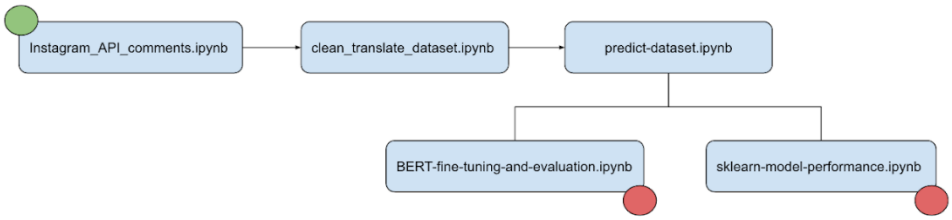


Figura 14: *Orden de ejecución de los notebooks implicados*

4.2 Recolección de datos

Para la parte de la recolección de datos, las bibliotecas principales utilizadas han sido las mencionadas en el apartado 3 de data collection, `apify_client`, `selenium` y `beautifulSoup` contando con `pandas`, `pickle` y otras librerías auxiliares que han sido de ayuda durante el proceso de implementación de esta parte inicial del proyecto para manejar y guardar tablas de datos. La idea en esta sección es recolectar comentarios de publicaciones de jugadores de fútbol en instagram, esto ha sido realizado en el notebook `"Instagram_API_comments.ipynb"`.

Primeramente, como se ha explicado en anteriores secciones, se ha decidido utilizar la herramienta web Apify que cuenta con una librería para Python que permite realizar llamadas a la API de Instagram. Hay diferentes endpoints de Instagram a los cuales acceder, pero tal y como está planteada la extracción de datos, la idea es llamar al endpoint que proporciona información general sobre las 30 primeras publicaciones de cada jugador.

El objetivo de esto no es conseguir toda la información, es poder almacenar el código de publicación o "shortCode" y posteriormente iterar por cada uno de ellos.

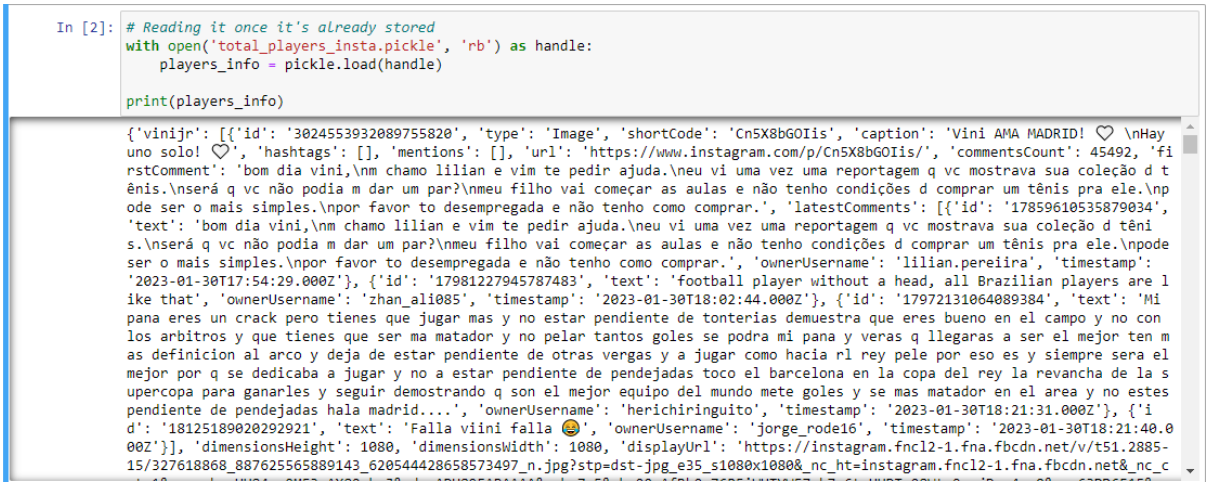


Figura 15: *Información general de los jugadores extraída con* `apify_client`

A continuación, se hace uso de la librería `selenium` para acceder a la web de instagram. Cabe recalcar que, para que funcione el módulo que se encarga de abrir una instancia del navegador, se requiere un ejecutable que hay que descargar de la página oficial del módulo del navegador sobre el cual estamos trabajando, en mi caso se ha trabajado con Chrome y se ha descargado la versión del driver en la página de ChromeDriver.

Una vez realizado el paso anterior y habiendo iniciado sesión en Instagram, se puede pasar al último paso que es la extracción de los comentarios.

Como había comentado al principio, se dispone de una lista con códigos de publicaciones y se itera por cada uno de ellos haciendo uso de selenium para "scrapear" los comentarios deseados. A medida que se extraen comentarios se almacenan en un dataframe con el nombre del jugador y el comentario hacia él.

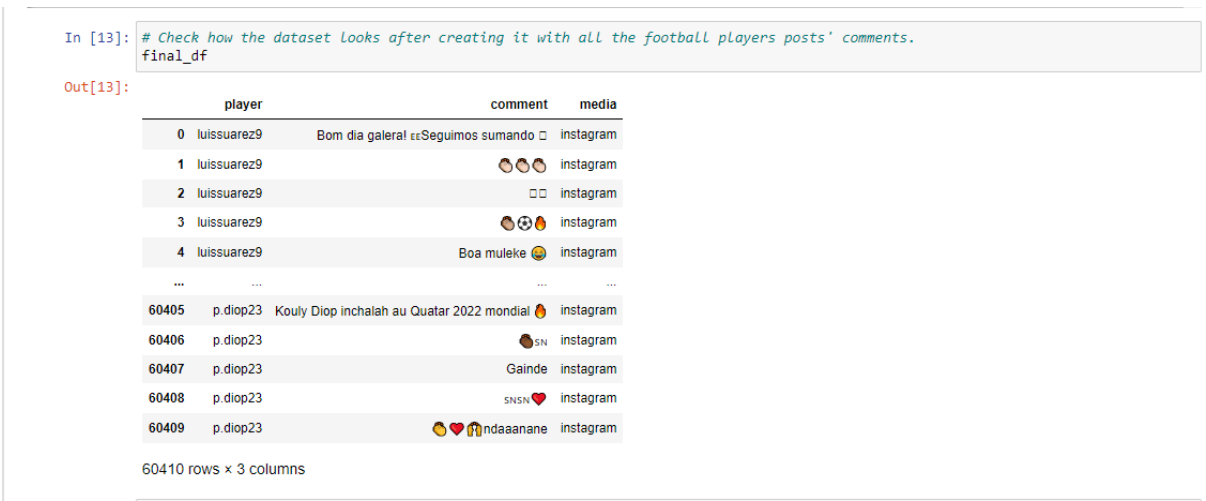


Figura 16: *Dataset inicial en forma de dataframe*

4.3 Data cleaning

Enlazando con el apartado anterior, en el mismo fichero `"Instagram_API_comments.ipynb"` se hace una pequeña limpieza del dataset inicial eliminando emojis y símbolos pero la mayor parte de esta parte se realiza en el fichero `"clean_translate_dataset.ipynb"` aunque, al ir creando varios datasets a medida que se va avanzando como si fueran "checkpoints", esta función de limpieza de datos también está presente en otros ficheros y momentos de la implementación.

Después de aplicar esta metodología se dispone de un dataset de 30.000 comentarios habiendo reducido considerablemente su tamaño y aumentando su calidad gracias a las técnicas utilizadas.

4.4 Preprocesado de datos

4.4.1 Creación del dataset

En este apartado trata sobre diferentes tipos de preprocesado, primero se hablará sobre el preprocesado a la hora de traducir los comentarios del dataset. A continuación se adjunta una imagen del código que se encuentra en `"clean_translate_dataset.ipynb"` donde podemos ver, gracias a `"tqdm"`, que el proceso de traducir el dataset completo (30.000 comentarios aproximadamente) duró alrededor de 4 horas.

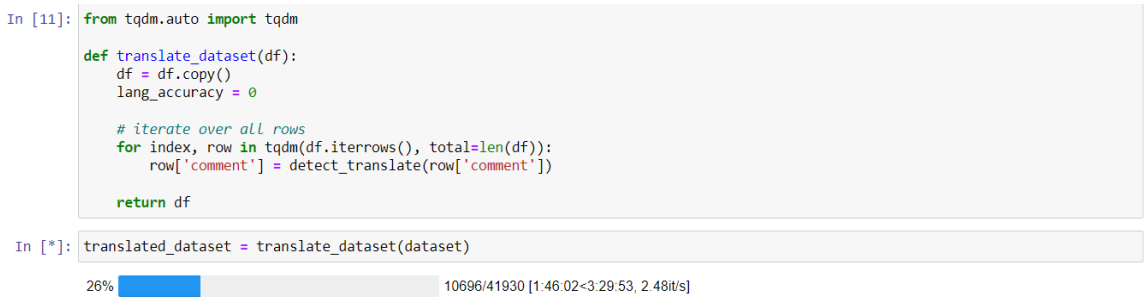


Figura 17: *Código fuente del proceso de traducción del dataset*

Después de traducirlo, se reordenan los datos de forma aleatoria con tal de no crear patrones a la hora de utilizar el modelo, se eliminan las filas del dataset que no se han logrado traducir correctamente y llegados a este punto, se utiliza la herramienta externa Google Sheets para etiquetar manualmente cada comentario con un 1 o un 0 en la columna de "sentiment". De esta forma, aunque el trabajo de etiquetar cada

comentario sea laborioso, se estaría creando un dataset íntegro, guardado con el nombre `"final_dataset_translated_shuffle.csv"`.

Llegados a este punto, se utiliza un modelo pre-entrenado con un conjunto de datos mayor, llamado `"training.1600000.processed.noemoticon.csv"`, para predecir el resto de las muestras del dataset, ya que se piensa en que en un futuro, el trabajo pueda continuar o pueda ser reproducido por otra persona que quiera mejorarlo. Esto se realiza en el notebook `"predict-dataset.ipynb"`.

Luego, se utilizan de nuevo algunas técnicas de data cleaning y se crea un nuevo dataset que contiene solamente comentarios etiquetados manualmente bajo el nombre de `"cleaned_dataset.csv"`. Finalmente, se amplía el número de filas en 200 filas más, como se ha hecho anteriormente con Google Sheets pero esta vez en lugar de etiquetar, el trabajo consiste en revisar las etiquetas que estaban predichas durante el proceso mencionado en el párrafo inmediatamente superior a este. Esto se hace porque, aunque estén predichas, si me fio de los resultados del modelo entrenado con otros datos, los nuevos datos pueden estar bajo el mismo sesgo o errores. El dataset definitivo, utilizado en durante la implementación ha sido `"definitive_dataset.csv"`.

A modo de resumen y para comprender mejor el proceso de transformación que ha ido experimentando, se adjunta a continuación un esquema representando el orden de transformación de forma general, ya que internamente, el conjunto de datos ha sido sometido a varias transformaciones y procesamientos adicionales para mejorar su calidad y adecuación al modelo, aunque no se reflejen explícitamente en este informe:

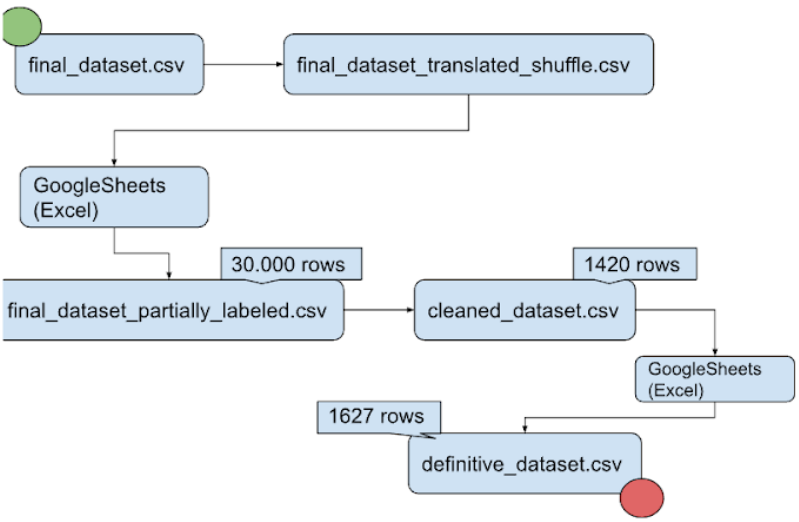


Figura 18: *Orden de modificación del dataset utilizado*

4.4.2 Preprocesado de datos para el modelo

También se trata el preprocesado, esta vez en el notebook `"sklearn-model-performance-ipynb"` pero manipulando datos del dataset sin que sea éste modificado. Esta vez se tratará el texto para poder ser utilizado de forma óptima por los modelos a evaluar, utilizando algoritmos y técnicas relacionadas con TF-IDF entre otras.

En el notebook mencionado, la librería utilizada principalmente ha sido `"nltk"`, una librería especializada en procesamiento de texto. Se han utilizado varias funciones de esta librería como stopwords en inglés, la librería "`re"` para pasar el texto a minúscula y eliminar símbolos con el uso de expresiones regulares, otras necesarias para probar la lematización e incluso "`WordNet"` para visualizar palabras ofensivas y no ofensivas.

4.4.3 TfidfVectorizer() vs CountVectorizer()

Después de este primer preprocesado, se pasa a utilizar el vectorizador cuya función se explicará a continuación. Tanto `TfidfVectorizer()` como `CountVectorizer()`, convierten una colección de documentos (corpus) (en este caso se tomará como documento a cada frase del dataset) a una matriz de características. Las características que componen el vectorizador, en el contexto de `TfidfVectorizer()` y `CountVectorizer()` son las unidades de texto utilizadas para construir la representación numérica del corpus.

El objetivo del vectorizador es asignar un valor numérico en el caso de `TfidfVectorizer()`, los pesos (TF-IDF) y en el caso de `CountVectorizer()` el número de ocurrencias, a cada característica para cuantificar su importancia relativa en cada documento (frase) del corpus. Esto permite representar cada documento (frase) como un vector numérico, donde cada componente del vector corresponde a una característica y su valor indica la importancia relativa de esa característica en el corpus. Esta matriz representará la importancia de cada palabra en el corpus y se utilizará como entrada para los modelos de aprendizaje automático.

Cada uno de los vectorizadores tiene una parte de preprocesamiento donde se eliminan algunos símbolos y caracteres y luego un parte de tokenización donde se separa cada una de las palabras de distintas formas a elegir.

En este caso se aplica el preprocesamiento del corpus previamente y para la parte de tokenización, por defecto, se divide el texto en unigramas pero teniendo en cuenta que este trabajo trata de análisis de sentimientos, los bigramas capturan mejor el sentimiento ya que ciertas combinaciones de palabras pueden transmitir emociones de manera más efectiva. Se inicializan los vectorizadores con el parámetro `ngram_range(1,2)` para considerar tanto unigramas como bigramas para la representación vectorial del texto. También se inicializa `TfidfVectorizer()` con el

parámetro `use_idf=True` con tal de utilizar IDF y tener en cuenta la relevancia respecto al corpus.

Hay que tener en cuenta que para datasets pequeños con cadenas de texto no muy largas, `CountVectorizer()` puede funcionar mejor ya que solo mirando las veces que una palabra ha aparecido puede ser suficiente, así que decidí analizar los resultados de los vectorizadores después de vectorizar el corpus y posteriormente comparar estas dos maneras de vectorizar con mi dataset y con 5 modelos distintos para resolver esta disyuntiva. Antes de realizar el experimento se comparará su funcionamiento y se comentarán algunos aspectos a tener en cuenta:

`TfidfVectorizer()`, tiene ciertos aspectos a tener en cuenta a la hora de decidir si escoger este vectorizador u otro:

- Al limitar el número de características, se seleccionan las palabras clave más importantes y distintivas. Esto puede ayudar a que los modelos se centren en la información más relevante y discriminativa para la clasificación.
- El modelo tiene menos parámetros a aprender y se vuelve menos propenso a memorizar patrones irrelevantes o ruidosos.

Por otro lado, al utilizar `CountVectorizer()` se deben tener en cuenta otros factores:

- `CountVectorizer()` no descarta las palabras frecuentes y las palabras tendrán mejor puntuación cuantas más veces salgan en el corpus. Además, las palabras más frecuentes pueden contener información valiosa para los modelos de clasificación y `CountVectorizer()` permite capturar esta información y utilizarla en el proceso de clasificación.
- Puede manejar datos de texto que no están estructurados ni presentan un patrón, donde la longitud de los comentarios y la presencia de palabras distintas pueden presentar una dificultad. Al tener en cuenta la frecuencia de cada palabra, proporciona una representación estructurada de los datos teniendo en cuenta simplemente el número de ocurrencias, que puede ser utilizada por los algoritmos de clasificación para extraer patrones y realizar predicciones.

Hay que mencionar que los experimentos se han realizado tokenizando el texto por bigramas.

En el Anexo 1, se visualizan los resultados después de hacer la predicción del conjunto de entrenamiento. A simple vista, tanto con con `TfidfVectorizer()` como con `CountVectorizer()` se observa que sufren overfitting ya que muestran una precisión cercana a 100%. Los únicos casos en los que la precisión ha sido, aunque no muy confiable, aceptable, ha sido con `TfidfVectorizer()` y con los modelos `LogisticRegression()` y `MultinomialNB()`. `LogisticRegression()` ha necesitado un número inferior de características para dar está puntuación, en cambio `MultinomialNB()` ha utilizado el 100% de las características.

Como conclusión, para que los distintos modelos funcionen de una manera eficiente, a priori, se ha visto que puede haber que variar el número de características a utilizar por ambos vectorizadores dependiendo del modelo.

A primera vista, `TfidfVectorizer()` funciona de una manera más eficiente sin el ajuste de hiper parámetros ya que considero que la forma de asignar importancia a las palabras es más adecuada para el problema encontrado donde la relevancia de una palabra no viene definida por su número de ocurrencias y una palabra poco común puede aportar mucha información. De todas formas, salta a la vista que hay un problema de sobreajuste y en la parte de resultados se verá si se puede solucionar mediante el ajuste de hiper parámetros.

También mencionar que los parámetros modificados están especificados en el notebook `"sklearn-model-performance.ipynb"` y han sido mencionados a lo largo de esta explicación.

4.4.4 Train-test split

A la hora de escoger qué tipo de métodos de partición de clases, es decir, las funciones que dividen el dataset en un conjunto de datos de entrenamiento, validación y de prueba, se han tenido en cuenta diversos factores. Se comienza mirando los 15 métodos de "splitter classes" del módulo de `model_selection` de sklearn y analizando cuál de ellos se ajustaría mejor a el caso en concreto.

De estos 15, descarto 5 de ellos porque trataban con grupos y en el conjunto de datos del que se dispone no existe una correlación entre comentarios, aunque vayan dirigidos al mismo jugador ya que hay una gran variedad de opiniones independientemente del jugador o la imagen sobre la que se comente, por ese motivo no se podrían definir grupos para realizar la división de train y test. Luego otro se descarta porque está pensado para problemas temporales ("time-series") y no es el caso.

De los 9 restantes hay algunos que no tienen en cuenta el desbalance de clases y se enfocan en mantener la proporción de éstas, otros que se enfocan en particionar específicamente los datos y esto es contrario a lo que se busca para el conjunto de datos sobre el que se está trabajando. Finalmente, quedan 4 métodos: `StratifiedShuffleSplit`, `StratifiedKFold` y `LeaveOneOut`.

Llegado a este punto, me planteo cuál de ellos se ajustaría mejor al problema en cuestión:

`LeaveOneOut` se divide el conjunto de datos con todas las posibles combinaciones de entrenamiento y prueba realizando el proceso de validación dejando cada vez una instancia del conjunto de datos fuera ("`OneOut`") y utilizando el resto como entrenamiento. Este tipo de divisor de clases es útil para evaluar la capacidad del modelo para generalizar y predecir correctamente en datos limitados. Tiene un coste computacional alto, pero resulta útil en un dataset pequeño (menos de 100 muestras) ya que se aprovecha al máximo la información para entrenar y evaluar el modelo.

Luego, entre `StratifiedKFold`, `StratifiedShuffleSplit` me decanto más por `StratifiedKFold` ya que mantiene la proporción entre las clases como lo hace `StratifiedShuffleSplit` pero también se basa en un enfoque de validación cruzada que divide el conjunto de datos en k pliegues (folds) de manera estratificada, lo que significa que se mantiene la proporción de las clases en cada fold incluso tiene un parámetro llamado "shuffle" para mezclar ejemplos, cosa que aumentaría la representatividad y la generalización. Además, tiene en cuenta el desequilibrio de clases, que, aunque no sea mucho, mejoraría el rendimiento a la hora de evaluar.

Quiero aclarar que para `StratifiedKFold` se busca evaluar el rendimiento del modelo en datos no vistos (test) y obtener una estimación más robusta de éste en lugar de simplemente evaluarlo en el conjunto de entrenamiento, esto se consigue utilizando la validación cruzada.

Teniendo en cuenta que la mayoría de los modelos estadísticos mejoran si el conjunto de datos de entrenamiento es mayor, la validación cruzada de `StratifiedKFold` estima el rendimiento de un modelo entrenado en un conjunto de datos 100×(k-1) k% de los datos disponibles, en lugar de en el 100% de ellos. Por tanto, si se realiza una validación cruzada para estimar el rendimiento y, a continuación, se utiliza un modelo entrenado con todos los datos, su rendimiento será ligeramente superior al que sugiere la estimación de la validación cruzada. Existen algunos artículos sobre este tema, los cuáles han sido utilizados como referencia para fundamentar el criterio de elección de un método u otro.

