75.06 / 95.58

Organización de datos

Reporte TP N°2: Críticas Cinematográficas

Procesamiento del Lenguaje Natural - Análisis de sentimiento
Preprocesamiento, entrenamiento y predicciones

Alen Davies Leccese: 107084
Luca Mauricio Lazcano: 107044

Junio 2023

1 Introducción

En este trabajo, se explora otra gran área aplicación de "machine learning": el procesamiento del lenguaje natural (PLN). Ésta es una rama de la inteligencia artificial, que trata sobre el análisis, interpretación o generación de textos en "lenguaje natural", es decir, lenguaje humano. El mismo que escribimos, leemos o hablamos normalmente.

En los trabajos anteriores, se trabajó con datos numéricos y categóricos, puros y duros. Pero en este trabajo, se verá que el poder del "machine learning" tiene mucho más alcance, y permite a las computadoras leer e interpretar, e incluso generar (hablar, escribir) texto, como si fuese un ser humano, sujeto a sus sutilezas, imprecisiones, subjetividades y sesgos. Es una área de la inteligencia artificial que, si bien tiene los mismos fundamentos teóricos que el resto del "machine learning" y lleva varios años bajo estudio y aplicación, hoy en día es furor gracias a los "Large Language Models", siendo ChatGPT el más reconocido en la discusión colectiva.

El PLN tiene varias otras aplicaciones, sobre todo en "Big Data" y en las redes sociales, donde se usa para analizar y monitorear contenido de texto generado por los internautas, con diversos fines: comerciales, de moderación y censura, informativos, "trending topics", etcétera.

El alcance de este trabajo es más acotado. Dado un dataset de críticas cinematográficas, en español, catalogadas como "positiva" o "negativas", se realizará un preprocesamiento sobre estos datos, para crear varios modelos que permitan, dada otra crítica, inferir con cierta precisión si es positiva o negativa. Este análisis se denomica "análisis del sentimiento", que consiste en clasificar el texto según qué "sentimiento" transmite el autor. En este caso, haremos una clasificación binaria, pero podrían buscarse otros sentimientos, basados en otros criterios.

El dataset provisto se encuentra en dos archivos, `train.csv` y `test.csv`. El primero para realizar el entrenamiento y evaluación de los modelos. El segundo para realizar predicciones a ser entregadas por Kaggle.

Al principio del archivo Jupyter adjunto se realizan todas las importaciones y definiciones necesarias.

2 Preprocesamiento

Hay varias formas de preprocesar textos, según el contexto, idiomas y características del problema. De todos modos, algunos pasos son bastante "estándar".

2.1 Capitalización y acentuación

En primer lugar, convertimos todas las reviews a formato `string` y todas minúsculas. A efectos del análisis, consideramos una misma palabra independientemente de su capitalización. También convertimos todas las vocales con tilde a su versión alfabética. Justificamos esta decisión considerando que, en los contextos poco formales, suelen omitirse las tildes, por lo que quedarían en dos "tokens" distintos, palabras que tienen el mismo significado. Es verdad que puede llegar a perderse algo de información, al juntarse parónimos [1]. Pero, por el mismo argumento evocado anteriormente, suponemos que estos quedarán mezclados de todos modos, por lo que consideramos apropiado ignorar la acentuación de las palabras.

[1] Palabras que se escriben igual, pero con distinta acentuación, teniendo distintos significados. Por ejemplo, *revólver* y *revolver*.

2.2 Tokenización

Consiste en subdividir un arreglo de caracteres (texto), en componentes unitarios, utilizados como elemento de análisis. En este caso, buscamos tokenizar al texto en palabras. Para ello, se usa el `RegexpTokenizer` de `nltk`, con la expresión regular `'\w+'`, que captura conjuntos de caracteres, números y guiones bajos, efectivamente separando al texto en palabras.

2.3 Filtrado de stopwords

Las stopwords (palabras vacías), son palabras muy comunes en un idioma que generalmente no aportan un significado importante al análisis de texto.

Las stopwords incluyen palabras como artículos ("el", "la", "los", "las"), pronombres ("yo", "tú", "él", "ella") y preposiciones ("a", "de", "en", "con").

