Presentación Final de Inteligencia Artificial: Detección de Fake News y Sesgo

Nancy Cruz, J. Luis Cortes, Pedro Llerenas, J. Antonio Zarco

Universidad de Monterrey

05 de Abril de 2025

Abstract

Presentamos 2 modelos que se complementan para detectar la veracidad de diversas noticias. Se utilizaron 2 conjuntos de datos para entrenar a cada uno de los modelos, uno está encargado de verificar la veracidad de la noticia. El segundo modelo es un modelo de regresión que nos indica el sesgo político que se puede ver en la noticia, complementando el funcionamiento del modelo anterior y proporciona un mejor panorama para discernir la veracidad e intereses de la información que consumimos.

Keywords: Inteligencia artificial, Sesgo, Clasificación, Agrupamiento, Veracidad, Noticias,

Detección de Fake News y Sesgo

Este documento tiene el fin de presentar el proyecto elaborado durante el curso de *inteligencia artificial.* Mostraremos las diferentes etapas por las cuales pasamos para el desarrollo de estos modelos. Primero pasamos por el proceso de selección de un dataset que nos fuera útil y con el cual fuera sencillo trabajar, posteriormente se buscó otro dataset con la finalidad de complementar el uso de los modelos y dar una respuesta más precisa. Una vez seleccionados los datasets se hizo el entrenamiento de los modelos, en particular el detector de noticias faltas, y otro aparte con la capacidad de detectar la inclinación política del autor.

Índice

Introducción

La inteligencia artificial es un campo de la ciencia relacionado con la creación de computadoras y máquinas que pueden razonar, aprender y actuar de una manera que normalmente requeriría inteligencia humana o que involucra datos cuya escala excede lo que los humanos pueden analizar. [1]

En años recientes, las redes sociales se han vuelto masivas, donde el flujo de información se ha convertido en algo inhumanamente posible de procesar y analizar. Específicamente, la cantidad de noticias y reportes acerca de nuestros alrededores nos llega de manera casi instantánea, con un nivel de propagación inmensa. Hacer una verificación de veracidad para tanta información no es posible (o ineficiente) para un consumir. La inteligencia artificial resulta útil en este problema, ya que nos puede ayudar a discernir entre noticias falsas o noticias con cierto tipo de sesgo político. Esta habilidad permitirá a los lectores realizar decisiones informadas.

Problema a Atacar

Consideramos que es un gran problema la distribución de noticias falsas en redes sociales, ya que mucha información es compartida por estos medios, esto trae consigo varias consecuencias, como el hecho de que se puede compartir grandes volúmenes de información en poco tiempo, lo cual provoca que las Fake News se distribuyan considerablemente, ya que es técnicamente imposible verificar la veracidad de todo el contenido que consumimos, por lo cual hemos desarrollado un modelo de inteligencia artificial el cual nos permite detectar noticias falsas y también conocer el sesgo político que pueden llegar a tener estas noticias.

Objetivo

Reducir el efecto de las noticias falsas y segadas en la comunidad para promover las decisiones informadas.

Conjunto de Datos

Se utilizaron 2 datasets para el entrenamiento de los 3 modelos.

Fake News Detection Datasets (FNDD)

Es un conjunto de noticias recolectadas de medios reales de alrededor del mundo. Se conforma por dos archivos CSV: *Fake.csv* y *True.csv*.

Como lo indican sus nombres, Fake.csv contiene noticias categorizadas como falsas y True.csv contiene noticias categorizadas como verdaderas. Las noticias falsas fueron recolectadas de distintos medios, cuyos dominios fueron categorizados como *no factibles* por PolitiFact, mientras que las verdaderas fueron recabadas mayormente de Reuters. Cabe destacar que toda la información encontrada en estos archivos está en inglés. Estos archivos se encuentran estructurados de la siguiente manera:

title	**text**	**subject**	**date**
Título de la noticia	*Contenido de la noticia*	*Tópico de la noticia*	*Fecha de publicación de la noticia*

Inicialmente, no tenemos una columna de label, ya que la separación se hizo mediante la creación de archivos distintos. La siguiente tabla muestra la distribución de noticias:

Noticia	**Conteo**
Verdadera	21417
Falsa	23481

BABE - Media Bias Dataset: Annotation by Experts (BABE)

Es un conjunto de noticias recolectadas de diversos medios. Nos enfocamos precisamente en 2 bases de datos de este dataset: final_labels_sg1.xlsx y final_labels_sg2.xlsx. Los datos tienen la siguiente estructura:

text	news_link	outlet	topic	type
Contenido de la noticia	Link al articulo	Medio que publicó la noticia	Tópico	Inclinación política del autor

label_bias	label_opinion	biased_words
Indicador de sesgo	Indicador de sentimiento del texto, ya sea con opinión o puramente factual	Lista de palabras sesgadas

Como ha sido explicado en Spinde2021f, estos datasets han sido etiquetados por expertos, que asimismo superan en calidad a su previo dataset etiquetado por el público abierto.