4.5 Model selection

Antes de comenzar la selección del modelo que mejor se ajuste al problema en concreto, veo necesario hacer una pequeña investigación de los 5 distintos modelos a evaluar en este trabajo.

4.5.1 LinearSVC

Primeramente, un modelo lineal es un tipo de modelo que se basa en la idea de una relación lineal entre las variables de entrada y la variable de salida. Supone que los datos pueden ser representados mediante una combinación lineal de características ponderadas (con su peso TF-IDF o simplemente la frecuencia de aparición). En un modelo lineal, se asume que la relación entre las variables de entrada y la variable de salida puede ser representada mediante una ecuación lineal o ecuación de la recta y esta ecuación lineal toma la forma de:

$$y = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

Donde:

- y = variable de salida o el valor que se desea predecir (0 o 1).
- x_1, x_2, ..., x_n = variables de entrada o vector de características.
- w_1, w_2, ..., w_n = pesos asociados por el modelo a cada característica o vector de pesos.
- b = es el sesgo o “bias” (cuánto se alejan las predicciones de un modelo respecto a los valores reales).

El modelo `linearSVC()` implementa una variante del algoritmo SVM(Support Vector Machine) llamada SVM lineal. El objetivo del modelo lineal es, mediante el algoritmo mencionado, aprender los valores óptimos de los pesos (w_1, w_2, ..., w_n) y el sesgo (b) a partir de los datos de entrenamiento, de manera que la ecuación lineal pueda hacer predicciones precisas para nuevos datos.

Explicando más en detalle el algoritmo, en un problema de clasificación binaria se dispone de un hiperplano (o recta) de separación. Esta recta se consigue durante el proceso de entrenamiento encontrando los pesos adecuados, extraídos de las características, y un sesgo adecuado para minimizar la función de pérdida que en este caso es la función de descenso del gradiente (SGD). Utilizando la fórmula declarada anteriormente, se consigue crear una recta discriminativa. El objetivo de esta recta es maximizar el margen de separación entre las clases, es decir, que esté a la mayor distancia posible, perpendicularmente, de un ejemplo de cada clase.

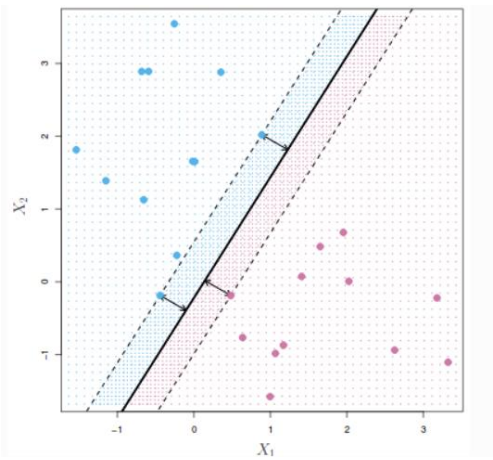


Figura 19: *Plano de representación de características donde se aprecia el margen de separación óptimo*

Para lograr maximizar el margen de separación se cuenta con vectores de soporte, de ahí el nombre del algoritmo, que son los subconjuntos de muestras que están más cerca de esta recta, siendo muy importantes a la hora de definir su ubicación y orientación [15].

A la vez, también se buscan para clasificar correctamente todas las muestras de entrenamiento con el objetivo de maximizar el margen de separación de las dos clases en cuestión. Las características extraídas son representadas en un espacio según un peso asociado por el modelo.

La idea de este algoritmo es, para cada muestra con su respectivo peso y sesgo se calcula la función de decisión, es decir, la función de la recta. El signo del valor obtenido de la función de decisión determina la clasificación de la muestra. Si el resultado de esta función es positivo pertenece a una clase y si es negativo, a la otra clase. También se tiene en cuenta el valor absoluto del resultado, ya que si es mayor querrá decir que el modelo confía en que ese resultado pertenece a una clase en concreto. Esto es porque si el valor es grande respecto la recta, visualmente se puede decir que está alejado de la recta y está lejos del límite discriminativo entre una clase y otra.

Para la implementación del modelo se ha decidido utilizar `TfIdfVectorizer()` ya que como se puede apreciar en el Anexo 1, el tipo de vectorizador no afecta significativamente al modelo, ya que la manera de asignar pesos a cada característica y su capacidad para extraer características relevantes es buena independientemente del tipo de vectorizador. El aspecto que sí que afecta significativamente al rendimiento es el número de características utilizadas por cada vectorizador, pero después del estudio visto en el Anexo 1, se cuenta con el número óptimo.

4.5.2 MultinomialNB

Los modelos Naive Bayes asumen independencia entre las características dada la fórmula basada en la probabilidad condicional y pueden funcionar bien un corpus donde esta teoría es razonable, como por ejemplo en un problema donde dada una palabra “B” pueda seguirle una palabra “A”. Se asume que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica, por eso el nombre de “Naive”, inocentes.

Entre los diferentes modelos Naive Bayes se encuentra `MultinomialNB()`, este modelo se basa en la distribución multinomial que consiste en representar datos con múltiples clases posibles, en este caso binomial porque hay solamente dos clases posibles. Esto se hace con el uso de etiquetas como pueden ser “ofensivo” o “no ofensivo”.

Al implementar una fórmula basada en la probabilidad condicional, habiendo aprendido con anterioridad de las frecuencias de las palabras de un conjunto de entrenamiento, haciendo uso de la fórmula mencionada podrá afirmar que una palabra “A” pertenece a una clase “X” habiendo un evento anterior como puede ser haber visto que una palabra “B” con características similares a “A” pertenecía a una clase “X”.

La fórmula implementada por este tipo de modelos es:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (6)$$

Donde:

- P(A|B) = probabilidad de que ocurra A sabiendo que B ya ha ocurrido.
- P(B|A) = probabilidad de que ocurra B sabiendo que se ha dado A.
- P(A) = probabilidad de que ocurra A.
- P(B) = probabilidad de que ocurra B.

Aunque normalmente, este tipo de modelos prefieren el recuento con números enteros, éste, también funciona aunque no de forma óptima, con recuentos fraccionados, es decir, con decimales como ocurre al utilizar tf-idf lo que nos sugiere que puede funcionar mejor con `CountVectorizer()` que con `TfidfVectorizer()`.

Algunas ventajas del modelo son:

- En casos donde se asume independencia entre los datos, el algoritmo se comporta mejor que otros modelos de clasificación, incluso con menos datos de entrenamiento.
- El desacoplamiento de las distribuciones de características condicionales de clase significa que cada distribución puede ser estimada independientemente como si tuviera una sola dimensión. Esto ayuda con problemas derivados de la dimensionalidad y mejora el rendimiento.

Algunas desventajas del modelo son:

- Al asumir independencia es probable que no refleje cómo son los datos reales.
- Cuando el conjunto de prueba tiene una característica que no ha sido observada en el conjunto de entrenamiento, el modelo le asignará una probabilidad de cero y será inútil realizar predicciones. Uno de los principales métodos para evitar esto, es aplicar técnicas de suavizado, siendo la estimación de Laplace una de las más populares.

En conclusión, a niveles prácticos el modelo aprende de las frecuencias o importancia de las palabras obtenidas durante el entrenamiento, esto lo consigue recibiendo los datos de un vectorizador. Una vez tiene una muestra de datos, utilizando la distribución multinomial y el algoritmo de Bayes asigna la probabilidad de pertenecer a una clase a cada palabra vista durante el entrenamiento.

Por ejemplo, si la palabra “bueno” aparece 5 veces durante el entrenamiento y las 5 veces esta palabra estaba etiquetada como positiva, la probabilidad calculada es de 100% por eso cuando el modelo intente predecir la palabra bueno, teniendo la probabilidad de 100% asignada a esa palabra, la clasificará como positiva. Esto ocurrirá con todas las palabras vistas durante el entrenamiento con sus respectivas

probabilidades de pertenecer a una clase, teniendo en cuenta su frecuencia y la clase a la que pertenecía (1 o 0, positiva o negativa) pero al asumir independencia entre ellas no se tiene en cuenta parecidos ni nada por el estilo.

Esto hace pensar que puede pasar si una palabra no ha sido vista nunca. Según la implementación por defecto del modelo, al no haber visto nunca esa palabra y no tener ninguna probabilidad asignada, le asigna una probabilidad de 0 o muy cercana a 0.

Sin embargo, algunas implementaciones de `MultinomialNB()` pueden utilizar técnicas de suavizado para asignar una probabilidad baja a todas las palabras, incluso a las que no se han visto, con tal de evitar este problema de probabilidad 0 y así poder tratar con palabras no vistas durante el entrenamiento. Si se intentara predecir una palabra nunca vista, el modelo le asignaría una probabilidad muy baja de pertenecer a ambas clases dependiendo de la técnica de suavizado utilizada y basándose en probabilidad se le acabaría asignando una clase.

4.5.3 Logistic Regression

La regresión logística es uno de los algoritmos de ML más simples y utilizados para la clasificación binaria de clases, basado en el método estadístico. Aunque en el nombre del modelo se incluye la palabra regresión, principalmente se utiliza en problemas de clasificación.

Este modelo dispone de unas variables independientes como pueden ser las características extraídas de nuestro corpus y su función es, mediante el uso de una función matemática, modelar las relaciones que puedan haber entre las diferentes características para predecir la variable dependiente, es decir, la clase 0 o 1. En este punto entra en juego la función matemática sigmoidea, es la función de activación de la capa de salida del modelo que tiene forma de “s” y su procedimiento es tomar un valor de entrada lineal y transformarlo en un valor entre 0 y 1 como una probabilidad [16].

$$P(t) = \frac{1}{1+e^{-t}} \qquad (7)$$

Donde:

- t = valor numérico al cual se le aplica la función sigmoidea.
- e = base del logaritmo natural

En el contexto de la regresión logística, se transformará una combinación lineal de características en una probabilidad, para eso la función sigmoidea ayuda a ajustar los valores de las características en una escala de probabilidades donde valores

cercanos a 0 representan ser cercanos a una clase y valores cercanos a 1 a la otra clase.

El funcionamiento general del modelo sería, tal y como se hace en otros modelos, durante la fase de entrenamiento ajustan los pesos de las características mirando la combinación que reduzca la función de pérdida al valor mínimo óptimo. Una vez ajustados los pesos de cada característica, se utiliza la función sigmoidea para tomar ese valor numérico y transformarlo a un valor entre 0 y 1 como se ha mencionado anteriormente y esta probabilidad representa la confianza del modelo en que la muestra pertenece a una clase u otra. Si la salida de la función es mayor que 0,5 se considera que la probabilidad de pertenecer la clase 1 es alta, si es menor, de pertenecer a la clase 0.

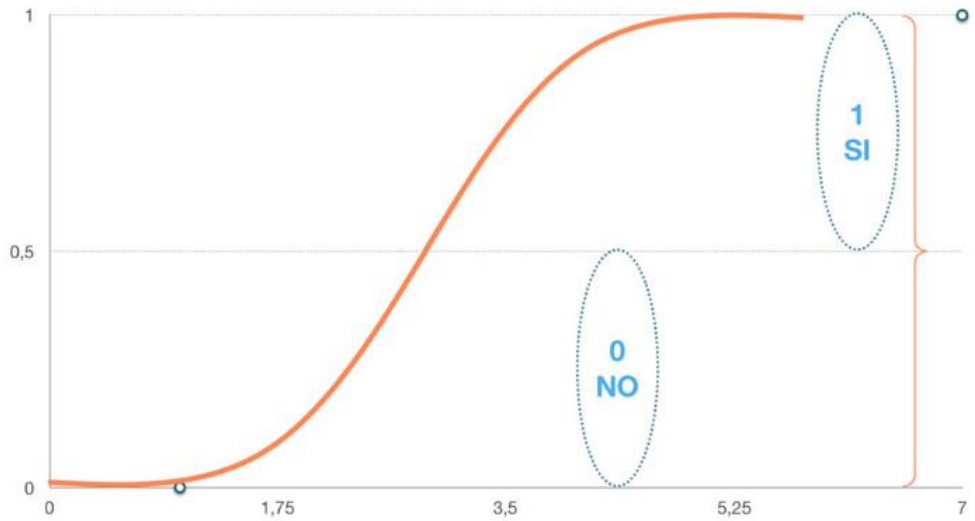


Figura 20: *Esquema de funcionamiento de función sigmoidea*

4.5.4 Decision Tree Classifier

Los árboles de decisión son una técnica de aprendizaje automático supervisado muy utilizada. Como su nombre indica, esta técnica de ML toma una serie de decisiones en forma de árbol, divide el conjunto de datos en subconjuntos más pequeños teniendo en cuenta sus características formando un árbol de decisiones. Se comienza con un nodo raíz con unas determinadas características y se intenta predecir a qué clase pertenece ese nodo. Los nodos intermedios (las ramas) representan decisiones o reglas y los nodos finales (las hojas) representan una clase o resultado final.

El objetivo del modelo es crear un árbol de decisión que clasifique las muestras de la forma más eficiente posible, por eso durante la fase de entrenamiento, selecciona las características más informativas y discriminativas, también llamadas predictores, para dividir el conjunto de datos en subconjuntos más puros. Cada nodo del árbol

representa una característica y en cada nodo se evalúa la calidad de su división, utilizando criterios que maximicen la pureza de ésta. Luego, se crean dos nuevos nodos hijos correspondientes a las divisiones resultantes, el algoritmo del árbol de decisiones evalúa si la división resultante mejora la calidad de la partición y el proceso se repite recursivamente hasta que se alcanza algún criterio de parada.

Para realizar esta división se tienen en cuenta diferentes criterios como la reducción de la impureza o la ganancia de información de un nodo. La impureza se refiere a como de mezcladas están las clases a predecir en un nodo, es decir, si la medida de la impureza o "gini" es 0 quiere decir que el nodo es totalmente puro y se afirma que pertenece a una clase.

$$gini = 1 - \Sigma (p_i^2) \qquad (8)$$

Donde:

- Σ = suma sobre todas las clases.
- p_i = muestras que pertenecen a la clase i.

Por otro lado, la ganancia de información se basa en la entropía, que mide el grado de incertidumbre o desorden en el conjunto de datos. En cada paso de la construcción del árbol, se calcula la entropía antes y después de la división en función de la característica considerada. La ganancia de información se define como la diferencia entre la entropía inicial y la suma ponderada de las entropías de los subconjuntos obtenidos.

$$entropy = - \Sigma (p_i * \log2(p_i)) \qquad (9)$$

Donde:

- Σ = suma sobre todas las clases.
- p_i = muestras que pertenecen a la clase i.

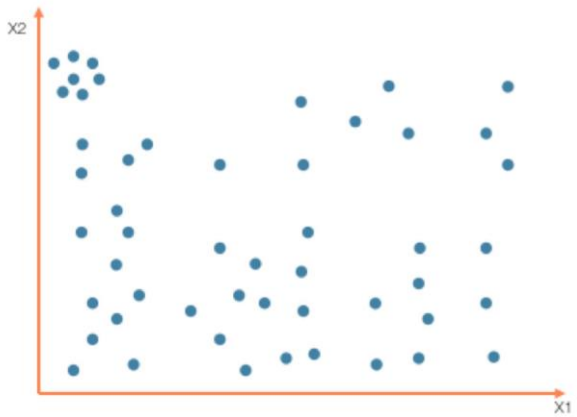


Figura 21: *Espacio de características previo a la división en subconjuntos*

Figura 22: *Espacio de características dividido en subconjuntos en un problema de clasificación multiclase*

Una vez creado el árbol de decisión, se utiliza para hacer predicciones sobre nuevas muestras. Para ello, se sigue el camino desde la raíz del árbol hasta llegar a una hoja, pasando por los nodos rama tomando decisiones basadas en las características de la muestra. La clase asignada a la muestra es la clase mayoritaria de las muestras que llegan a esa hoja.

En un problema de análisis de sentimientos como el que nos encontramos en este caso, se dispone de un conjunto de características de cada frase como pueden ser la longitud de la frase, el tono, el número de veces que ha aparecido una palabra, etc. El árbol de decisión, en este caso tiene nodos que evalúan estas características y toma decisiones acordes a esto. Un ejemplo sería tener un nodo que evaluase si en una frase aparece una palabra ofensiva y dependiendo de la decisión se sigue un camino u otro del árbol. Esto se hace sucesivamente evaluando cada decisión determinada durante el entrenamiento del modelo hasta que se llega a un nodo hoja donde después de haber evaluado todas condiciones de esa frase se llega a una conclusión y se etiqueta la frase como ofensiva (0) o no ofensiva (1).

4.5.4 Random Forest Classifier

Una vez explicados los árboles de decisión se puede entender con más facilidad el modelo Random Forest. Está formado por un conjunto o “ensemble” de árboles de decisión individuales. Mediante un parámetro llamado “`bootstrap`” se puede entrenar cada uno de estos árboles con una muestra ligeramente distinta de los datos de entrenamiento. La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo, cuyo funcionamiento ha sido explicado anteriormente.

Antes de comenzar a explicar un aspecto clave del modelo en cuestión, veo necesario explicar brevemente el problema de los modelos con el equilibrio entre sesgo y

varianza. El sesgo o “bias” hace referencia a cuánto se alejan las predicciones de un modelo respecto a los valores reales y la varianza se refiere a cuánto cambia el modelo dependiendo de los datos utilizados en su entrenamiento.

Cuanta más complejidad tenga un modelo mejor podrá actuar frente al sesgo mejorando las predicciones, pero si se acaba ajustando demasiado bien a los datos se acaba produciendo un sobreajuste. Los modelos con pocas ramas tienden a tener un sesgo alto porque no pueden representar bien las relaciones entre las variables, pero una varianza baja. Por el contrario, los modelos con muchas ramas tienden a ajustarse demasiado a los datos y a tener sobreajuste, pero teniendo un sesgo bajo.

Para solucionar el problema de desequilibrio aparecen los métodos “ensemble”. Estos métodos combinan diferentes modelos en uno nuevo con el objetivo de conseguir un equilibrio entre sesgo y varianza.

Uno de los tipos de “ensemble” más comunes y el que es utilizado por `RandomForestClassifier()` es el método de “bagging” (Bootstrap Aggregating). Este método consiste en ajustar múltiples modelos de árboles de decisión, cada uno con un subconjunto aleatorio distinto de los datos de entrenamiento. A la hora de predecir, todos los modelos que forman el “ensemble” participan aportando su predicción. Finalmente, el valor final es la media de la clase más frecuente, en este caso.

Obteniendo el promedio, se está reduciendo la varianza siempre y cuando los modelos no estén correlacionados. Para evitar esta correlación aún más, durante la fase entrenamiento de cada árbol de decisión, `RandomForestClassifier()` utiliza un número aleatorio de “m” predictores antes de evaluar cada división. He de recordar que los predictores son las características discriminativas o más informativas utilizadas por el modelo para hacer subdivisiones en los datos y así obtener resultados más puros. Para encontrar el número óptimo de predictores se puede utilizar la validación cruzada.

En conclusión, al utilizar un número aleatorio de predictores se está reduciendo la correlación entre cada uno de los diferentes árboles y por lo tanto obteniendo resultados diferentes no influenciados por una división de datos muy discriminativa. De esta forma el modelo final reduce la varianza y también el sobreajuste. Este sobreajuste también se reduce, ya que al utilizar árboles de decisión entrenados con diferentes subconjuntos de datos y un número aleatorio de características se consigue una mayor diversidad y el modelo final es más robusto y generalizado. Incluso llega un punto en el que agregar más árboles no mejora significativamente el rendimiento del modelo en el conjunto de test ya que el error en el test set se estabiliza [17].

5. Resultados y discusión

En esta sección se presentan los resultados de la investigación que se ha llevado a cabo utilizando los materiales y metodologías mencionadas anteriormente. Los resultados se expondrán de manera objetiva, intentando hacerlo sin ningún sesgo o interpretación, con el objetivo de realizar un análisis reproducible y escalable en un futuro.

5.1 Primeros resultados

Para este apartado, después del análisis realizado en el Anexo 1 y el Anexo 2, para cada modelo se utilizará un tipo de vectorizador u otro y con un número de características concreto dependiendo de cómo de bien le afecten al rendimiento de cada modelo, ya sea por un tema de complejidad, selección de características, asignación de pesos, etc.

El notebook que contiene estos análisis y los siguientes resultados es `"sklearn-model-performance.ipynb"` en su totalidad.

Para entender algunas de las métricas utilizadas para la evaluación de los diferentes modelos se puede consultar el apartado de Métricas y curvas de evaluación.

Primeramente, se ha decidido analizar los resultados obtenidos al evaluar la precisión de cada modelo con sus respectivos parámetros por defecto utilizando una matriz de confusión y adicionalmente un gráfico que muestra la curva de Precision-Recall. Estos resultados se pueden encontrar en el Anexo 3 de este trabajo.