Las stopwords se eliminan para reducir el ruido y el tamaño del vocabulario en el texto analizado, centrándose en las palabras clave. Al eliminar estas palabras vacías, se puede mejorar el rendimiento de algoritmos de procesamiento de texto.

Para esto, usamos el conjunto de "stopwords" del idioma español e inglés, provistos por `nltk`, y los quitamos de las reviews.

2.4 Filtrado por largo de palabras

También consideramos apropiado filtrar las palabras de menos de 3 letras, ya que, en general, las palabras cortas no aportan significado, sino que suelen tener otras funciones en un texto.

2.5 Creación del vocabulario

Aquí analizamos la cantidad de palabras únicas que quedan luego del preprocesamiento, creando un vocabulario, y analizando las palabras más comunes que quedan.

Encontramos que el vocabulario consiste de de más de 160 mil palabras, gran cantidad de las cuales aparecen una sola o muy pocas veces, o son palabras con errores de tipeo. En cierto sentido, pueden considerarse "outliers". La gran mayoría del "significado" de un texto, se ouede

inferir de un conjunto más reducido de palabras. Además, entrenar modelos con demasiados tokens llevaría mucho tiempo.

Decidimos entonces limitar el tamaño del vocabulario a 10 mil palabras. Según lo investigado, parece ser un número adecuado, con un buen balance entre "performance" y velocidad de procesamiento.

Graficamos las 20 palabras más comunes para tener una idea.

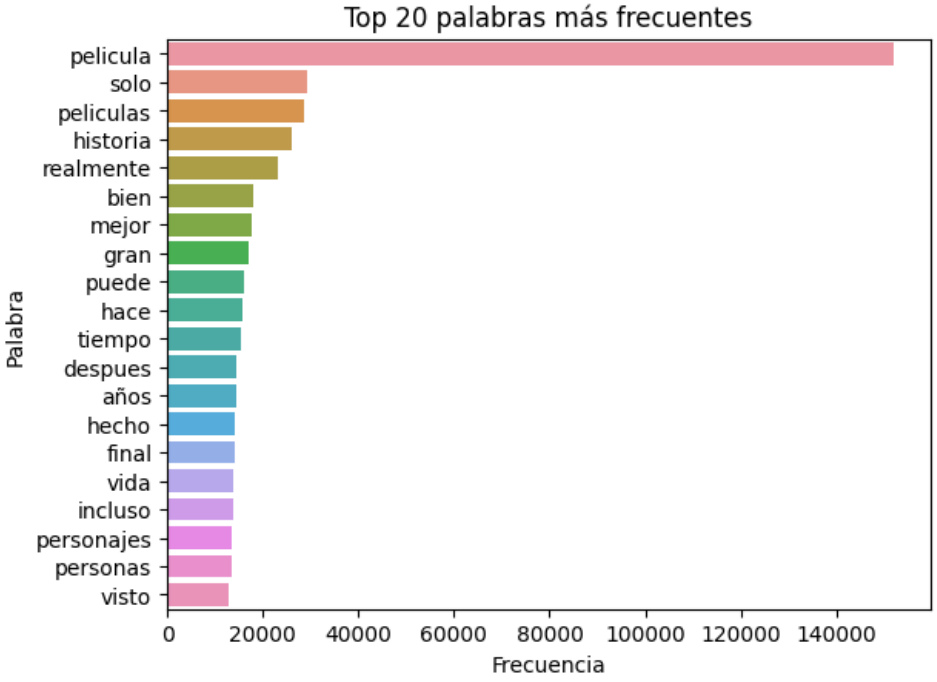

Figure 1: 20 palabras más frecuentes.

No es sorprendente encontrar "pelicula" y "peliculas" en las posiciones 1 y 3. En el caso de "solo", realizamos una búsqueda en el dataset para ver en qué contextos aparece, y por qué es tan frecuente. Encontramos que se usa principalmente como adverbio, y no aporta ninguna información. Para el resto de las palabras, se ve que tienen que ver con películas (historia, final, personajes), juicios de valor (gran, bien, mejor) o adverbios (realmente, despues, incluso).

Eliminamos entonces de las reviews las palabras que no están en el vocabulario acotado.