En resumen, BABE consiste de 3,700 oraciones con anotaciones hechas por expertos. Notemos también que el primer archivo es un subconjunto del segundo, con 1000 registros menos. Sin embargo, notamos que estos 1000 registros nuevos no están completos; es decir, todos los valores de *type* se encuentran nulos. Por lo tanto, tendremos que rellenarlos con otro modelo de IA.

Análisis Exploratorio de Datos

Fake News Detection Datasets (FNDD)

Se hizo el análisis exploratorio de datos analizando la distribución del tipo de noticias según vienen clasificadas en *FNDD graficando la cantidad de noticias para cada uno de los tópicos marcados en el dataset*

Se hizo el análisis del número promedio de palabras en cada noticia, análisis por día, mes y año; se buscaron noticias con palabras clave.

Se analizó la frecuencia de palabras en las noticias falsas y verdaderas, así como la correlación entre los tópicos, fechas y palabras más comunes presentes en el dataset

BABE - Media Bias Dataset: Annotation by Experts (BABE)

Para este segundo dataset, utilizado para el modelo de regresión que predice el nivel de sesgo de una noticia, primero realizamos una revisión rutinaria de columnas, sus tipos de datos, y posibles valores nulos. Hemos quitado la columna de *news_link*, ya que esta no aporta información adicional para el modelo.

#	Column	Non-Null Count	Dtype
0	text	3674	object
1	outlet	3674	object
2	topic	3674	object
3	type	2674	object
4	label_bias	3674	object
5	label_opinion	3674	object
6	biased_words	3674	object

Notemos que *type* tiene 1000 registros nulos. Estos son precisamente los registros nuevos comparados a los del archivo de la versión 1. Para rellenarlos, utilizamos un modelo de clasificación. A continuación, presentamos la idea a alto nivel:

```
df_null_type = df[df['type'].isnull()]
df.dropna(subset=['type'], inplace=True)
X = df.drop(columns=['type'])
```

```
y = df['type']

categorical_cols = ['outlet', 'topic', 'label_bias', 'label_opinion']
text_cols = ['text', 'biased_words']

# Preprocessing phase for the data, encoder and tfidvectorizer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_cols),
        ('text1', TfidfVectorizer(max_features=5000), 'text'),
        ('text2', TfidfVectorizer(max_features=500), 'biased_words')
    ]
)

# Model pipeline for RandomForestClassifier
model = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

Una vez hecho esto, generamos un nuevo archivo CSV con toda la información necesaria para comenzar a realizar el análisis del dataset para el modelo de regresión.

Preparación de columnas

Primero observemos la columna *label_bias*.

```
df['label_bias'].value_counts()
```

```
label_bias
Non-biased      1863
Biased          1810
No agreement       1
Name: count, dtype: int64
```

Solo nos interesan los no sesgados y los sesgados. Por lo que nos deshacemos del caso no acordado. Además, realizaremos un encoding binario para realizar operaciones con estas etiquetas.