A continuación, se adjunta un gráfico de ejemplo representativo de todos los modelos estudiados:

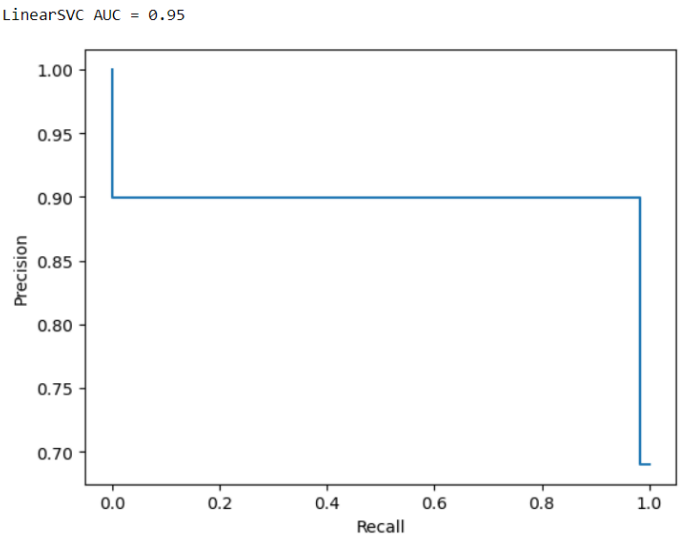


Figura 23: *Curva de precision-recall del modelo linearSVC*

Después de visualizar el gráfico, se llega a la conclusión de que, en general, los modelos comienzan en un valor en el eje de precisión de 1.0 y descienden rápidamente hasta un valor de 0.9 aproximadamente en este mismo eje. Podría ser debido a que, al inicio, los modelos cuentan con un umbral de clasificación estricto, es decir que el modelo solamente asigna una clase como positiva si está muy seguro de ello.

Esto significa que el modelo está obteniendo una alta precisión al principio, pero también un "recall" (sensibilidad) bajo, ya que muchas muestras positivas que podrían estar por debajo del umbral no serán clasificadas correctamente y solo está clasificando muestras como negativas. Luego, el umbral de decisión se relaja y más muestras comienzan a ser etiquetadas como positivas, pero sin aumentar la precisión. Esta justificación se puede corroborar con los resultados del notebook `"sklearn-model-performance.ipynb"` la sección `"(parenthesis) Checking if threshold affects auc accuracy"`.

Con el método `predict_proba()` o `decision_function()`, algunos modelos pueden ajustar el umbral de clasificación y utilizar probabilidades o valores de puntuación en la predicción en lugar de valores como 0 o 1 para controlar la probabilidad o puntuación necesaria en una muestra para clasificarla como positiva. Este punto será considerado durante el proceso de "fine-tuning" de los modelos.

Con tal de complementar la información obtenida con el uso de la gráfica de la curva de precision-recall también se utiliza la curva ROC que muestra la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos, en lugar de mostrar la tasa de verdaderos positivos. Los resultados obtenidos mediante esta métrica no son del estilo de la gráfica precision-recall por el hecho de que no son extremadamente parecidos entre ellos, por este motivo estarán adjuntados por separado en el Anexo 4.

Después de visualizar las diferentes gráficas se puede observar que en general todos los modelos presentan una relación entre verdaderos positivos y falsos como en un principio se ha mencionado en otros apartados de este trabajo, con un sobreajuste notable. Aun así, se puede apreciar que los que ofrecen mejor puntuación son `LogisticRegression()` y `MultinomialNB()` teniendo en cuenta este posible sobreajuste. Como se puede apreciar, la curva del modelo `DecisionTreeClassifier()` y de `linearSVC()` son las menos favorables y la de `DecisionTreeClassifier()` la que tiene una forma más lineal en lugar de curva.

Esto puede ser debido a la forma en qué detectan patrones los diferentes modelos, la forma de cómo tratan los datos o la complejidad de cada modelo. En algunos casos como `DecisionTreeClassifier()`, también puede ser debido a un umbral de clasificación muy equilibrado que hace que se asignen la misma cantidad de clases tanto como positivas como negativas. De todas formas, durante el proceso de calibrado de hiper parámetros podremos ver qué ocurre y si realmente se puede solucionar mediante el uso de hiper parámetros u otro tipo de técnicas.

5.2 Calibración de hiper parámetros y resultados

Después de haber visualizado las distintas métricas y curvas de precisión-recall y ROC, se pasará a visualizar las curvas de aprendizaje de cada uno de los modelos, especificando los hiper parámetros utilizados.

Solamente serán especificados los hiper parámetros que han mejorado el rendimiento de los modelos y los acompañarán una justificación de su elección. Para esto se han probado todos los que, a mi criterio, tendría sentido utilizar.

Finalmente, se compararán los resultados obtenidos con los que se han mostrado previamente, junto con las curvas de aprendizaje añadidas en el Anexo 5, que son las obtenidas sin el uso de parámetros modificados, para poder ver con mayor claridad su respectiva mejora.

Antes de comenzar, y tal y como se puede apreciar en el Anexo 5, los modelos están experimentando un problema de sobreajuste a los datos de entrenamiento ya sea porque no se ha aplicado regularización sobre el conjunto de datos o por el escaso número de muestras. Por ese motivo se han utilizado un número bajo de características en los vectorizadores ya que un número alto hacía pronunciar el sobreajuste sobre el conjunto.

5.2.1 Linear SVC

A continuación se verán los parámetros del modelo `linearSVC()` que han sido modificados para mejorar su rendimiento.

Hiper parámetros modificados	Valor
`C`	`0.2`
`fit_intercept`	`True`
`dual`	`False`
`multi_class`	`'crammer_singer'`
`class_weight`	`{0:0.6, 1:0.4}`
`random_state`	`42`

Estos han sido los parámetros utilizados que han mejorado el rendimiento del modelo por los siguientes motivos:

- El parámetro de regularización "`C`" es inversamente proporcional al nivel de penalización por los errores de clasificación. La modificación se ha basado en los diferentes valores de métricas utilizadas, priorizando el f1-score y buscando un valor que sea equilibrado para las dos clases. mejorando así la clasificación. El valor por defecto era 1 y se ha bajado a 0,2, penalizando más los errores y siendo más restrictivo a la hora de clasificar una muestra como positiva, mejorando así el rendimiento.

- El parámetro "`fit_intercept`" se utiliza para tener en cuenta el sesgo durante el entrenamiento del modelo, considerándolo un factor importante para que el modelo no haga tantas suposiciones.

- Se utiliza la formulación primal en lugar de la dual, ya que según scikit learn, para casos en los que el número de muestras sea mayor que el número de características se incrementa la eficiencia.

- Se utiliza '`crammer_singer`' ya que considerando todas las clases simultáneamente se obtiene un mejor resultado, ya sea por la cantidad de muestras o por contar con más muestras de una clase que de otra.

- Finalmente se ajusta el peso de las clases con "`class_weight`" para que la clase de comentarios negativos, de la cual se disponía de menos muestras, tenga más peso a la hora de clasificar ya que en un problema de este tipo la clase negativa debería tener más importancia, sobre todo si hay menos clases de este tipo. Se ha ajustado con esos valores para mantener el f1-score equilibrado entre las dos clases.

- El parámetro "`random_state`" tiene el valor 42 simplemente para la reproducibilidad de los resultados durante las diferentes pruebas pero no afecta al rendimiento. Otros parámetros han sido probados pero los valores óptimos que cambian el rendimiento del modelo son los mencionados en la tabla.

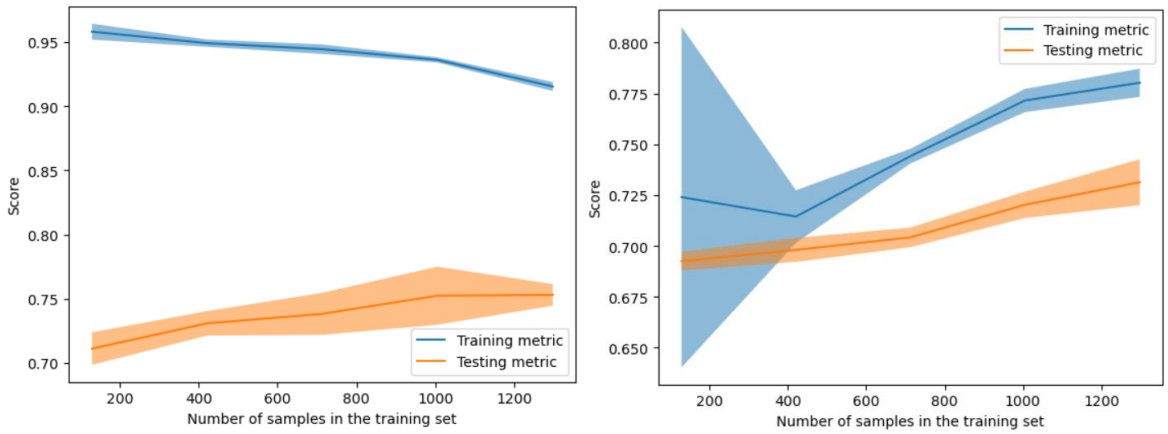


Figura 24: *Comparativa de las curvas de aprendizaje antes del calibrado de hiper parámetros (izquierda) y después (derecha) del modelo* `LinearSVC()` *sobre el conjunto de entrenamiento.*

Después de calibrar los hiper parámetros mencionados se puede apreciar que el sobreajuste que presentaba al principio ha sido solventado en cierta forma. El espacio azul difuminado que vemos en el gráfico derecho es la varianza del modelo y como se puede apreciar al principio del entrenamiento, es mayor, pero a medida que el umbral es más restrictivo va desapareciendo, volviendo a acercarse al sobreajuste.

De todas formas, las curvas entre el train y el test se han acercado lo que quiere decir que el modelo ha podido generalizar mejor con los nuevos hiper parámetros y la varianza en el conjunto de prueba ha sido reducida también.

Si se quisiera ver el resto de métricas, se pueden encontrar en los Anexo 6, 7 y 8.

5.2.2 MultinomialNB

En este modelo, ningún parámetro ha mejorado su rendimiento. El parámetro "`alpha`" que mide la suavización y por lo tanto la regularización da el mejor rendimiento con su valor por defecto, al igual que "`fit_prior`" y "`class_prior`" que con la proporción de asignar probabilidades a clases que tienen por defecto dan el mejor resultado posible.

Los resultados de las diferentes métricas se pueden encontrar en los Anexo 6, 7 y 8. Aunque como se ha comentado, no han variado ya que por la forma en que este modelo trata los datos no se ha encontrado una combinación que lo mejore. Aunque con otro tipo de alternativas podría haber mejorado, refiriéndose a técnicas de regularización sobre el conjunto de datos o data augmentation.

5.2.3 Logistic Regression

A continuación se verán los parámetros del modelo `LogisticRegression()` que han sido modificados para mejorar su rendimiento.

Hiper parámetros modificados	Valor
`solver`	`liblinear`
`fit_intercept`	`True`
`dual`	`True`
`random_state`	42

Se ha modificado el parámetro `"solver"` para que se utilice `"liblinear"` que, siguiendo la recomendación de scikit learn, funciona mejor para datasets pequeños, como es el caso. También se ha modificado el parámetro `"fit_intercept"` para tener en cuenta el sesgo durante el proceso de entrenamiento.

El parámetro "`dual`" está a True para que utilice la formulación dual, ya que funciona mejor con una `penalty` del tipo `"l2"` y un `solver "liblinear"`. Por otra parte, la formulación primal da mejores resultados cuando el número de muestras es mayor que el número de características y no es el caso.

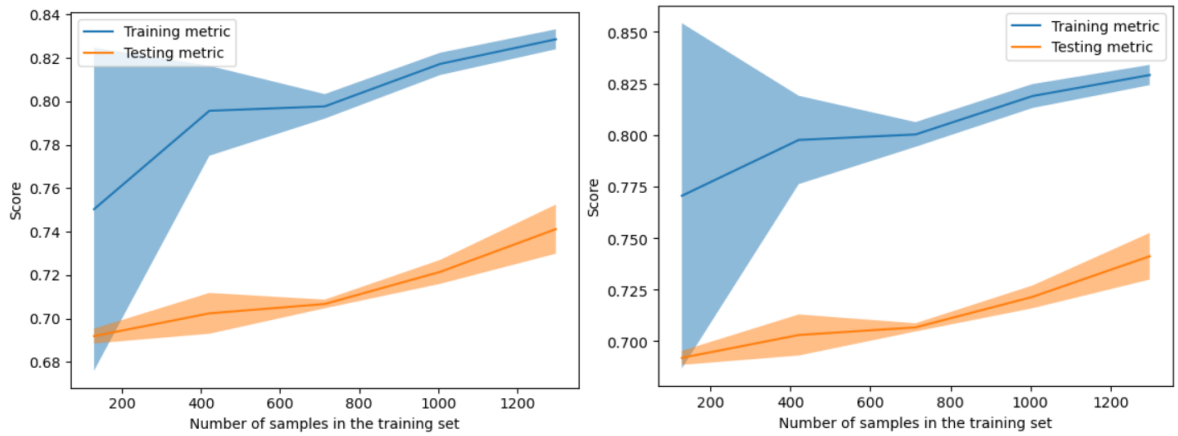


Figura 25: *Comparativa de las curvas de aprendizaje antes del calibrado de hiper parámetros (izquierda) y después (derecha) del modelo* `LogisticRegression()` *sobre el conjunto de entrenamiento.*

En este caso, el modelo `LogisticRegression()`, desde un principio presentaba buenos resultados, ya sea por la forma en cómo distribuye probabilidades durante el entrenamiento con técnicas de suavizado o por su función sigmoidea que debido a la cantidad de datos da mejores resultados.

Por este motivo, la mejora no es notable, aunque se ha conseguido mejorar ligeramente la precisión y reducido el sobreajuste del conjunto de entrenamiento, aunque finalmente tiende a éste.

De igual forma que con modelos anteriores, el resto de los resultados con las diferentes métricas calculadas se pueden encontrar en los Anexo 6, 7 y 8.

5.2.4 Decision Tree Classifier

A continuación se verán los parámetros del modelo `DecisionTreeClassifier()` que han sido modificados pero mejorar su rendimiento.

Hiper parámetros modificados	Valor
`max_depth`	6
`min_samples_leaf`	4
`max_leaf_nodes`	14
`random_state`	42

El parámetro "`max_depth`" controla la profundidad máxima del árbol, es decir, se puede controlar el número de niveles de decisión que tendrá. De esta forma se puede reducir su complejidad y el sobre ajuste. Se modifica a 6 porque con este valor se consigue la complejidad suficiente para tener en cuenta las dos clases, relativamente por igual, y además se consigue reducir la complejidad.

Para decidir qué valor tienen los siguientes parámetros, se parte de la idea de querer conseguir una proporción equilibrada entre complejidad y sencillez. Por ello, se tienen en cuenta los siguientes valores para los parámetros:

En `min_samples_leaf,el rango de [2,5]` y de `max_leaf_nodes de [10,20]` y se utiliza la función `GridSearchCV()` que busca la mejor combinación de parámetros y utiliza validación cruzada para comprobar su precisión con el modelo.

Después de comprobar que realmente mejora de la forma deseada, es decir, se reduce el sobreajuste y se equilibra la clasificación de clases, se establecen estos parámetros óptimos para el modelo.

El parámetro "`min_samples_leaf`" hace que en cada nodo se requiera 4 muestras de una clase para que se pueda continuar dividiendo en otros dos nodos y evaluando otra decisión, si no hay cuatro muestras de una clase ese nodo se convertirá en un nodo hoja. Hacer esto permite que el árbol se divida en hojas con un número pequeño de muestras y que se pueda llegar a tener más hojas y decisiones diferentes, aumentando así la complejidad del árbol.

Se complementa con el parámetro `"max_leaf_nodes"` que establece un número máximo de 14 hojas en el árbol reduciendo así el posible aumento de la complejidad provocado por `"max_leaf_nodes"`.

Finalmente, el parámetro `"random_state"` está simplemente para la reproducibilidad del experimento durante las pruebas.

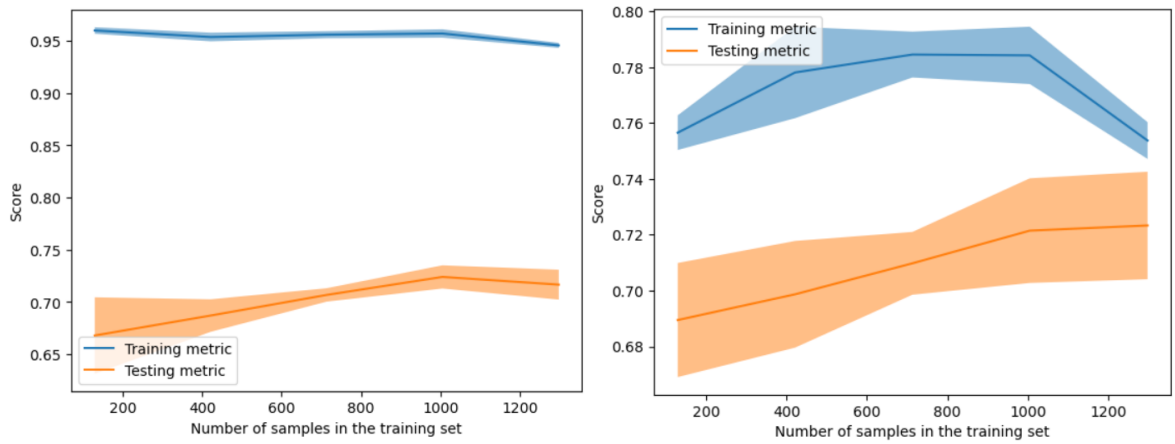


Figura 26: *Comparativa de las curvas de aprendizaje antes del calibrado de hiper parámetros (izquierda) y después (derecha) del modelo* `DecisionTreeClassifier()` *sobre el conjunto de entrenamiento.*

En el caso del modelo `DecisionTreeClassifier()`, se puede apreciar con mayor facilidad que con otros modelos, la diferencia entre el antes y el después del calibrado de hiper parámetros, ya que se ha reducido el sobreajuste considerablemente.

De todas formas, la incertidumbre en los dos conjuntos, tanto el de prueba como el de entrenamiento, ha aumentado. Esto puede ser debido a la reducción de complejidad del árbol que por un lado hace menos complejo el árbol, pero de esta forma se está obteniendo un árbol más sencillo que puede tener más errores al haber reducido el número de decisiones posibles.

Esta forma convexa en el gráfico de la derecha significa que el rendimiento del modelo mejora a medida que aumenta el tamaño del conjunto de entrenamiento, pero llega a un nivel óptimo en el que el error se estabiliza y llegados a este punto baja. Esto nos puede decir que para el conjunto de datos del cual se dispone puede funcionar bien, pero si fuera mayor habría que ajustar los parámetros para estabilizar esta recta.

El resto de los resultados con las diferentes métricas calculadas se pueden encontrar en los Anexo 6, 7 y 8.

5.2.5 Random Forest Classifier

A continuación se verán los parámetros del último de los modelos, `RandomForestClassifier()`, que han sido modificados pero mejorar su rendimiento.

Hiper parámetros modificados	Valor
`max_depth`	`10`
`n_jobs`	`3`
`n_estimators`	`120`
`min_samples_leaf`	`2`
`max_leaf_nodes`	`12`
`class_weight`	`{0:0.56, 1:0.44}`
`random_state`	`42`

En este caso, `RandomForestClassifier()` implementa un bosque de `DecisionTreeClassifier()s`, por lo tanto los parámetros `"max_depth"`, `"min_samples_leaf"` y `"max_leaf_nodes"` se han escogido por el mismo motivo que en el caso anterior, se incrementa el nivel de complejidad con "`min_samples_leaf`" pero se reduce con `"max_leaf_nodes"` y se regula con "`max_depth`".

El parámetro `"n_estimators"` define el número de estimadores (árboles) que se utilizan para entrenar el modelo `RandomForestClassifier()`. El valor por defecto es 100 pero igual que en el caso anterior, con el uso de `GridSearchCV()` se encuentra el valor óptimo.

El parámetro `"class_weight"` permite asignar manualmente un peso a cada clase. Igual que se ha hecho con los otros modelos se ha probado cada uno de los parámetros y, este en concreto, se ha probado por el motivo de disponer de menos muestras etiquetadas como ofensivas (0) en proporción con las no ofensivas(1). De esta forma, se balancea esta asignación de pesos obteniendo un mejor resultado.

Por último, el parámetro `"n_jobs"` se utiliza simplemente para utilizar más núcleos del procesador y agilizar el proceso de entrenamiento del modelo ya que tiene un coste computacional alto frente al de otros modelos. El parámetro `"random_state"`, como en casos anteriores simplemente tiene ese número para la reproducibilidad de los resultados durante los experimentos.

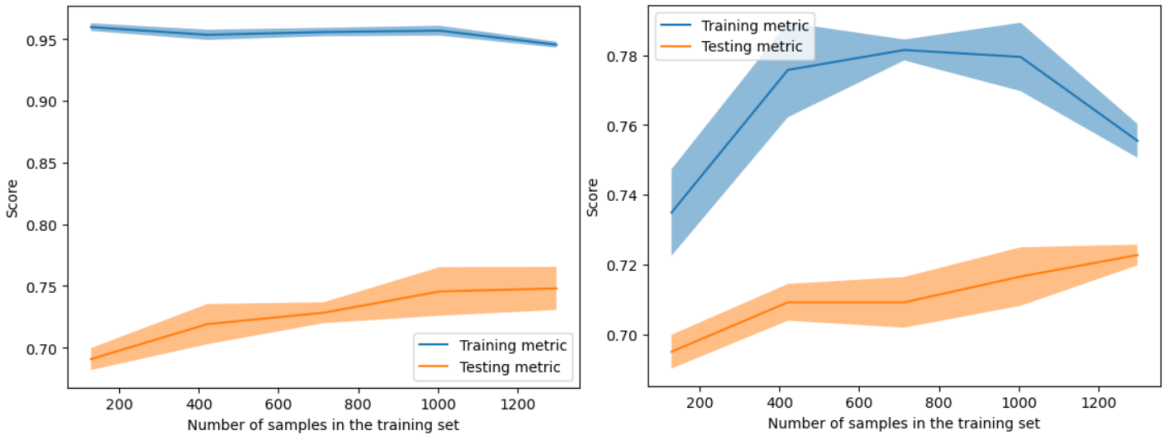


Figura 27: *Comparativa de las curvas de aprendizaje antes del calibrado de hiper parámetros (izquierda) y después (derecha) del modelo* `RandomForestClassifier()` *sobre el conjunto de entrenamiento.*

Como se puede apreciar en la figura se ha conseguido reducir significativamente el sobreajuste que presentaba el modelo `RandomForestClassifier()`. De igual forma que con `DecisionTreeClassifier()` el gráfico de la derecha presenta una curva convexa en los datos de entrenamiento. Esto puede ser debido a cómo el modelo trata los datos actuales, pero en casos donde la cantidad de datos aumente se debería reconfigurar la combinación de hiper parámetros.