2.6 Truncado de las reviews

Viendo que algunas reviews tienen un largo exagerado, y considerando que en general es suficiente con las primeras oraciones para determinar el sentimiento de una crítica, acotamos el largo de estas a las primeras 300 palabras.

2.7 Bag of words

La técnica Bag of Words es una forma de representar datos de texto en aprendizaje automático, en forma de vectores. El concepto básico es tratar cada documento de texto como una "bolsa" de palabras, donde se ignora el orden y la estructura gramatical de las palabras. En lugar de eso, se enfoca únicamente en la presencia y frecuencia de las palabras en el texto.

Vamos a utilizar `TfidfVectorizer` de `scikit-learn` que implementa la técnica de ponderación TF-IDF (Term Frequency-Inverse Document Frequency) para convertir datos de texto en características numéricas.

Existen muchas otras técnicas y opciones de procesamiento, pero consideramos que el procesamiento ya realizado es suficiente.

3 Bayes Naive

Para este modelo, luego de experimentar un poco y notar que es extremandamente rápido, decidimos realizar un `RandomizedSearchCv` con cross validation, y 15 combinaciones, con 5 folds por combinación (el valor por defecto). Esto permite realizar 75 fits, que se ejecutaron en un tiempo razonable, obteniendo como mejores parámetros, los siguientes:

```
Mejores parámetros: {'force_alpha': True,
'fit_prior': True, 'class_prior': None, 'alpha': 0.5}
Mejor métrica: 0.8432260543879829
```

Entrenamos el modelo de Bayes Naive con los mejores hiperparámetros y realizamos las predicciones.

Las métricas de los resultados obtenidos son:

```
Accuracy: 0.8439333333333333
Recall: 0.8492555013835815
Precision: 0.8433656110965716
f1 score: 0.8463003085811831
```

La matriz de confusión obtenida:

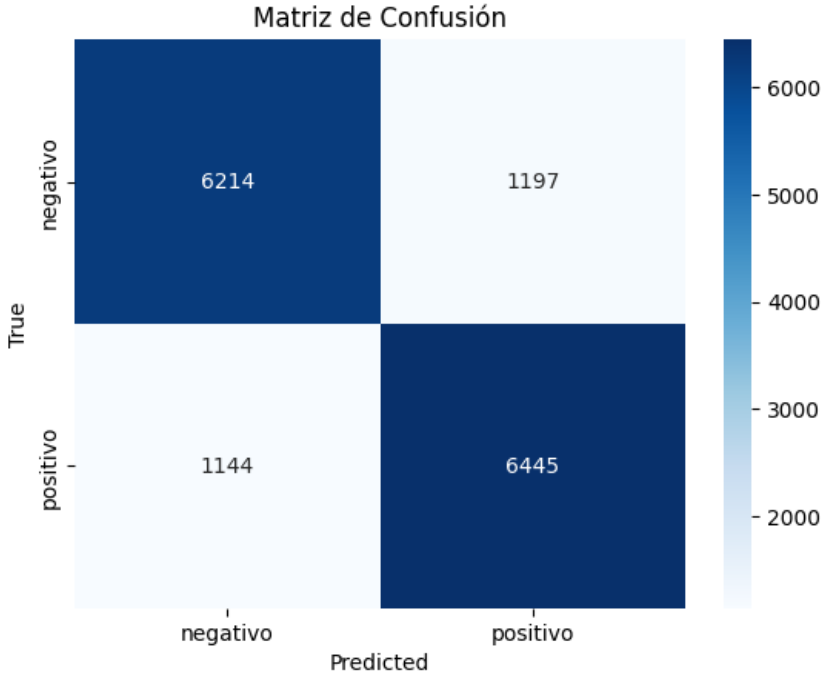

Figure 2: Matriz de confusión Bayes Naive.

Con este modelo, el resultado obtenido en Kaggle fue de **0,715**.