```
# Encode label bias, now that we have only 2 categories: biased and
unbiased.
df_encoded = pd.get_dummies(df, columns=['label_bias'],
drop_first=True).rename(columns={'label_bias_Non-biased':'label_bias'
})
df_encoded['label_bias'] = (~df_encoded['label_bias']).astype(int)
```

Ahora, la columna es 0 si no es sesgado, y 1 si lo es.

Definiendo el nivel de sesgo

Para determinar que tan sesgado es un texto, utilizaremos 3 componentes del dataset: *label_bias*, la columna que determina si un artículo es sesgado (0 o 1), el número de palabras sesgadas en el texto (*biased_words)*, y la posición política del autor *type* (-1 si es de la izquierda, 0 si es central, 1 si es de la derecha). Con estos parámetros, definimos el *Bias_Metric* como el producto de estos valores:

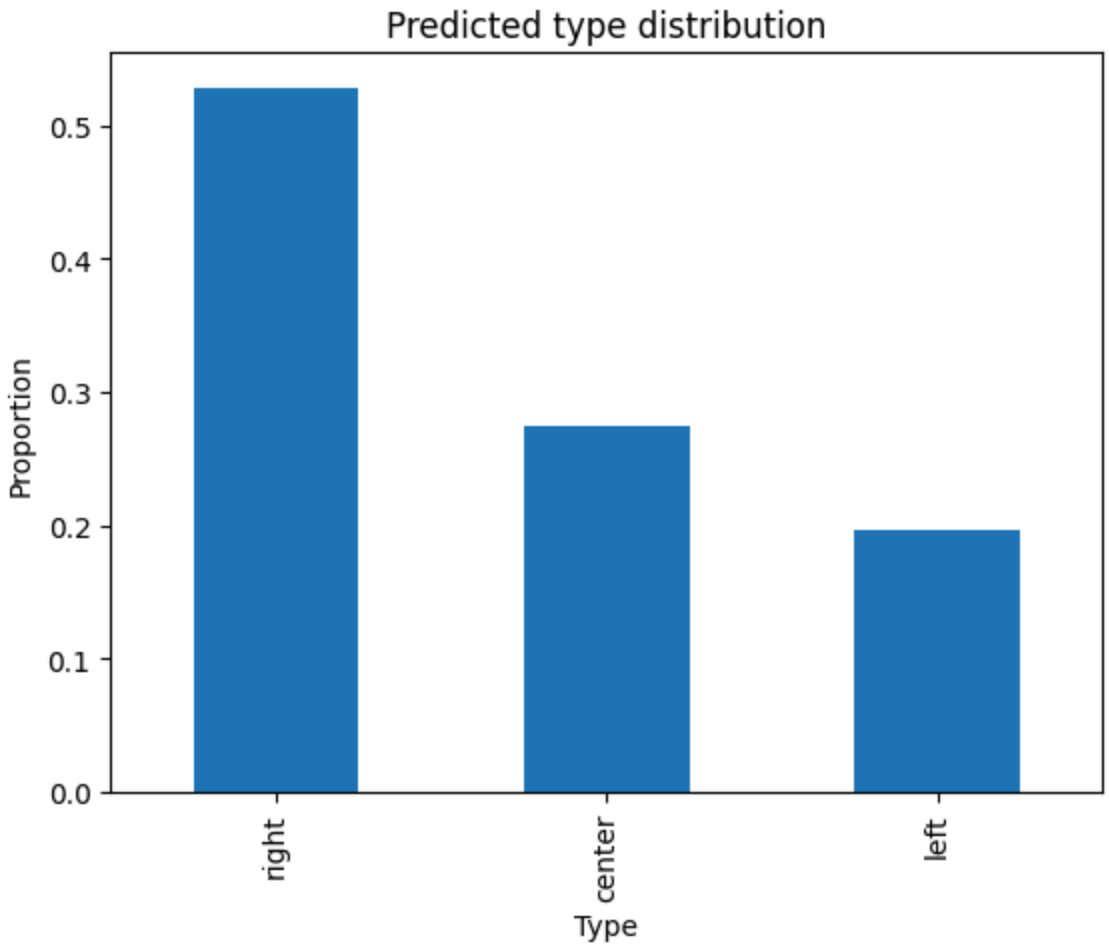
$$Bias\_Metric = label\_bias * len(biased\_words) * type$$

En código, lo escribimos como

```
# Returns the bias score for a given row
def score_bias(row):
    biased_word_count = count_biased_words(row['biased_words'])
    position = 1 if row['type'] == 'right' else 0 if row['type'] ==
'center' else -1
    bias = row['label_bias']
    return bias * position * biased_word_count

# Count number of biased words in a column.
# This expects a specific type of string.
# See the parsing function.
```

```
def count_biased_words(col):
    if isinstance(col, str):
        col = parse_string_array(col)
    return len(col)