El resto de los resultados con las diferentes métricas calculadas se pueden encontrar en los Anexo 6, 7 y 8.

5.3 Modelo escogido

Finalmente, después de haber visto los diferentes resultados de los modelos estudiados, para decidir cuál de ellos será escogido para clasificar la polaridad de comentarios en Instagram hacia futbolistas profesionales, se muestra el siguiente gráfico:

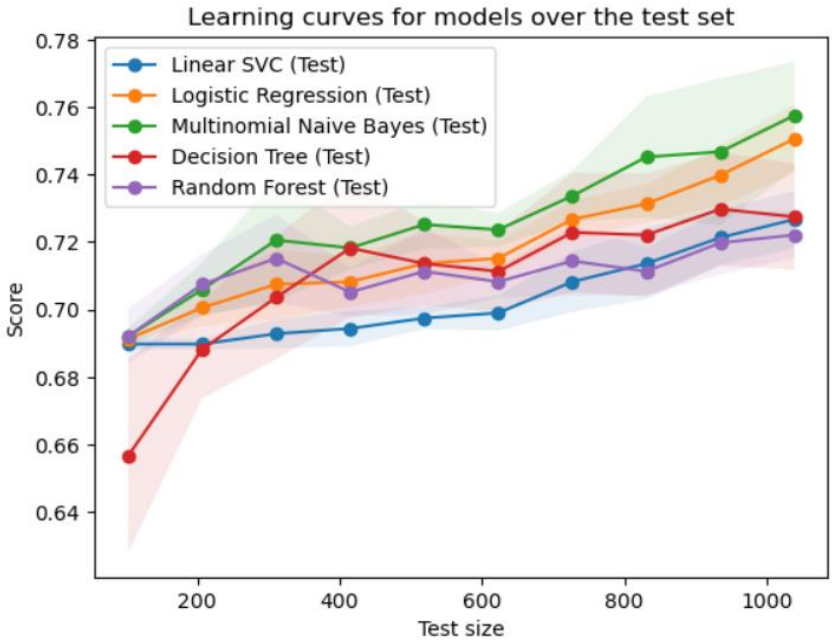


Figura 28: *Comparativa de las curvas de aprendizaje sobre el conjunto de prueba*

Como se puede apreciar en este gráfico, finalmente, el modelo que presenta mejores resultados sobre el conjunto de prueba es el modelo `RandomForestClassifier()`, esto puede ser debido a la forma de capturar relaciones complejas entre las diferentes palabras del corpus y la forma de tratar las características para la clasificación. Ha demostrado que en datos no vistos ofrece una mejor puntuación y no solo eso, sino que no presenta picos significativos durante la predicción.

Este modelo será escogido como clasificador en este problema de análisis de sentimientos teniendo en cuenta su puntuación y su escalabilidad ya que, si en un futuro el trabajo piensa ser continuado y ampliar el conjunto de datos, este modelo, según lo estudiado a priori, puede ser el que mejores resultados proporcione. Esto, puede no ser del todo cierto ya que se pueden aplicar técnicas de regularización sobre el conjunto de datos o técnicas de data augmentation para enriquecer el corpus, pero como digo, esto se podrá investigar más a fondo en un futuro.

6. Conclusiones y trabajo futuro

6.1 Conclusión y modelo escogido

El primer objetivo de este trabajo consistía en la creación de forma autónoma de un conjunto de datos que pueda ser utilizado para entrenar los diferentes modelos a estudiar. Desde la recolección de datos, pasando por el proceso de limpieza de datos y acabando con el preprocesado de estos datos.

El segundo objetivo consistía en la investigación previa de 5 modelos de la biblioteca de aprendizaje automático scikit-learn, cuyo resultado se puede ver reflejado en el apartado 4 de este trabajo, la implementación.

Finalmente, como se ha podido apreciar en previos apartados, el modelo escogido para el problema de análisis de sentimientos, que consiste en determinar la polaridad de un comentario escrito hacia un futbolista profesional en la red social Instagram, es el modelo `RandomForestClassifier()`. Los motivos de su elección están comentados en el apartado 5.3 en base a los diferentes estudios realizados.

6.2 Problemas encontrados

En este apartado se explicarán los problemas encontrados a lo largo del proceso de desarrollo de este trabajo. Para comenzar, como viene siendo habitual en la realización de este trabajo, se explicarán en primer lugar los problemas encontrados durante el proceso de recolección de datos.

6.1.1 Instagram API

EL notebook donde se encontraron estos problemas fue `“Instagram_API_comments.ipynb”`. Como ya se ha mencionado en anteriores apartados, la primera intención era utilizar la API proporcionada por para acceder a los comentarios de una forma óptima. El problema aquí fue que para poder realizar llamadas a esta API había que cumplir unos requerimientos como pueden ser: Solamente acceder a datos de tu propio perfil de Instagram, encontrar la URL correcta ya que habían diferentes versiones de la API, la creación de un token y una serie de pasos más que no llevaban a ningún lugar ya que para poder acceder a información ajena se necesitaba utilizar otro tipo de API de Instagram (graph API) y al no disponer de tiempo cambié de estrategia.

En este mismo fichero también se encontraron problemas durante la implementación de la segunda estrategia que consiste en “scrapear” los comentarios. Aunque las funciones para extraer los comentarios estaban creadas y eran funcionales, tenían algunos errores que estaban relacionados con la velocidad de respuesta de la página web o en cómo se trataban algunos elementos HTML. Estas funciones iteraban por cada jugador extrayendo grandes cantidades de comentarios y podían estar de 3 a 6 horas ejecutándose por lo que dejaba el programa ejecutándose al despertar y, al volver de mi jornada laboral, comprobaba si había habido algún problema, si lo había habido la ejecución se paraba y perdía el progreso.

La manera de refinar estas funciones seguía la metodología de prueba y error, al inicio muchos errores y a medida que iba corrigiéndolos también iba entendiendo mejor que podía fallar y que no. Durante este proceso de refinamiento, en lugar de extraer comentarios de los 25 jugadores iba extrayendo de 3 en 3, de esta forma creí que podía reducir la tasa de fallo al reducir también la duración del programa.

Finalmente, más que un problema en sí sería un apunte que remarcar. Hoy en día si ejecuto el programa de `“Instagram_API_comments.ipynb”` fallará porque el código de la web de instagram varía constantemente y al tener que acceder a cada elemento HTML por separado ya que no se dispone de la API, este programa está escrito para elementos antiguos de la página web de Instagram que, en su momento eran funcionales pero actualmente han sido reemplazados.

6.2.2 Implementación BERT

Durante la implementación del modelo BERT, en el fichero `“BERT-fine-tuning-and-evaluation.ipynb”`, encontré problemas a la hora de entrenar de modelo ya que torch, cuando ha de realizar cálculos, por defecto, asigna la ejecución de éstos al procesador, `torch.device(‘cpu’).`

Después de investigar, concluí que era más eficiente utilizar la tarjeta gráfica (gpu) para realizar los cálculos ya que tal y como están diseñadas algunas tarjetas gráficas de la marca NVIDIA como es mi caso, permiten paralelizar procesos computacionales. A muy alto nivel, si comparamos el funcionamiento de una tarjeta gráfica con el de un procesador, pongamos como ejemplo que al hacer 10 sumas del estilo de 2+2, en un procesador, estas sumas se ejecutarían una detrás de otra como si estuvieran en una cola. En cambio, una tarjeta gráfica que tenga una arquitectura CUDA realizaría estas operaciones de forma paralela reduciendo de forma considerable el tiempo de ejecución.

Una vez visto esto, decidí utilizar `torch.device(‘cuda’)` en lugar de la opción ‘cpu’ pero al ejecutar la función de entrenamiento del modelo, recibía un mensaje de error porque se estaba intentando utilizar pythorch (llamando a CUDA) cuando esto no era posible debido a que la versión de CUDA no estaba siendo detectada en el programa. Después de investigar un tiempo, averigüé que era un problema de compatibilidad entre versiones de pytorch y de CUDA debido a que la versión más reciente de pytorch era la 11.8 pero la versión de CUDA de mi tarjeta gráfica era la 12.1. Por este motivo,

tuve que hacer “downgrade” de la versión de CUDA de mi tarjeta gráfica a la 11.8 para poder así utilizarla y aumentar la eficiencia del programa.

Para hacernos una idea, hice la prueba de cuánto tiempo tardaba en ejecutar la parte de entrenamiento, predicción y evaluación del modelo con un número de batch_size de 32 con un dataset de 1.000.000 de ejemplos. Primero utilicé el procesador y tardó unos 15-20 minutos como mínimo, después probé con la tarjeta gráfica y los cálculos habían terminado en 10 segundos. Finalmente utilicé la opción ‘cuda’ para realizar este proceso.

6.2 Trabajo futuro

Como trabajo futuro a este proyecto hay ciertos aspectos que tendría en cuenta. En primer lugar, me esforzaría en crear un conjunto de datos de mayor calidad, ya sea ampliando el que actualmente se utiliza o buscando otras alternativas. También utilizaría otras técnicas de preprocesado de datos ya que considero que si se aplicaran de forma correcta técnicas como lemmatization o stemming se podría llegar a tener un conjunto de datos de mayor calidad.

En segundo lugar se podría indagar más en las últimas tecnologías y modelos que están al nivel de estado del arte dedicados al Natural Language Processing, que utilicen algoritmos de memoria a largo plazo y que están mayormente capacitados para este tipo de problemas a diferencia de como ocurre con los modelos utilizados, que no se dedican únicamente a problemas de análisis de sentimiento, sino que se pueden utilizar en otro tipo de problemas, cosa que creo que hace no desarrollar el mejor rendimiento.

En tercer y último lugar, creo que, aparte de explorar nuevos modelos para mejorar la clasificación, se deberían explorar técnicas de Data augmentation que permitieran mejorar el conjunto de datos y, en algún caso, solucionar o suavizar problemas que luego puedan afectar negativamente a los modelos como el sesgo, la varianza, el desbalance de clases y algunos más.

7. Bibliografía

[1] F. M. Plaza-Del-Arco, A. Montejo-Ráez, L. Alfonso, U. Ure˜ureña-López, and M.-T. Martín-Valdivia, “OffendES: A New Corpus in Spanish for Offensive Language Research,” pp. 1096–1108, doi: 10.26615/978-954-452-072-4_123.

[2] J. Barnes, R. Klinger, and S. Schulte im Walde, “Assessing State-of-the-Art Sentiment Models on State-of-the-Art Sentiment Datasets,” pp. 2–12.

[3] E. Gilbert and C.J.Hutto, “View of VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text,” 2014. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399. [Accessed: 08-Jun-2023].

[4] G. Miller, “WordNet: A Lexical Database for English,” 1995.

[5] D. Khurana, A. Koli, K. Khatter, and S. Singh, “Natural Language Processing : State of The Art , Current Trends and Challenges Department of Computer Science and Engineering Accendere Knowledge Management Services Pvt . Ltd ., India Abstract,” *Sentim. Anal. has become one Most profound Res. areas with increasing growth Soc. media web. Nowadays, millions users Exch. their views, ideas, expressions, Feel. Opin. Soc. media like twitter Facebook. Se*, no. Figure 1, 2017.

[6] Y. Goldberg and O. Levy, “word2vec Explained: deriving Mikolov et al.’s negative-sampling word-embedding method,” no. 2, pp. 1–5, 2014.

[7] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, no. June 2018, pp. 1532–1543, 2014, doi: 10.3115/v1/d14-1162.

[8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, no. Mlm, pp. 4171–4186, 2019.

[9] “The Current State of the Art in Natural Language Processing (NLP) — AICromo.” [Online]. Available: https://aicromo.com/ai-blogs/the-current-state-of-the-art-in-natural-language-processing-nlp. [Accessed: 09-Jun-2023].

[10] A. Vaswani *et al.*, “Attention is all you need,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 5999–6009, 2017.

[11] “scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 09-Jun-2023].

[12] S. Robertson, “Understanding inverse document frequency: On theoretical arguments for IDF,” *J. Doc.*, vol. 60, no. 5, pp. 503–520, 2004, doi: 10.1108/00220410410560582.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[14] Y. Wu *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” pp. 1–23, 2016.

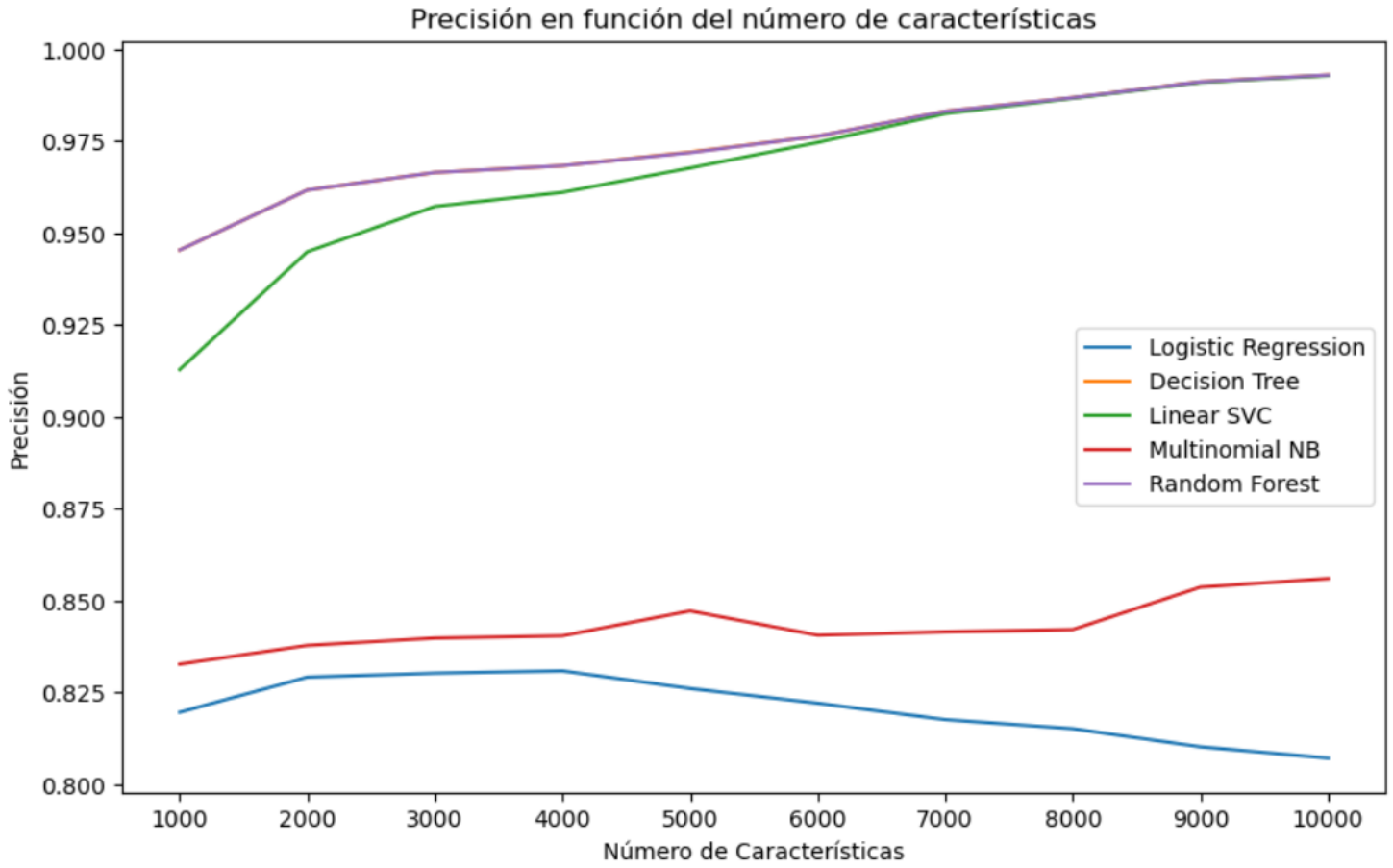
[15] R. Amat, “Máquinas de Vector Soporte (Support Vector Machines, SVMs),” 2017. [Online]. Available: https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines. [Accessed: 09-Jun-2023].

[16] “RPubs - Tarea_1_WAF_Sklearn.” [Online]. Available: https://rpubs.com/William_Astocondor/Tarea_1_WAF_Sklearn. [Accessed: 10-Jun-2023].

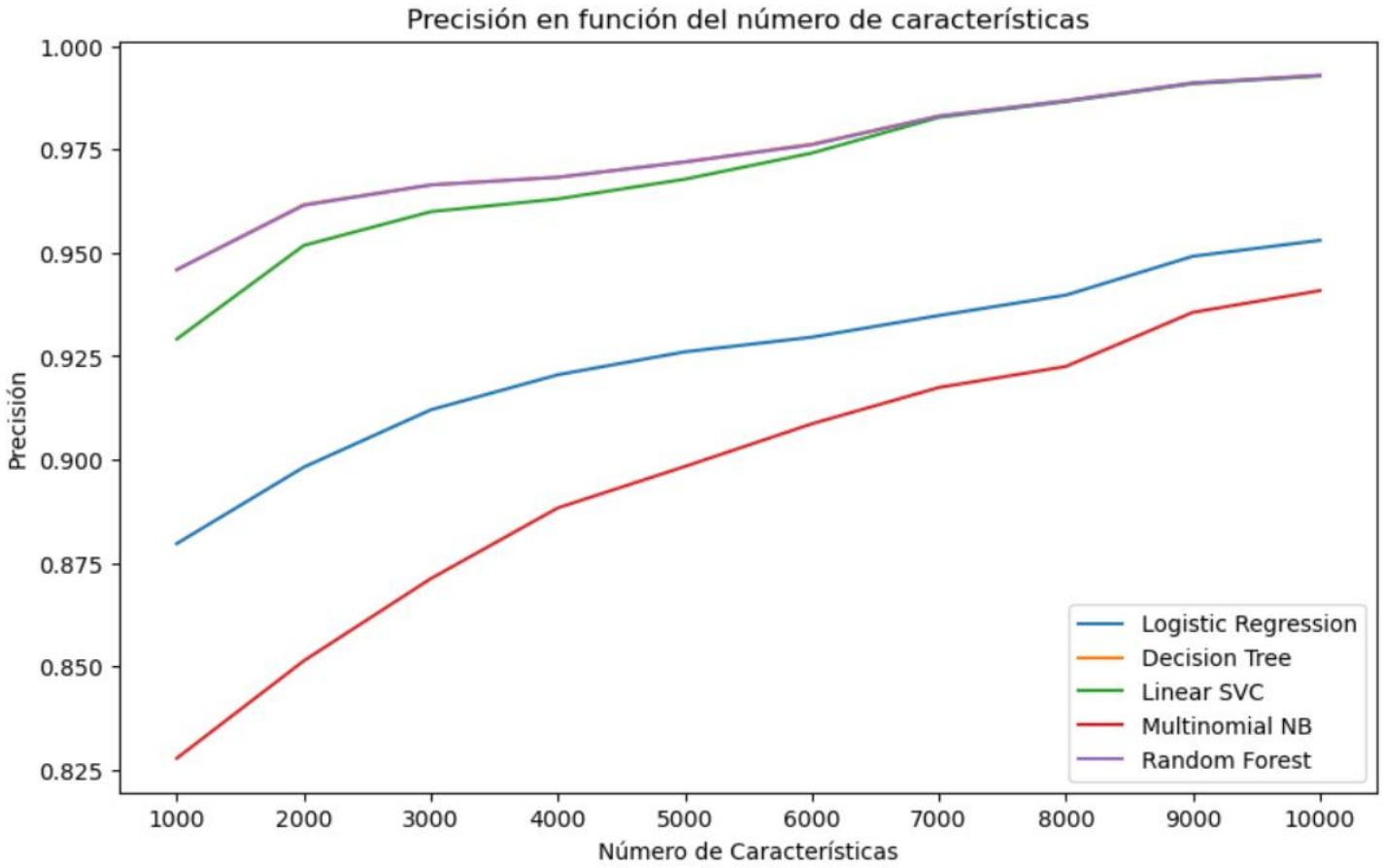
[17] “Random Forest python.” [Online]. Available: https://www.cienciadedatos.net/documentos/py08_random_forest_python.html. [Accessed: 10-Jun-2023].

Anexos

Anexo 1: Comparación de vectorizadores mediante la evaluación de modelos dependiendo del número de características con TF-IDF vectorizer:



Comparación de vectorizadores mediante la evaluación de modelos dependiendo del número de características con CountVectorizer:



Anexo 2: Tablas de comparación de las diferentes precisiones obtenidas dependiendo del vectorizador según los parámetros: modelo, número de características del vectorizador y precisión del modelo.