4 Random Forest

En el caso de este modelo, que, por cierto, había dado excelentes resultados en el trabajo práctico anterior, los resultados no fueron tan satisfactorios. Debido al extremadamente largo tiempo que tardaba el `RandomizedSearchCV`, lo ejecutamos con 5 combinaciones, y 2 folds por combinación, dando un total de 10 fits. Se obtuvieron los siguientes hiperparámetros, aunque creemos que, dado suficiente poder de cómputo, estos pueden mejorarse.

```
Mejores parámetros: {'n_estimators': 100, 'min_samples_split': 5,
'max_samples': 0.99, 'max_depth': None, 'ccp_alpha': 0.01}
Mejor métrica: 0.7241887891698224
```

Las métricas de los resultados obtenidos son:

```
Accuracy: 0.7028666666666666
Recall: 0.8115693767294768
Precision: 0.670476812540823
f1 score: 0.7343070044709389
```

La matriz de confusión obtenida:

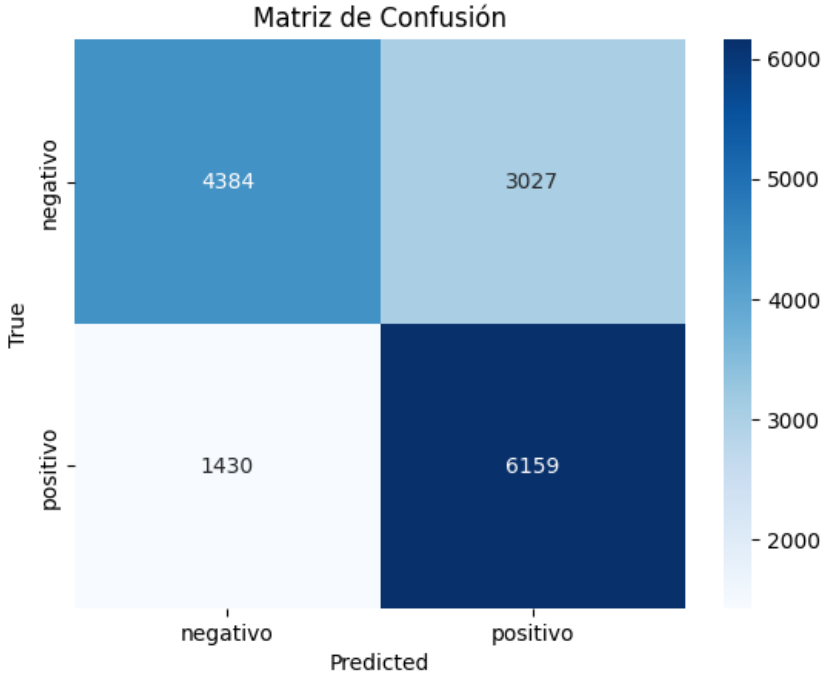

Figure 3: Matriz de confusión Random Forest.

Con este modelo, el resultado obtenido en Kaggle fue de **0,612**.

5 XGBoost

Para este modelo, el panorama fue similar al de Random Forest. Se tardó mucho tiempo en correr el `RandomizedSearchCV`, aunque esta vez se dejaron los folds del cross validation en su valor por defecto (5), probando entonces un total de 25 fits. Los hiperparámetros obtenidos son:

```
Mejores parámetros: {'subsample': 0.42105263157894735, 'n_estimators':
100, 'max_depth': 2, 'learning_rate': 0.4448979591836735, 'lambda': 0,
'gamma': 0, 'alpha': 1}
Mejor métrica: 0.8161230789803892
```

Las métricas de los resultados obtenidos son:

```
Accuracy: 0.8163333333333334
Recall: 0.8517591250494136
Precision: 0.7986162589572523
f1 score: 0.8243320793215584
```

La matriz de confusión obtenida:

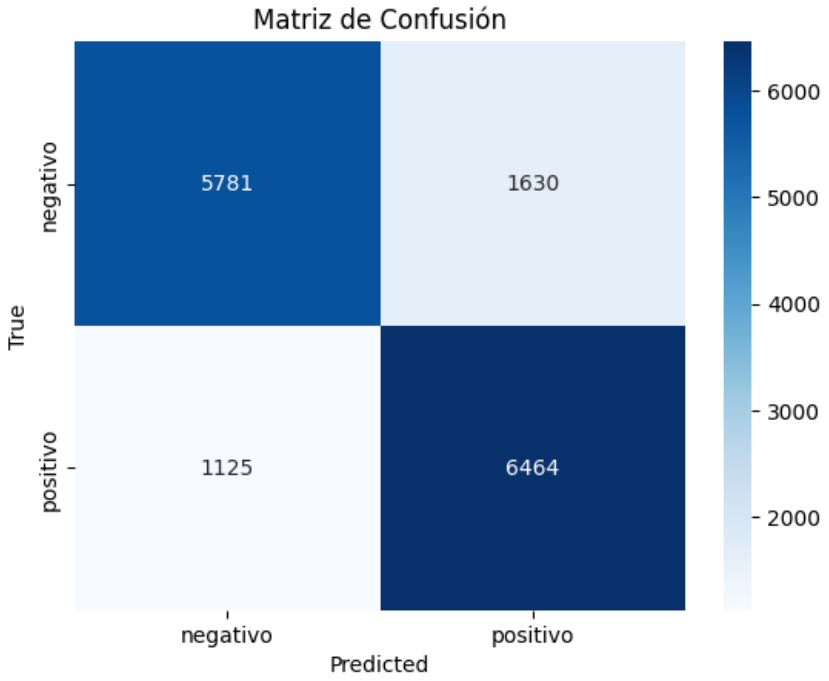

Figure 4: Matriz de confusión XGBoost.

Con este modelo, el resultado obtenido en Kaggle fue de **0,694**.

6 Red Neuronal

En el caso de la red neuronal, nos encontramos con diversas dificultades a la hora de crear y compilar el modelo, sobre todo debido al formato de los datasets, que no eran compatibles con los "tensores", usados por `Tensor Flow`.

Para remediar esto, se cambió el método de tokenizado, usando `Tokenizer` de `Keras`. En el notebook se presenta en más profundidad el procesamiento realizado y más explicación.

Otro asunto a considerar, fue el hecho de que la red neuronal necesita un tensor "denso" (dimensión de entrada fija), y las reviews, luego de la codificación, forman uno "ragged", es decir, con longitudes variables. Fue necesario entonces rellenar con tokens que no aporten información, para convertir el tensor al formato necesario.

Respecto a la arquitectura del modelo, consiste en 4 capas. La capa de entrada es del

tipo `Embedding`. La "técnica" de Embedding consiste en convertir el conjunto de vectores a un espacio dimensional menor, pero capturando las semejanzas y diferencias semánticas entre palabras. Puede verse como una reducción de la dimensionalidad, de las 10000 palabras del vocabulario de entrada, a las 128 (embed size), que representan los "parámetros" con los cuales el modelo caracteriza cada palabra. Más información se brinda en el notebook adjunto.

Las capas intermedios consisten de capas `GRU`, estas son variantes simplificadas de `LSTM`, que permiten "recordar" información a largo plazo, evitando perder el aprendizaje obtenido en una parte anterior del conjunto de información.

La capa de salida es del tipo `Dense`, con activación `'sigmoid'`, lo cual permite mapear el valor predicho entre 0 y 1.

El resumen de la arquitectura del modelo es:

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, None, 128)         1305600

 gru_32 (GRU)                (None, None, 16)          7008

 gru_33 (GRU)                (None, 8)                 624

 dense_16 (Dense)            (None, 1)                 9

=================================================================
Total params: 1,313,241
Trainable params: 1,313,241
Non-trainable params: 0
_________________________________________________________________
```

Compilamos el modelo con pérdida `'binary_crossentropy'`, optimizador `'adam'` y buscando favorece la métrica `'accuracy'`.

Además, agregamos un callback `EarlyStopping` para prevenir el overfitting.

Se fitea el modelo, lo cual llevó varias horas.

Los scores obtenidos son:

```
Training Accuracy: 0.9827428460121155
Testing Accuracy: 0.5085999965667725
```

Esto resulta decepcionante. Claramente, el modelo creado sufre de muchísimo overfitting. De hecho, el modelo creado como "prueba" anteriormente, con un vocabulario de 100 palabras, reviews limitadas a sus primeras 20 palabras, apenas 16 parámetros de salida de la capa `Embedding`, resultando en una arquitectura con menos de 20 mil parámetros entrenables, resultó mucho mejor que el modelo aquí presentado; con sus 1,3 millones de parámetros.

Aún así, presentamos las métricas obtenidas:

```
Accuracy: 0.5086
Recall: 0.5049413624983529
Precision: 0.5146387322052108
f1 score: 0.5097439308280678
```

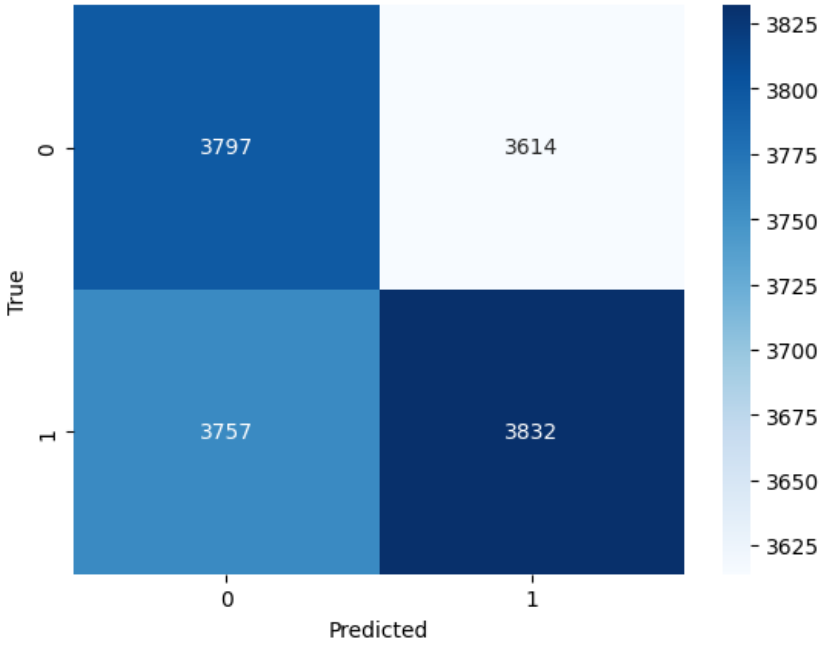

Figure 5: Matriz de confusión Red Neuronal.

El resultado obtenido en Kaggle fue de **0,491**.

El resultado con el modelo "de prueba" en Kaggle fue de **0,626**.

7 Ensamble

Para el caso del modelo de ensamble, se optó por el modelo de votación, o "Voting", que había dado buenos resultados en el trabajo práctico anterior. Elegimos los modelos `Random Forest`, `XGBoost` y `Bayes Naive`.

Primero, realizamos un resumen de los scores obtenidos con dichos modelos:

```
random_forest 0.718
xgboost 0.819
bayes_naive 0.844
```

Y del clasificador:

```
Voting Classifier: 0.8317106540450812
```

El resultado de las predicciones realizadas es:

```
Accuracy: 0.8304666666666667
Recall: 0.8811437607062854
Precision: 0.8029538904899135
f1 score: 0.8402337123829867
```

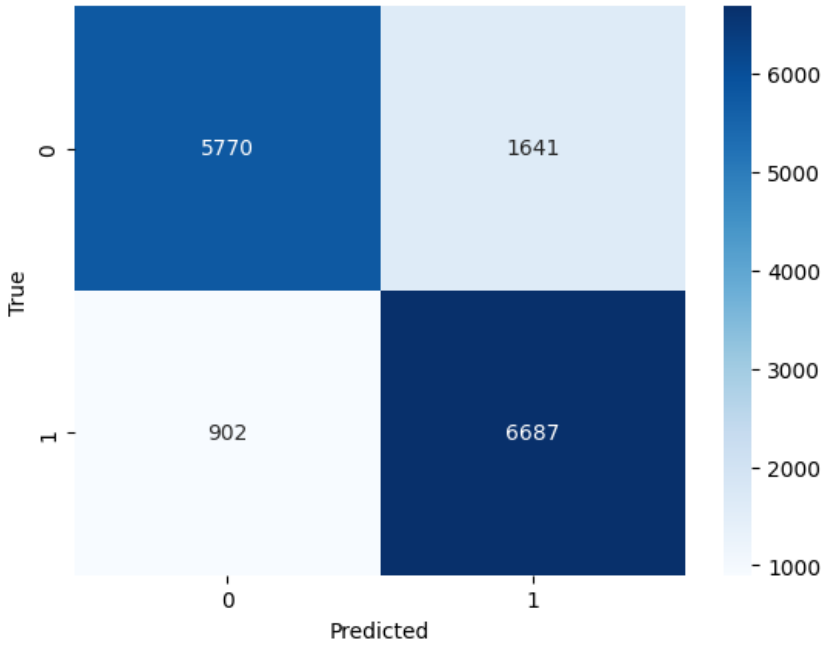

Figure 6: Matriz de confusión Ensamble.

Presentamos una comparación de la performance de los modelos, junto a la del clasificador:

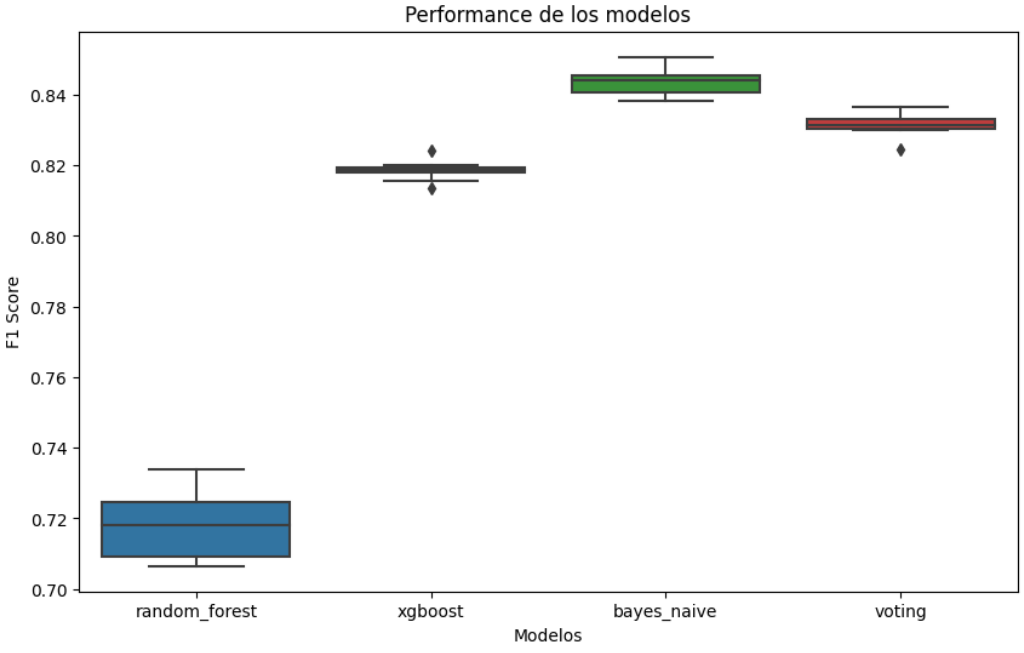

Figure 7: Scoring ensamble y modelos individuales.

El resultado obtenido en Kaggle fue de **0,709**.

8 Conclusión

En conclusión, el procesamiento del lenguaje natural es una disciplina en constante evolución que ha logrado avances significativos en la interacción entre las máquinas y el lenguaje humano. A través de algoritmos y técnicas de inteligencia artificial, se ha logrado desarrollar sistemas capaces de comprender, interpretar y generar texto o habla de manera más natural y precisa. A medida que el procesamiento del lenguaje natural continúa progresando, se espera que tenga un impacto cada vez mayor en áreas como la traducción automática, la atención al cliente, la búsqueda en línea y muchas otras aplicaciones prácticas.

El párrafo anterior fue redactado por ChatGPT, e ilustra el impacto y el potencial del área de la inteligencia artificial, y del PLN en específico, hoy en día, y a futuro. Si bien herramientas como ChatGPT son mucho más sofisticadas que lo explorado a lo largo de este trabajo práctico, y alo largo de la materia, es fascinante familiarizarse con los principios que hacen todo eso posible.

Respecto a la implementación del trabajo en particular, quizás podríamos haber obtenido mejores resultados teniendo a disposición mucha más capacidad de cómputo, permitiendo explorar más hiperparámetros en los modelos, y así eventualmente obtener mejores resultados.

Respecto al preprocesamiento, existen muchas más formas de preprocesar los textos, lo cual podría haber resultado en mejor entrenamiento, y por ende, en modelos con mayor poder de predicción que los logrados.