TF-IDF vectorizer results

	Model	Number of Features	Accuracy
0	Logistic Regression	4000	0.830867
1	Decision Tree	10000	0.992914
2	Linear SVC	10000	0.992760
3	Multinomial NB	10000	0.855976
4	Random Forest	10000	0.992914

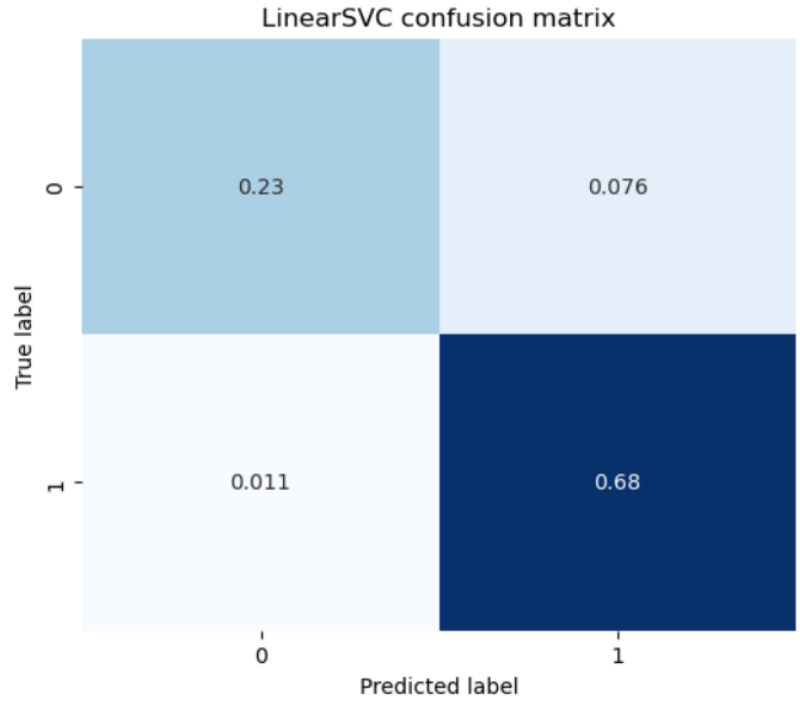
CountVectorizer results

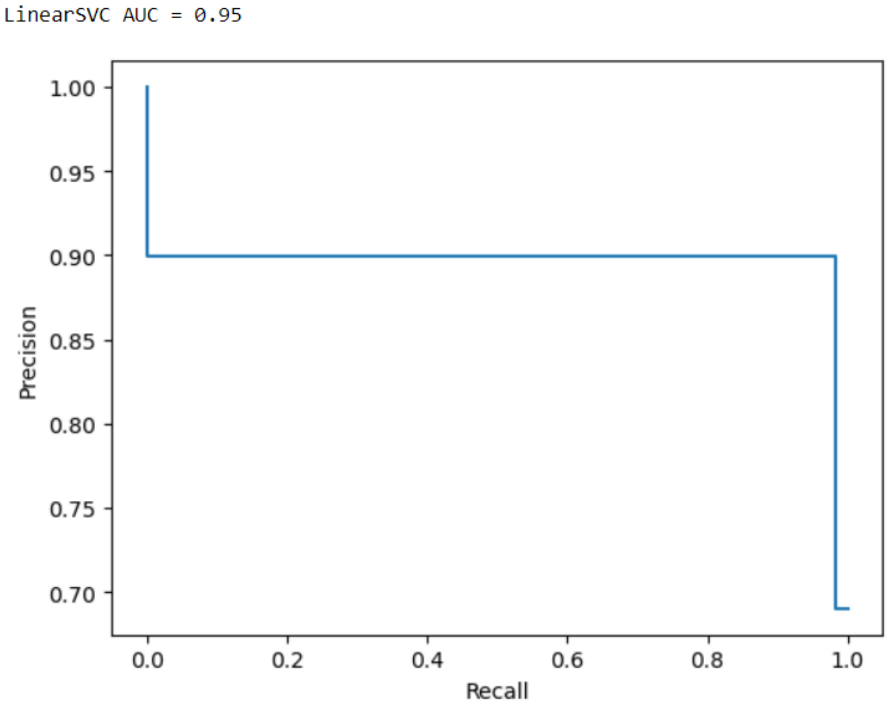
	Model	Number of Features	Accuracy
0	Logistic Regression	10000	0.953019
1	Decision Tree	10000	0.992914
2	Linear SVC	10000	0.992760
3	Multinomial NB	10000	0.940850
4	Random Forest	10000	0.992914

Anexo 3: Precisión por defecto de los modelos con su respectivo vectorizador óptimo representada por la matriz de confusión y su curva de Precision-Recall con la puntuación AUC:

LinearSVC()

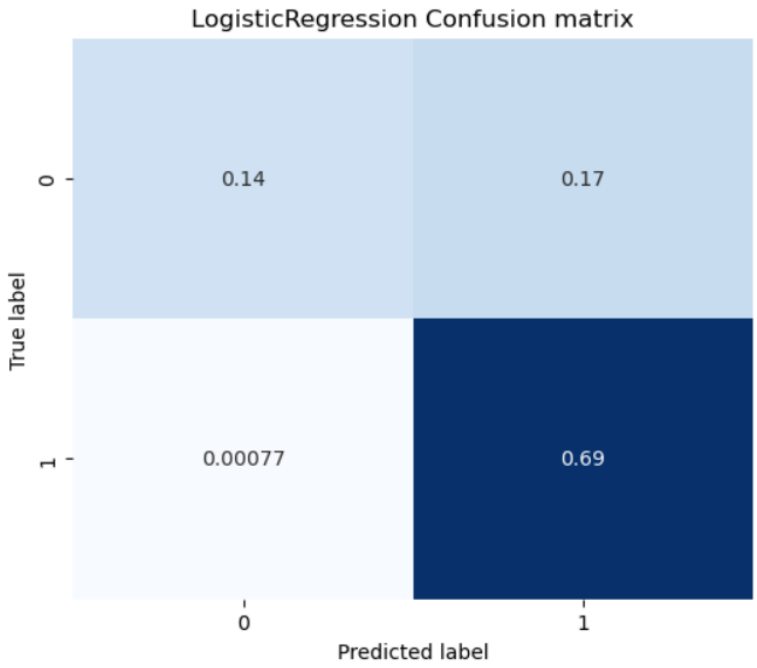
```
Training accuracy =  91.28157732593962
              precision    recall  f1-score   support

           0       0.95      0.75      0.84      2012
           1       0.90      0.98      0.94      4480

    accuracy                           0.91      6492
   macro avg       0.93      0.87      0.89      6492
weighted avg       0.92      0.91      0.91      6492
```



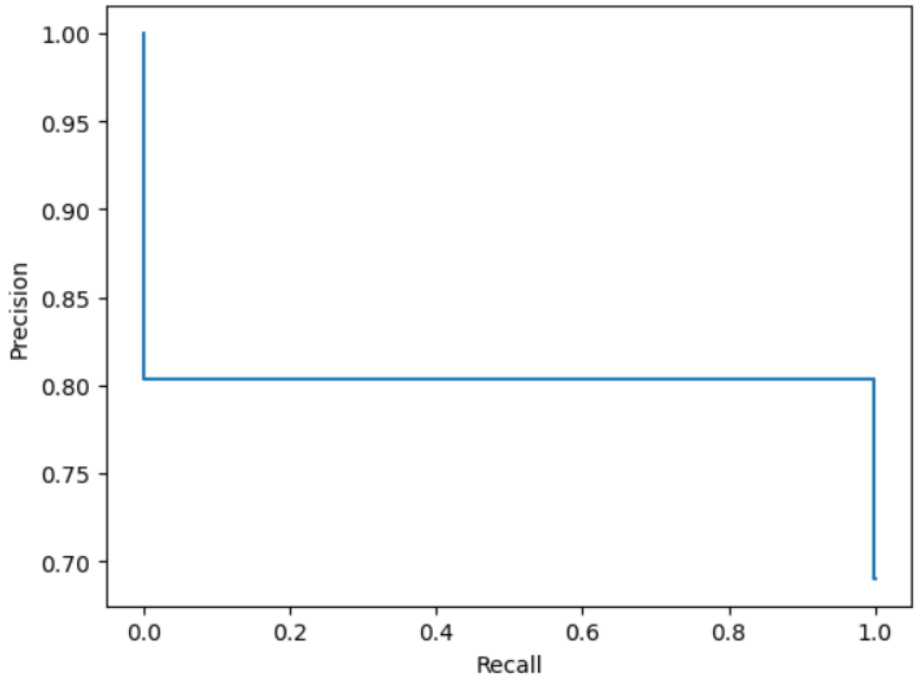


LogisticRegression()

```
Training accuracy =  83.08687615526802
              precision    recall  f1-score   support

           0       0.99      0.46      0.63      2012
           1       0.80      1.00      0.89      4480

    accuracy                           0.83      6492
   macro avg       0.90      0.73      0.76      6492
weighted avg       0.86      0.83      0.81      6492
```



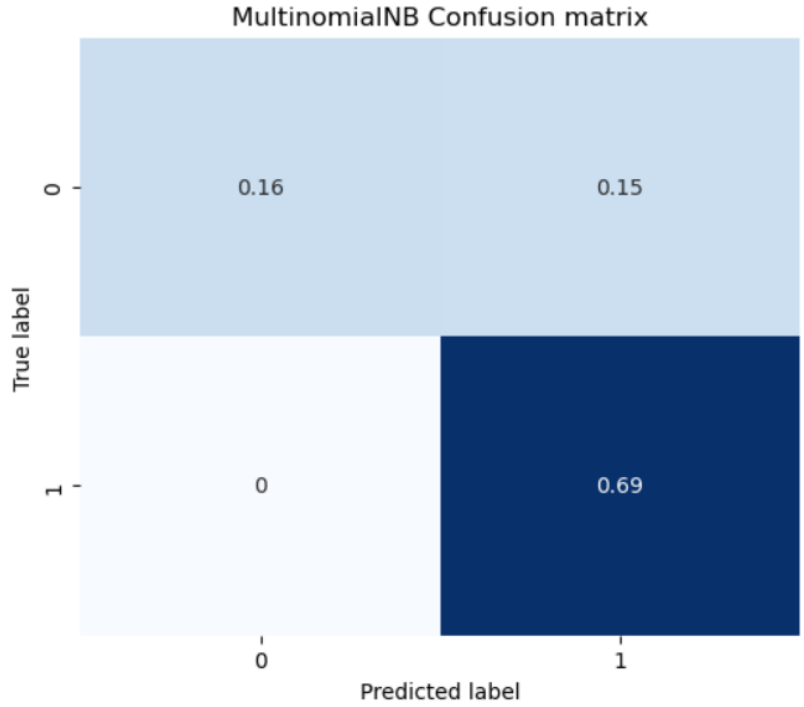
Logistic Regression AUC = 0.90



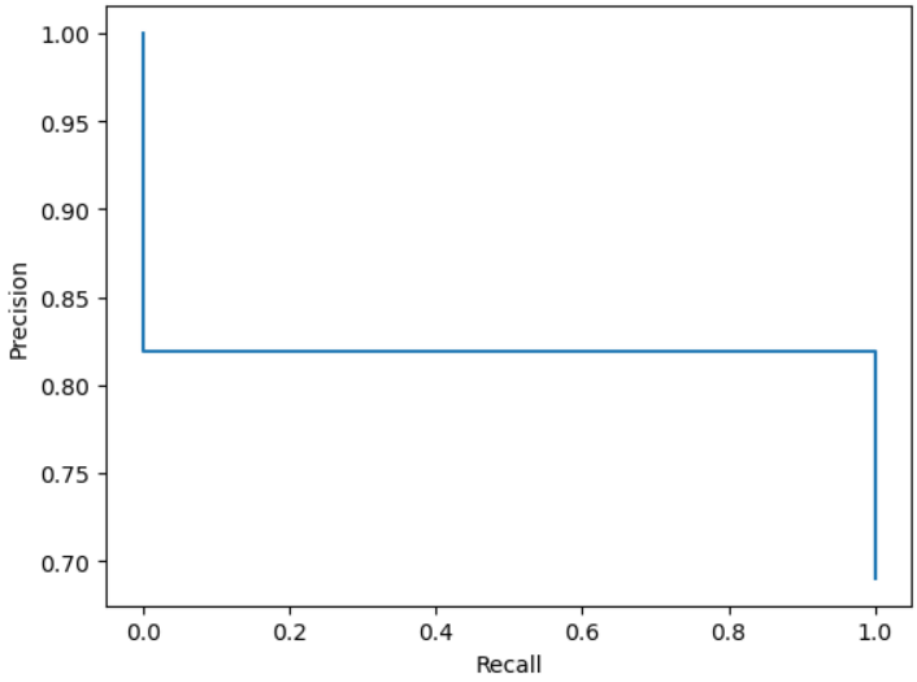
MultinomialNB()

Training accuracy = 84.7196549599507

	precision	recall	f1-score	support
0	1.00	0.51	0.67	2012
1	0.82	1.00	0.90	4480
accuracy			0.85	6492
macro avg	0.91	0.75	0.79	6492
weighted avg	0.87	0.85	0.83	6492

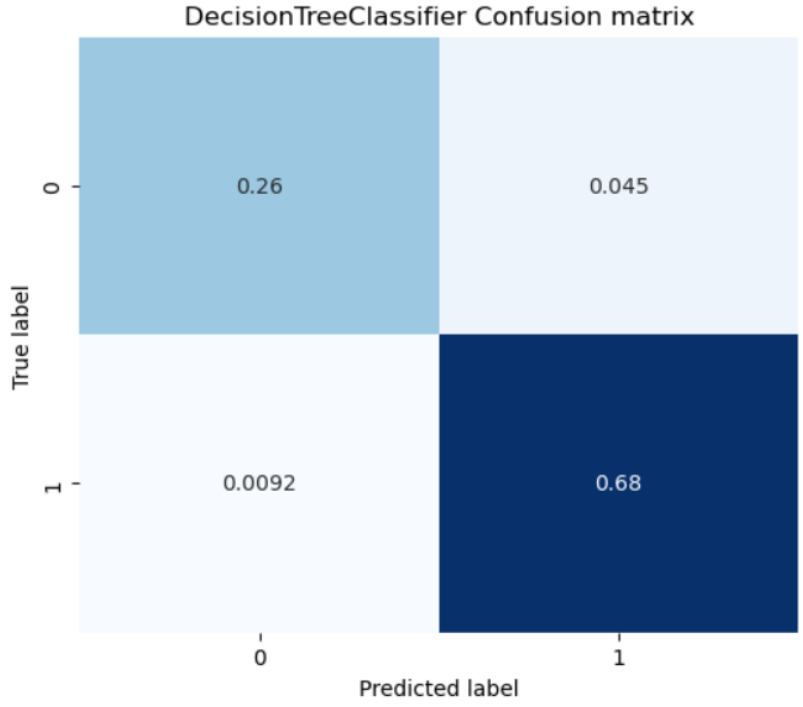


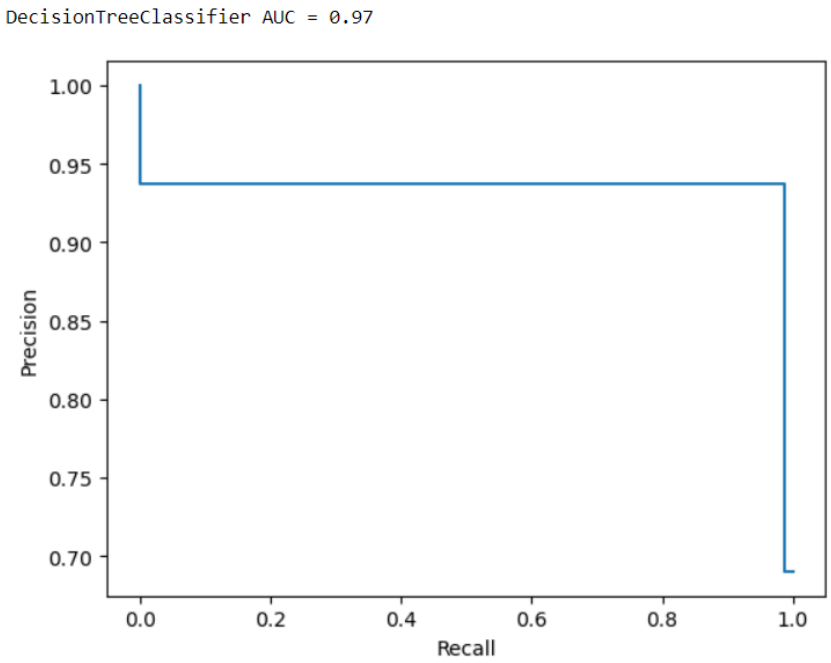
MultinomialNB AUC = 0.91



DecisionTreeClassifier()

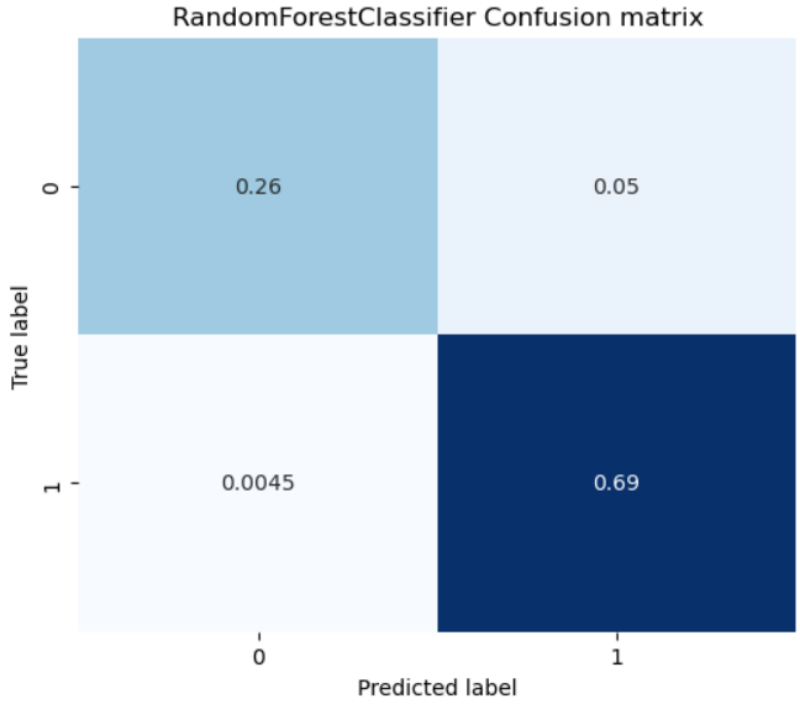
```
Training accuracy =  94.53173136167591
              precision    recall  f1-score   support

           0       0.97      0.85      0.91      2012
           1       0.94      0.99      0.96      4480

    accuracy                           0.95      6492
   macro avg       0.95      0.92      0.93      6492
weighted avg       0.95      0.95      0.94      6492
```

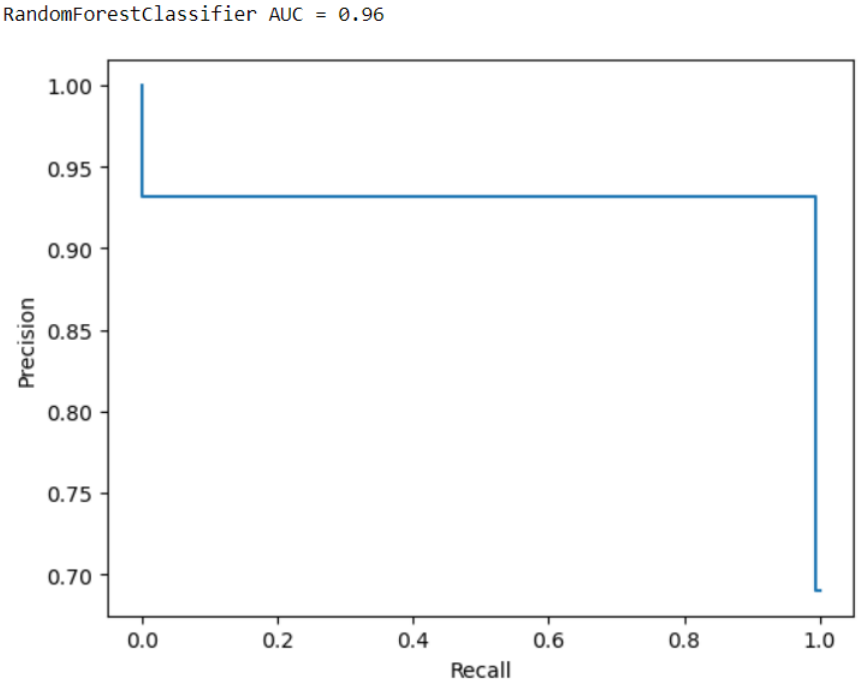




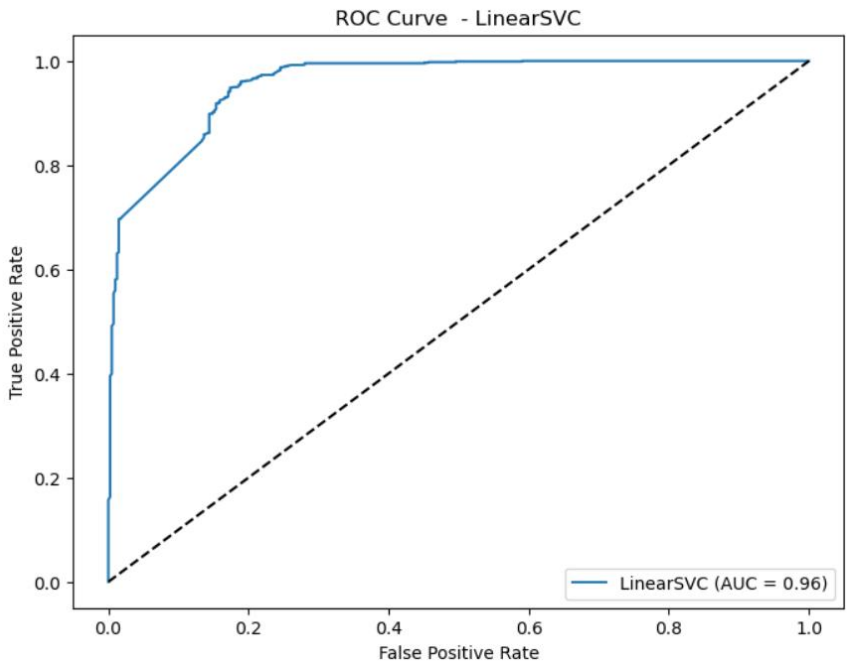
RandomForestClassifier()

```
Training accuracy =  94.53173136167591
              precision    recall  f1-score   support

           0       0.98      0.84      0.90      2012
           1       0.93      0.99      0.96      4480

    accuracy                           0.95      6492
   macro avg       0.96      0.92      0.93      6492
weighted avg       0.95      0.95      0.94      6492
```

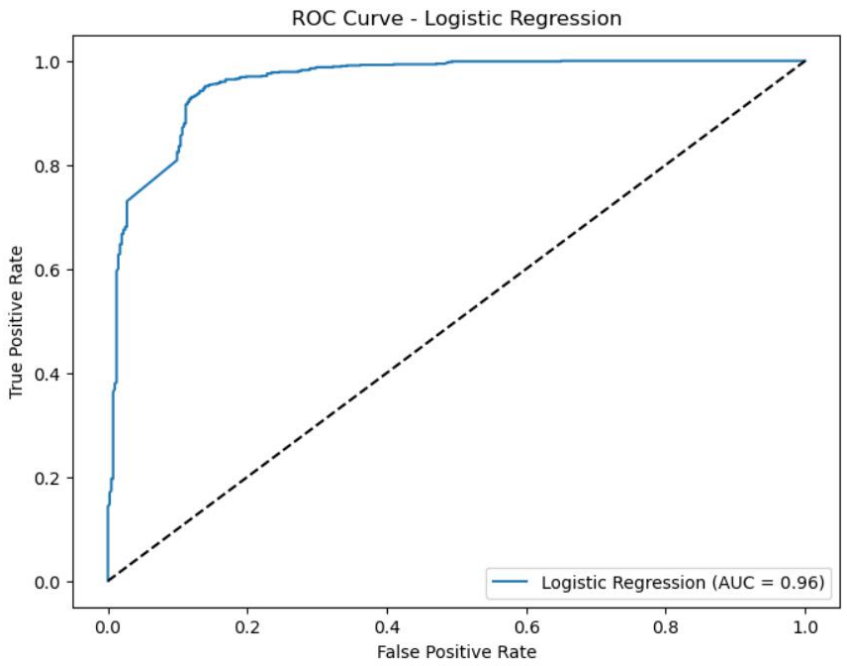




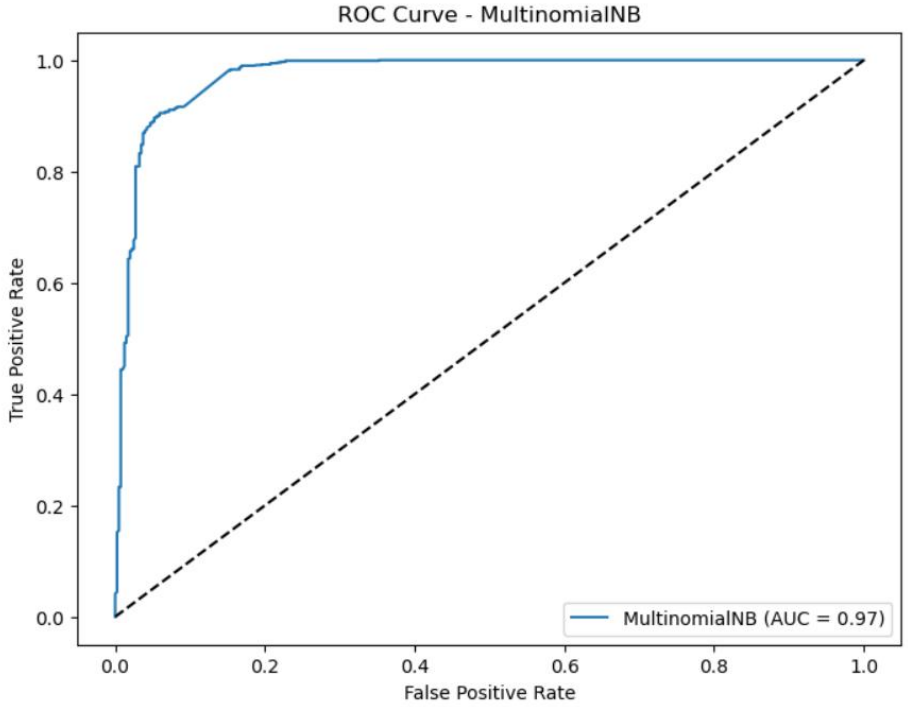
Anexo 4: Gráficas de la curva ROC de cada uno de los modelos:



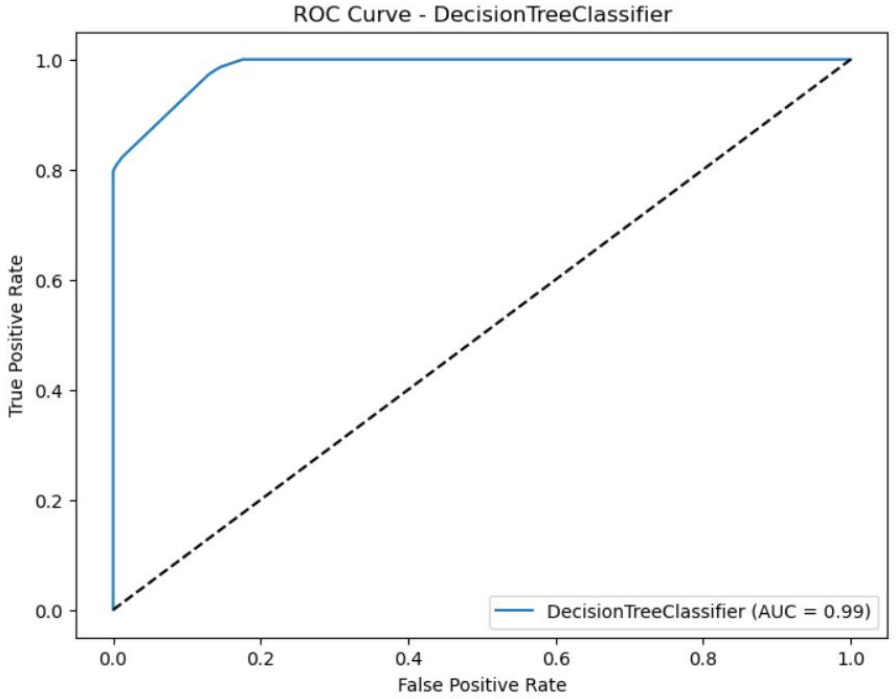
LinearSVC()



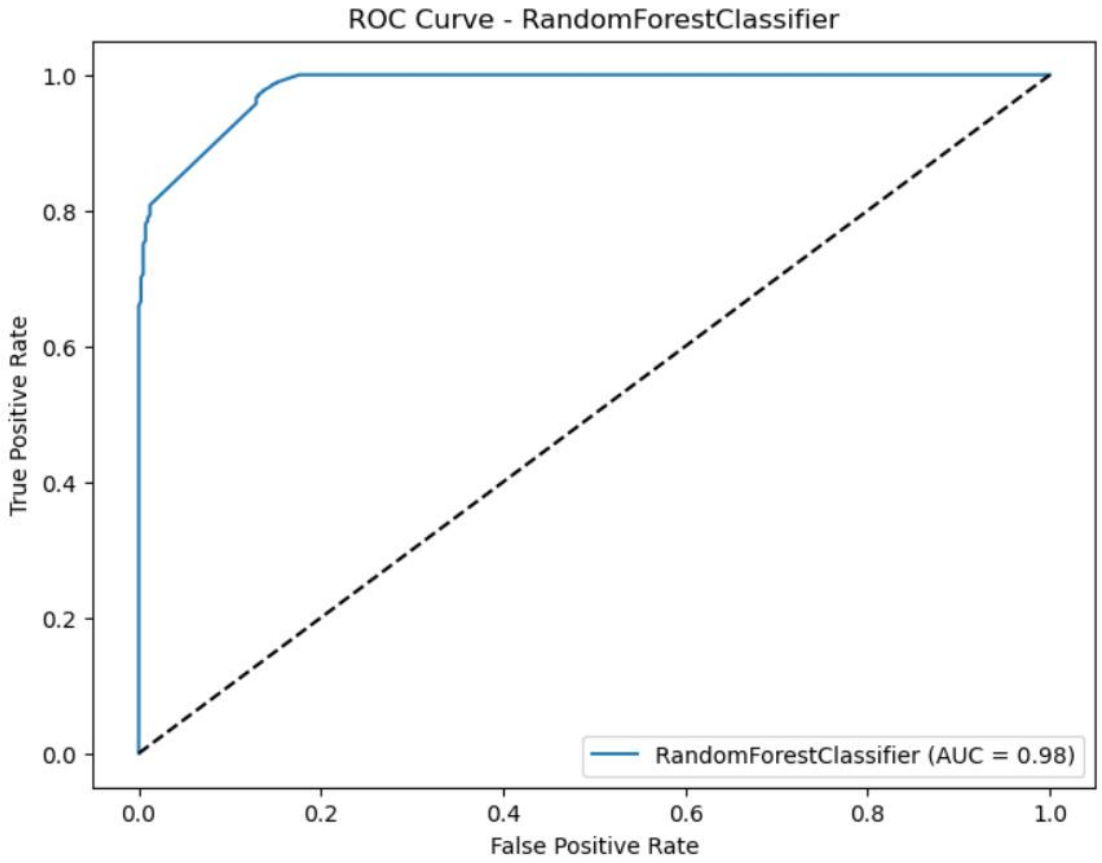
LogisticRegression()



MultinomialNB()

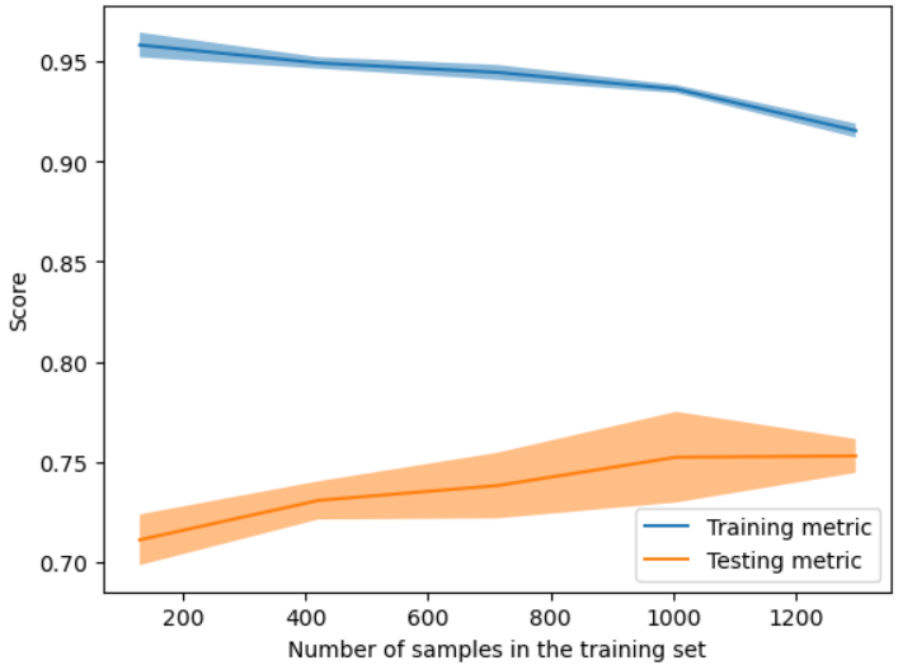


DecisionTreeClassifier()

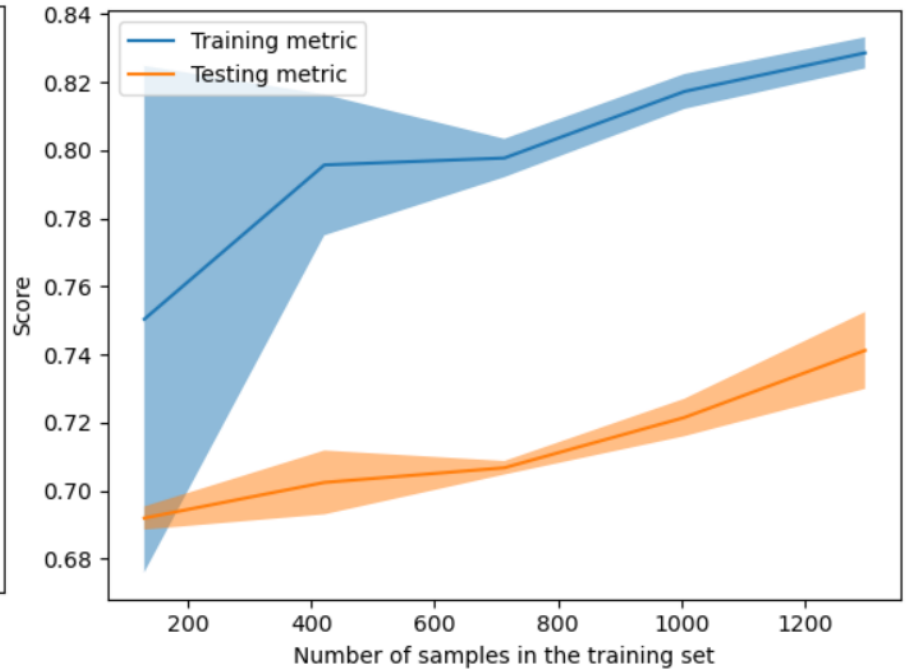


RandomForestClassifier()

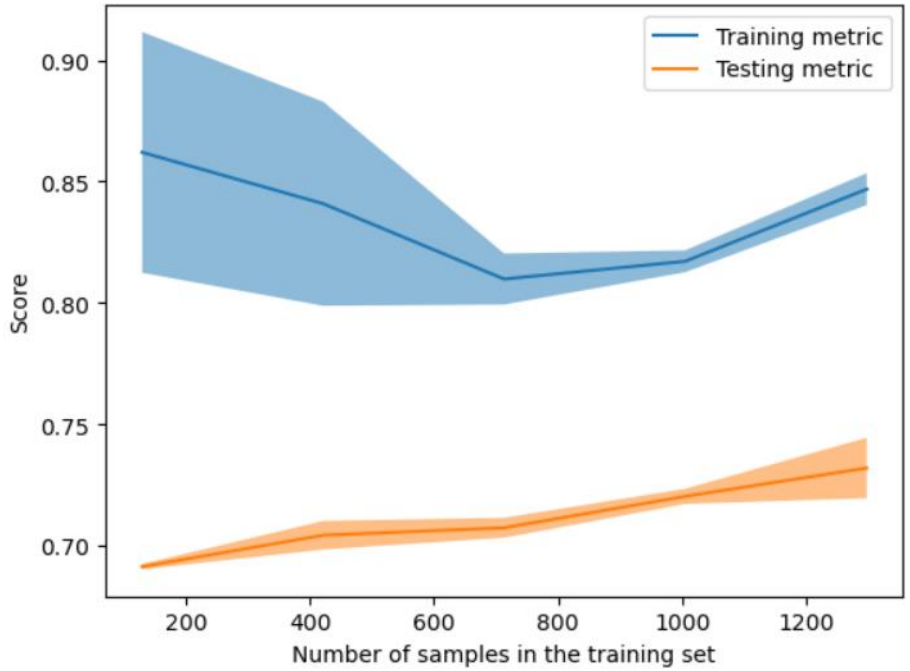
Anexo 5: Curva de aprendizaje de los modelos sin ajustar hiper parámetros:



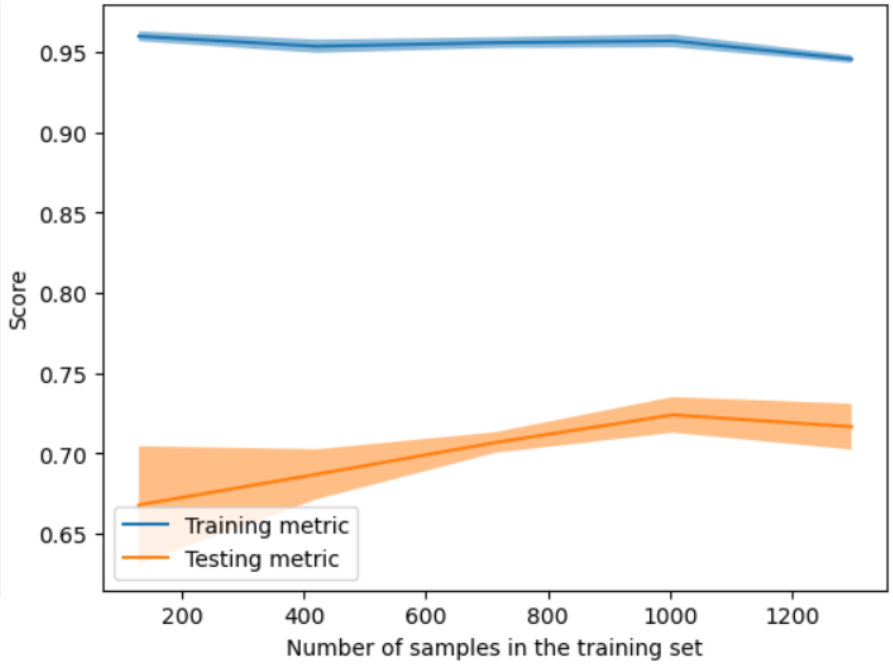
LinearSVC()



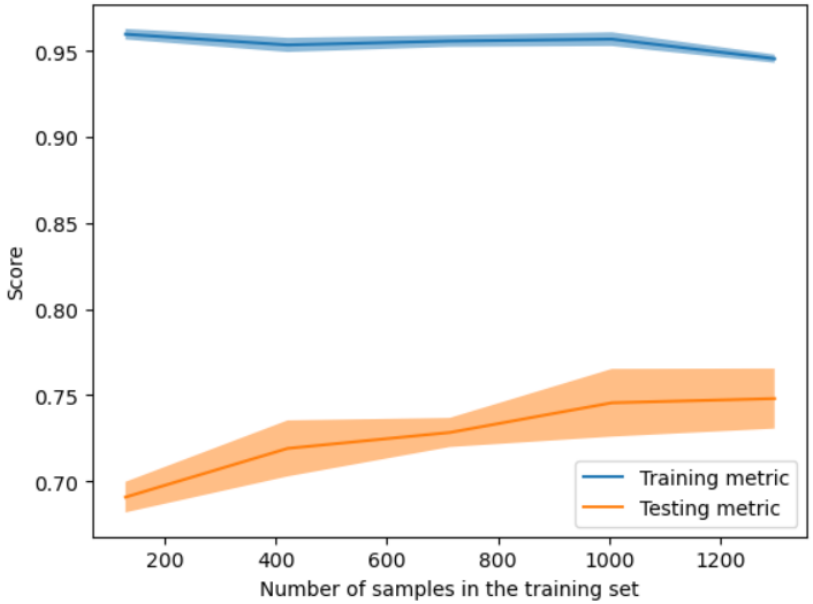
LogisticRegression()



MultinomialNB()



DecisionTreeClassifier()

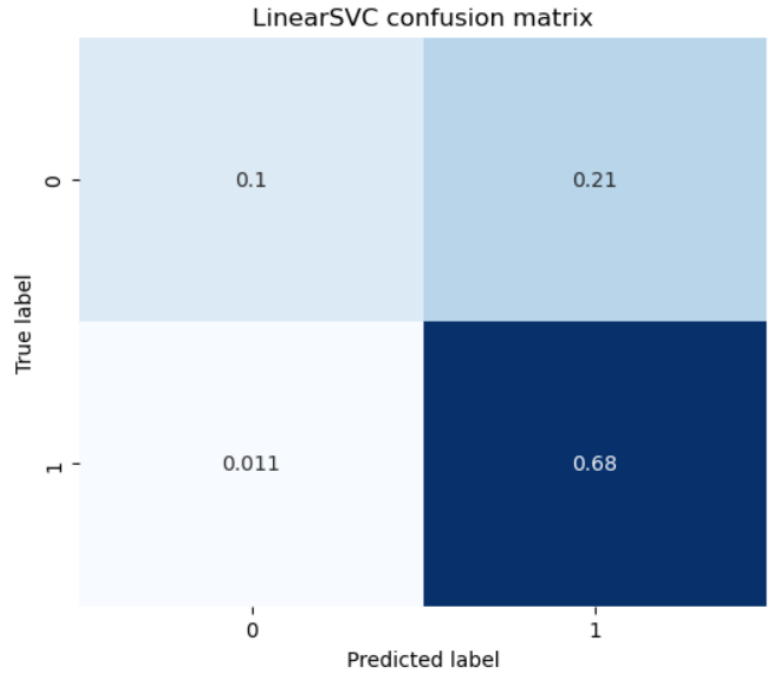


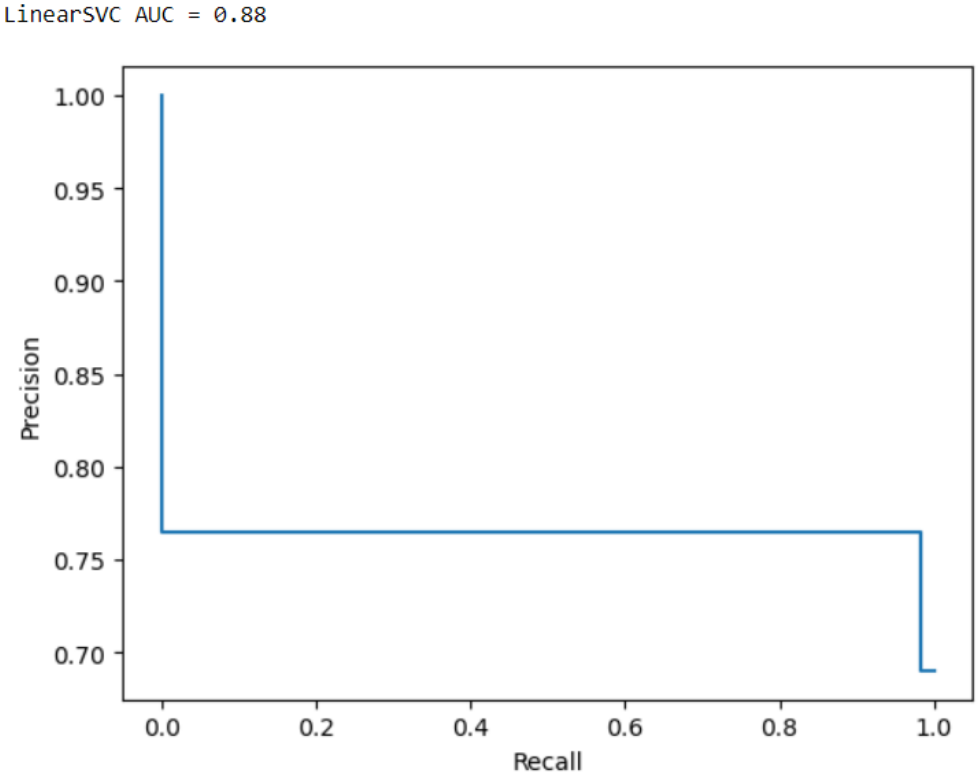
RandomForestClassifier()

Anexo 6: Precisión de los modelos con su respectivo vectorizador óptimo representada por la matriz de confusión y su curva de Precision-Recall con la puntuación AUC después de calibrar hiper parámetros:

LinearSVC()

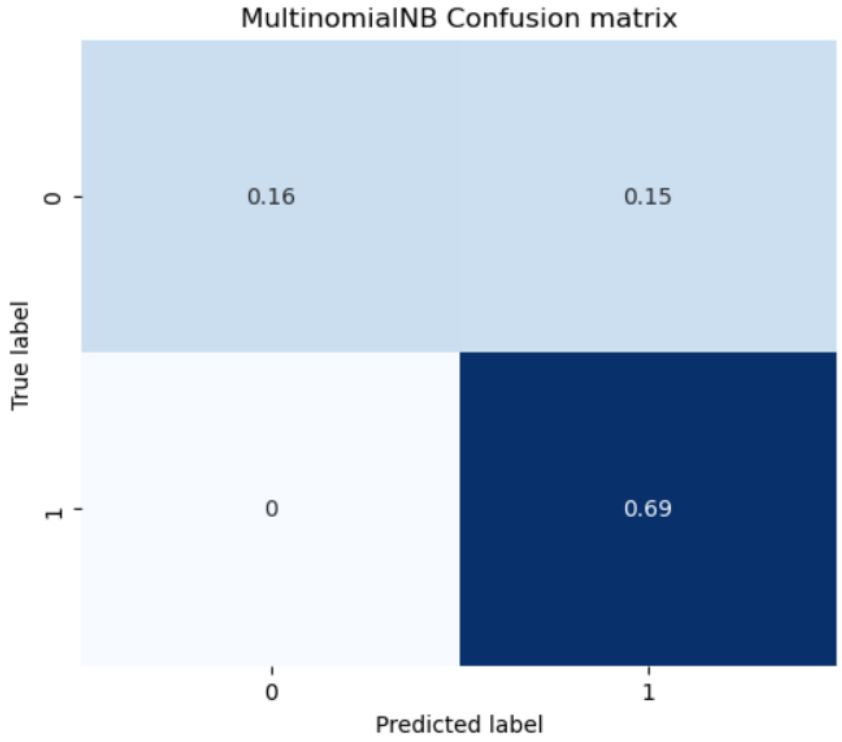
```
testing accuracy =  77.97288971041282
              precision    recall  f1-score   support

           0       0.90      0.33      0.48      2012
           1       0.76      0.98      0.86      4480

    accuracy                           0.78      6492
   macro avg       0.83      0.65      0.67      6492
weighted avg       0.81      0.78      0.74      6492
```

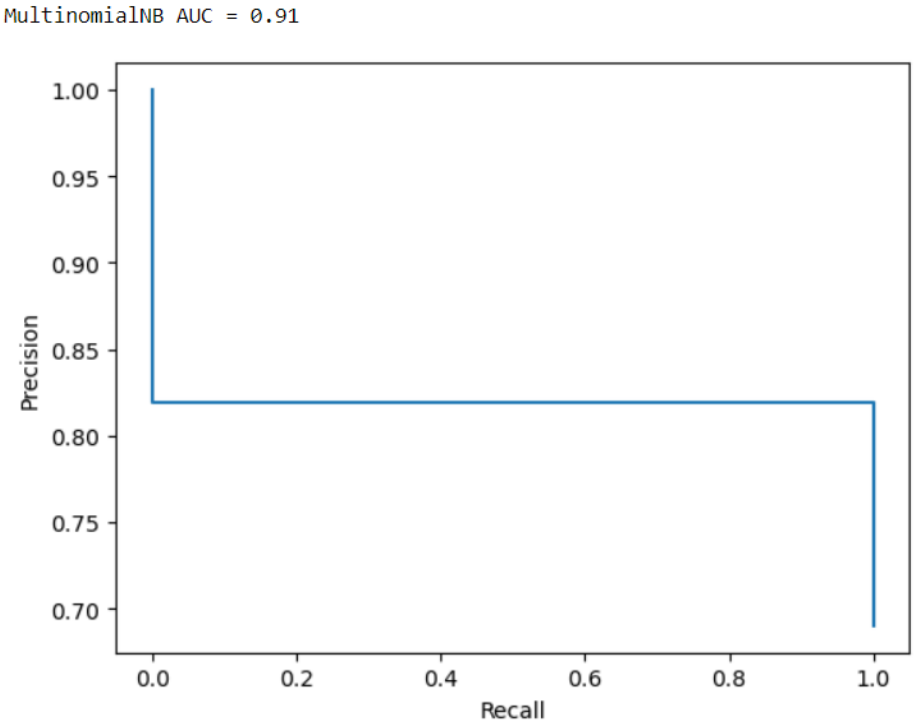




MultinomialNB()

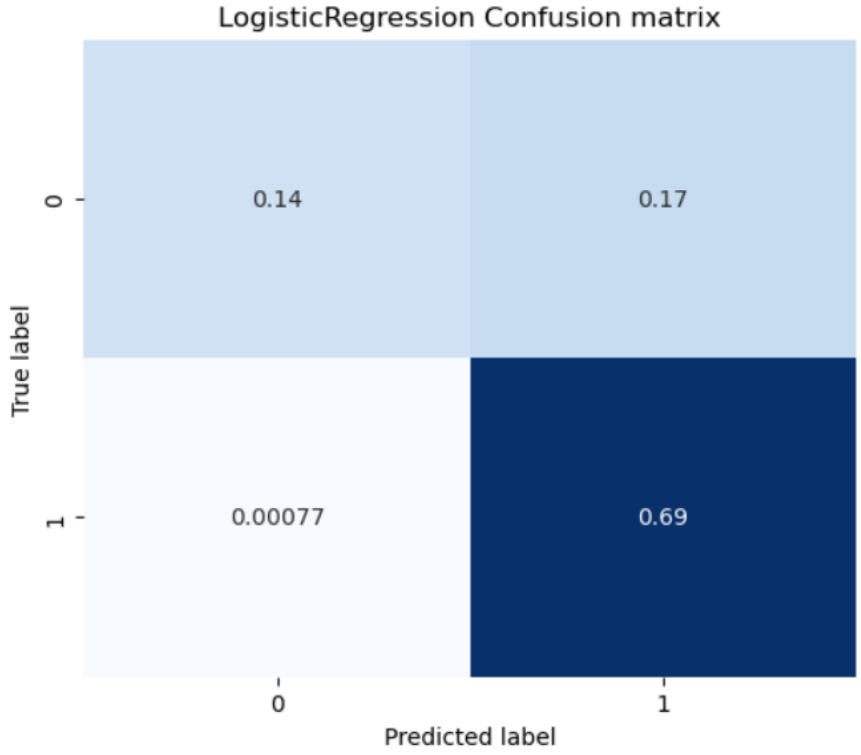
```
Training accuracy =  84.7196549599507
              precision    recall  f1-score   support

           0       1.00      0.51      0.67      2012
           1       0.82      1.00      0.90      4480

    accuracy                           0.85      6492
   macro avg       0.91      0.75      0.79      6492
weighted avg       0.87      0.85      0.83      6492
```

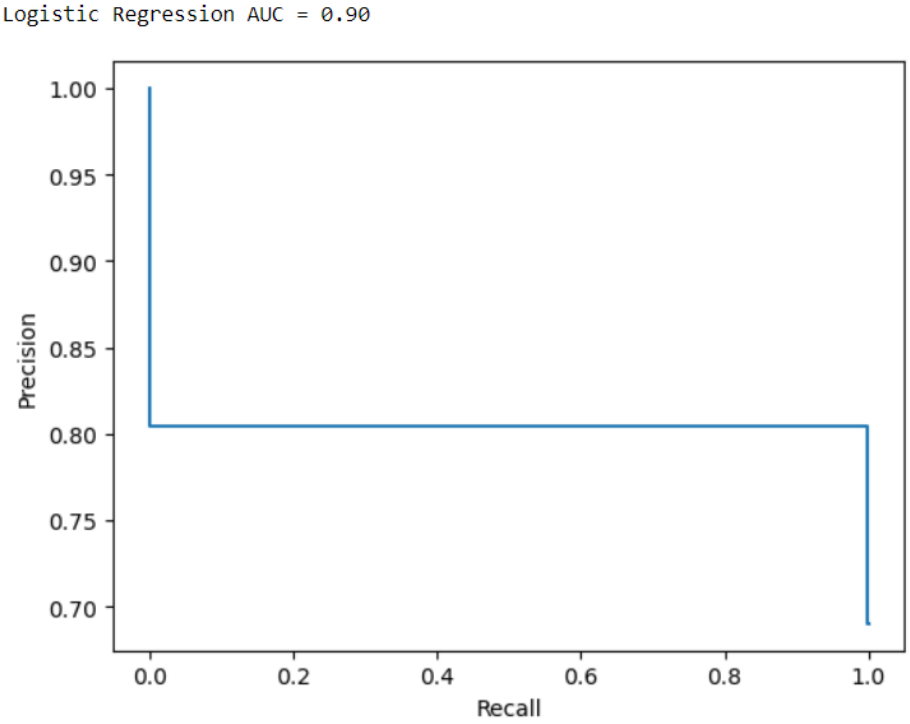




LogisticRegression()

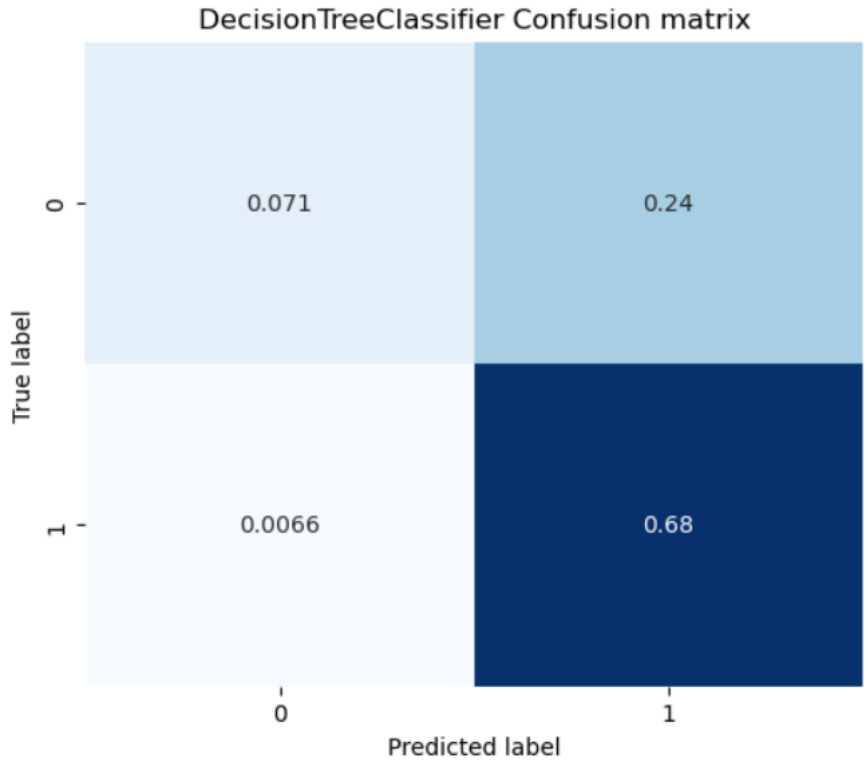
```
Training accuracy =  83.14849044978435
              precision    recall  f1-score   support

           0       0.99      0.46      0.63      2012
           1       0.80      1.00      0.89      4480

    accuracy                           0.83      6492
   macro avg       0.90      0.73      0.76      6492
weighted avg       0.86      0.83      0.81      6492
```

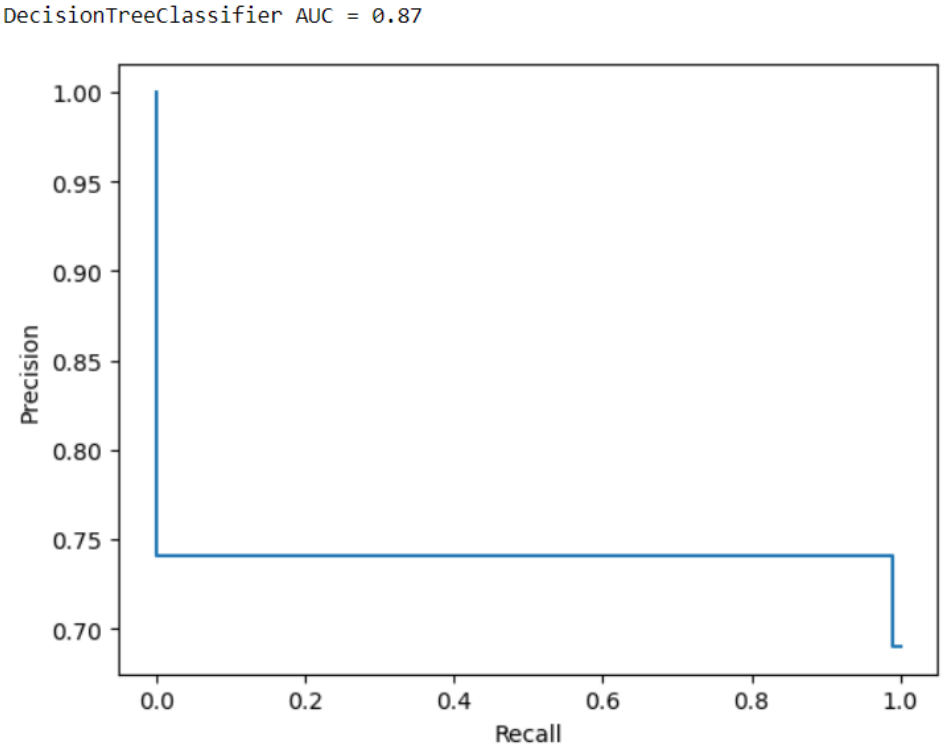




DecisionTreeClassifier()

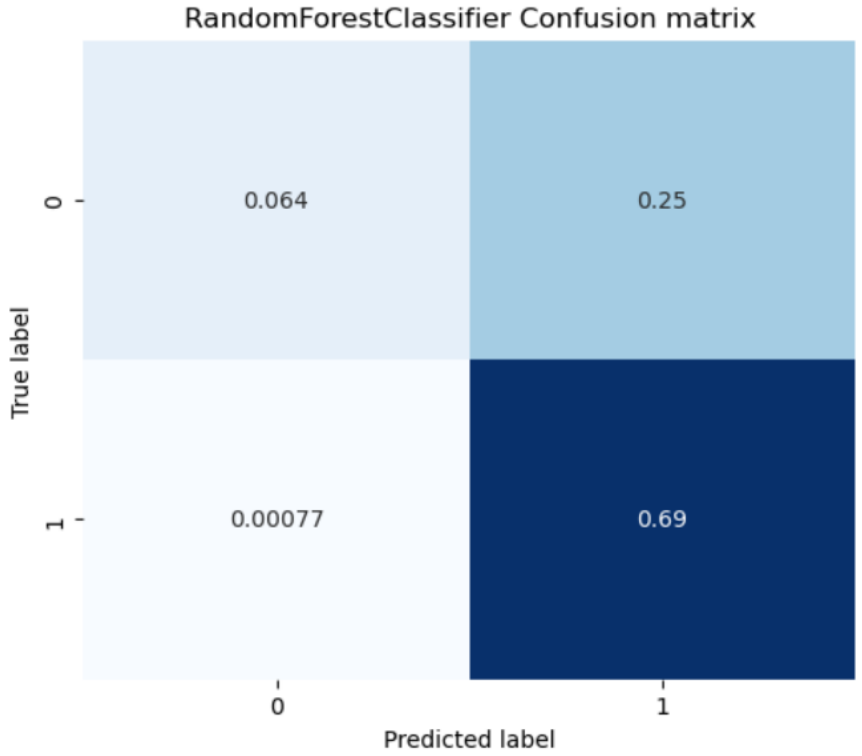
```
Training accuracy =  75.40049291435614
              precision    recall  f1-score   support

           0       0.91      0.23      0.36      2012
           1       0.74      0.99      0.85      4480

    accuracy                           0.75      6492
   macro avg       0.83      0.61      0.61      6492
weighted avg       0.79      0.75      0.70      6492
```

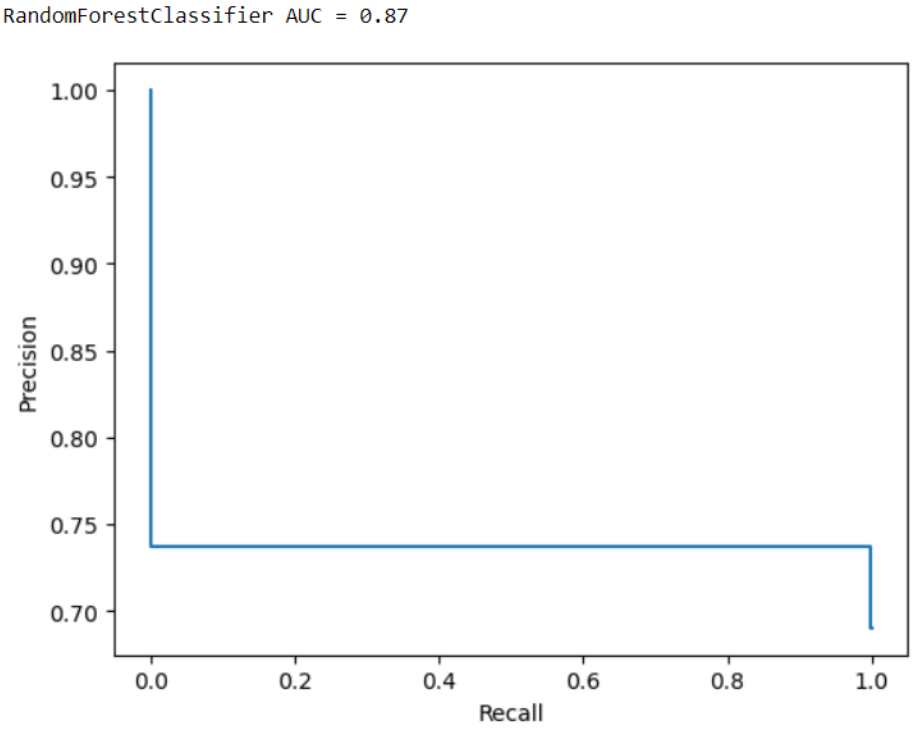




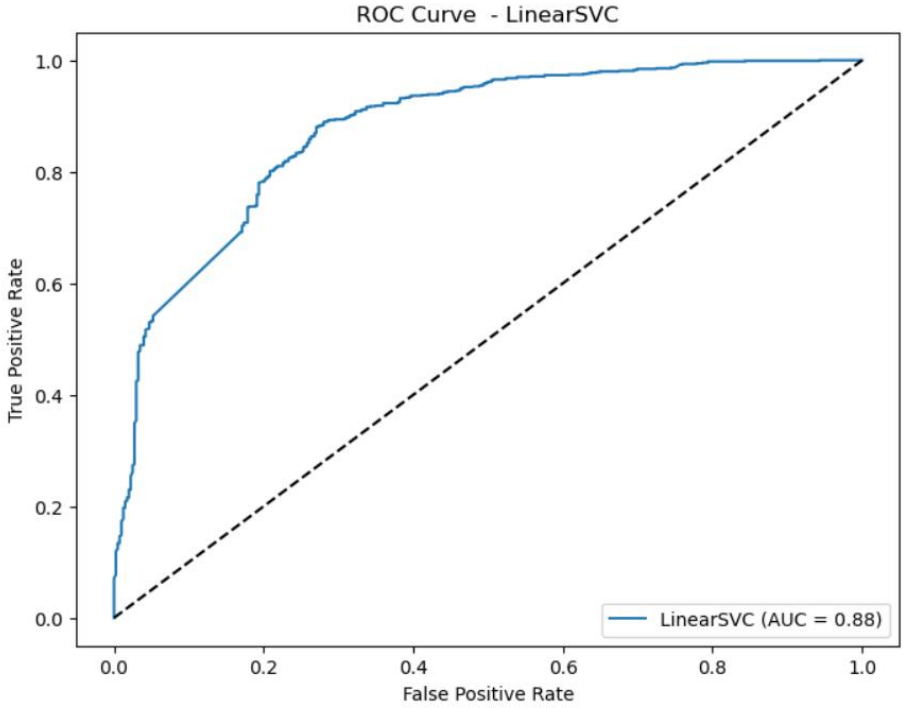
RandomForestClassifier()

```
Training accuracy =  75.29266789895256
              precision    recall  f1-score   support

           0       0.99      0.21      0.34      2012
           1       0.74      1.00      0.85      4480

    accuracy                           0.75      6492
   macro avg       0.86      0.60      0.59      6492
weighted avg       0.81      0.75      0.69      6492
```

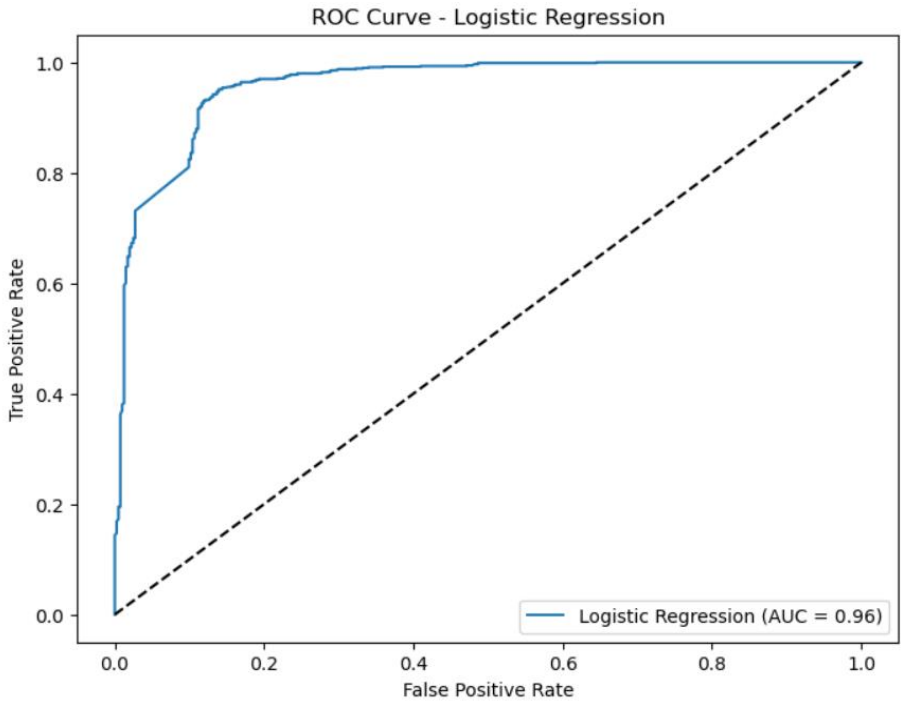




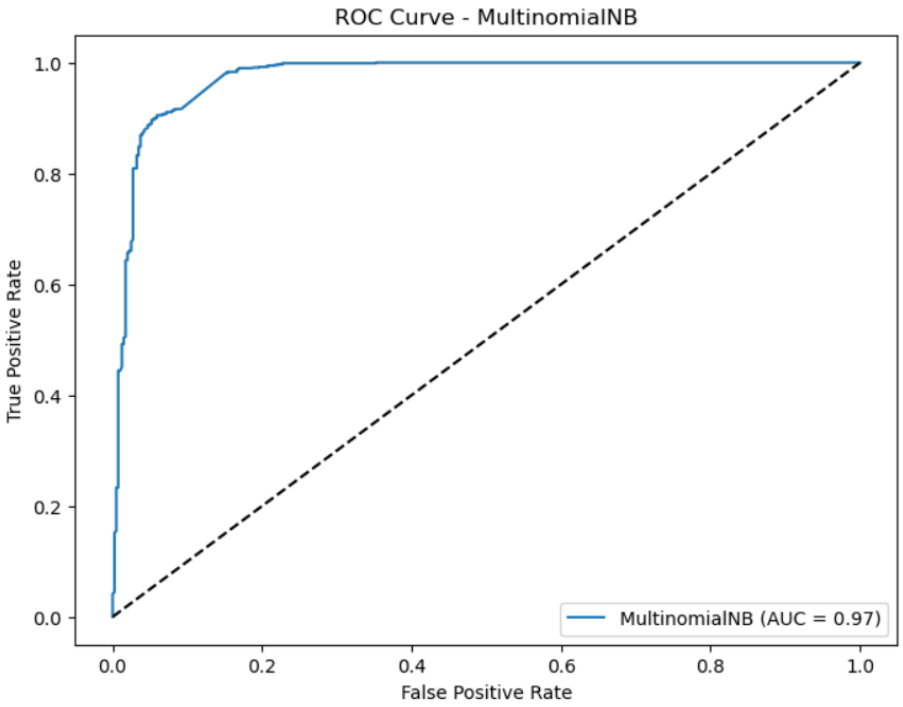
Anexo 7: Gráficas de la curva ROC de cada uno de los modelos después de calibrar hiper parámetros:



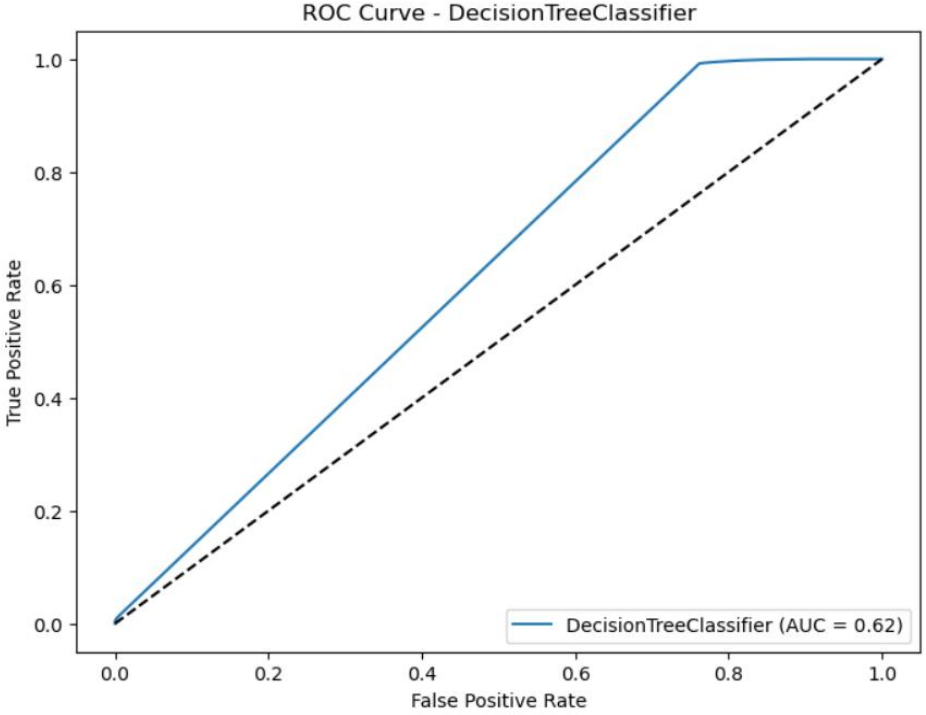
LinearSVC()



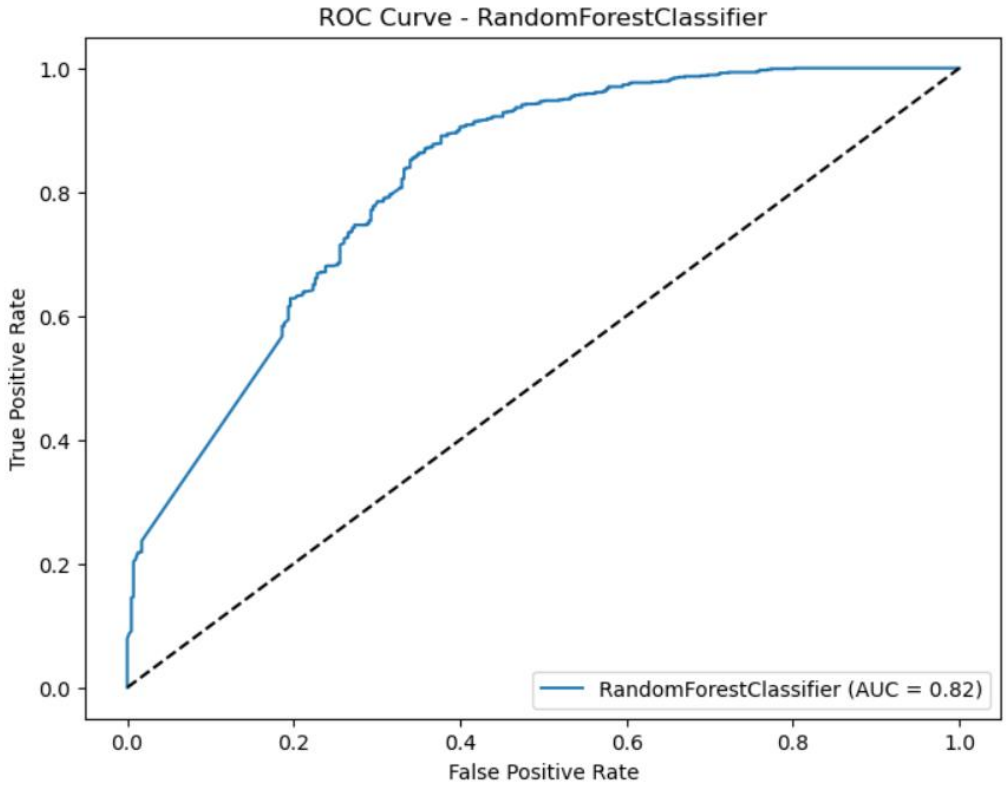
LogisticRegression()



MultinomialNB()

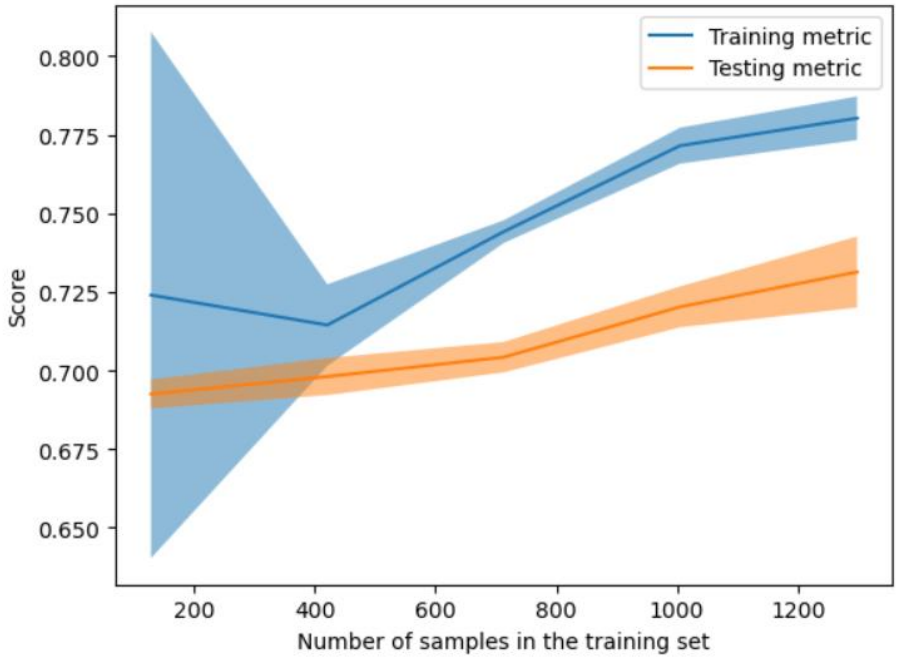


DecisionTreeClassifier()

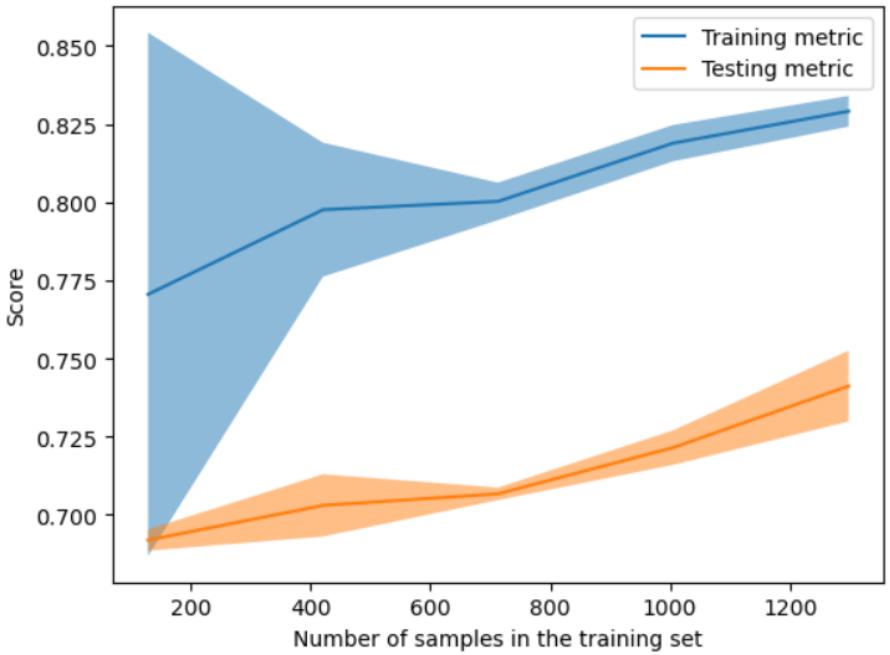


RandomForestClassifier()

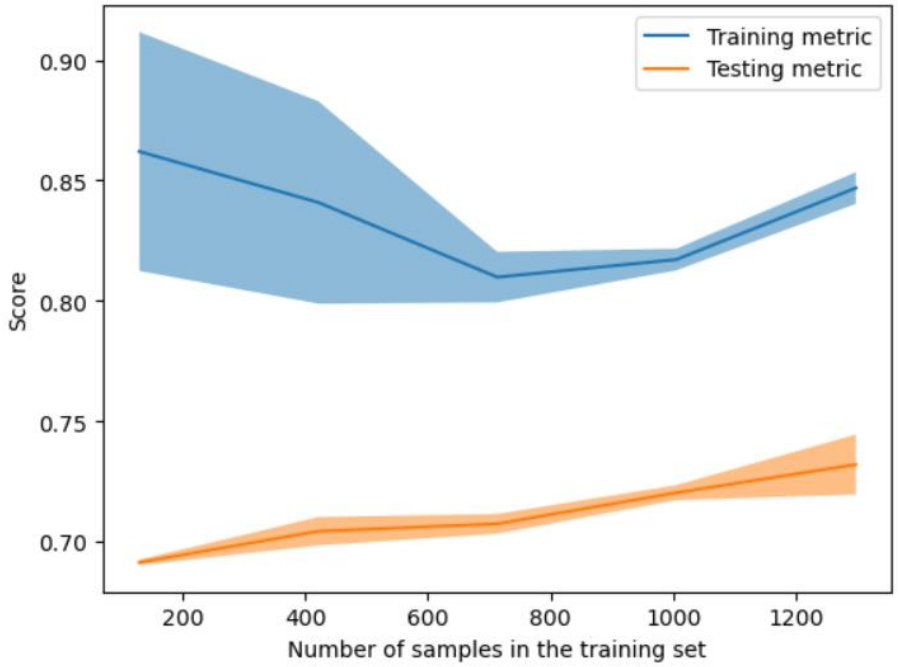
Anexo 8: Curva de aprendizaje de los modelos después de ajustar hiper parámetros:



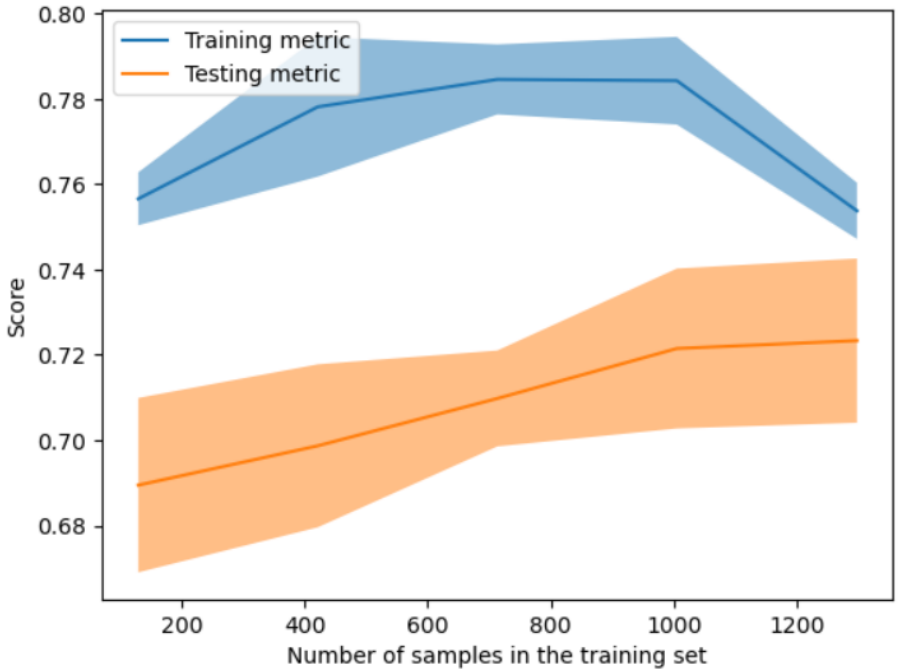
LinearSVC()



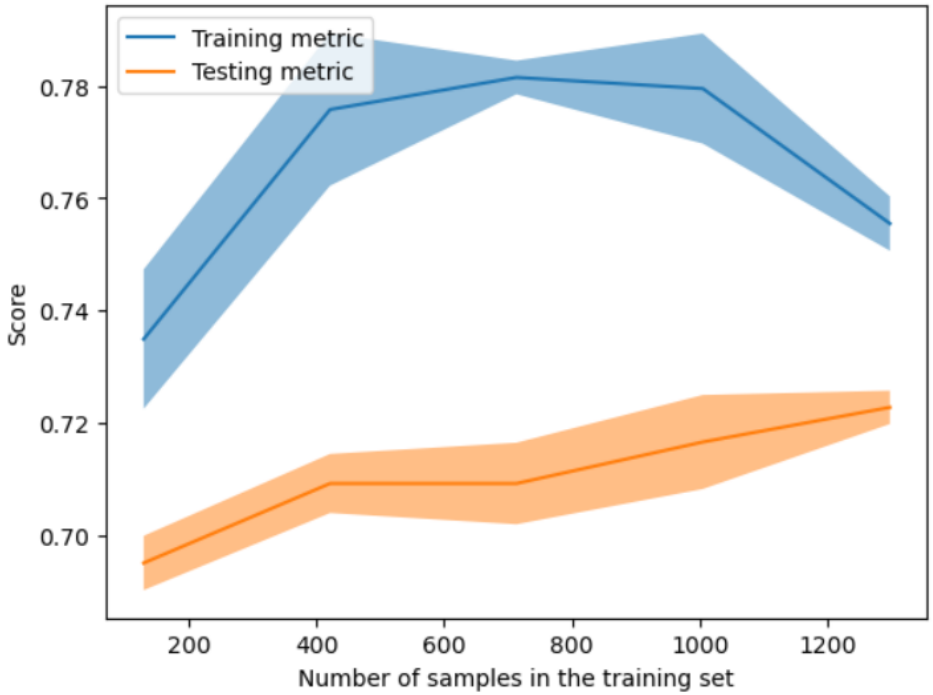
LogisticRegression()



MultinomialNB()



DecisionTreeClassifier()



RandomForestClassifier()