# Parses the column with biased words.
# This columns is in the form of a string: "['word1', 'word2']",
# and not a list.
def parse_string_array(col):
    if col == '[]':
        return []
    res = []
    col = col.strip('"')
    col = col.lstrip('[')
    col = col.rstrip(']')
    for word in col.split(','):
        res.append(word.strip("'"))
    return res
```

Esto nos genera una columna de enteros con signo. Esta es la columna que vamos a predecir utilizando el texto del artículo. Esto nos da la siguiente distribución.



Correlaciones

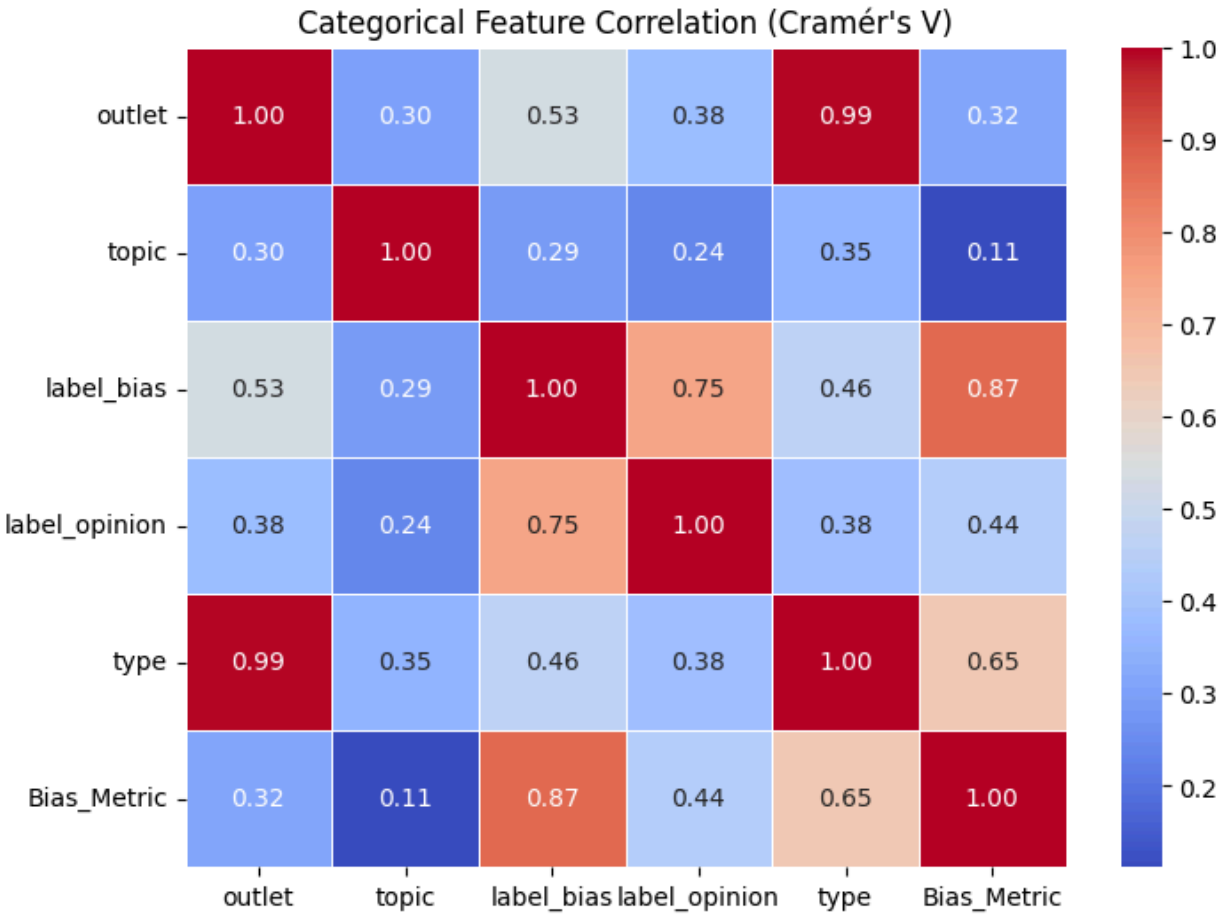
Primero, realizamos un análisis de correlaciones entre las variables. Como observamos anteriormente, la mayor parte de las variables son categóricas, por lo que deberemos utilizar *Cramér´s V* para determinar esta medida. *Cramér's V* se define como la correlación entre dos variables discretas, y se calcula mediante

$$V = \sqrt{\frac{\chi^2}{n \cdot \min\{k-1, l-1\}}},$$

Donde χ es la medida de la chi cuadrada, n es el número de *observaciones,* es decir, la cantidad de pares que resultan de hacer una contingencia entre dos columnas. Finalmente, k y l representan las dimensiones de nuestra tabla. En código, esto se representa como

```
import scipy.stats as stats

def cramers_v(x, y):
    # A calculation of Cramer's V for categorical-categorical
association
    confusion_matrix = pd.crosstab(x, y)
    chi2 = stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    r, k = confusion_matrix.shape
    return np.sqrt(chi2 / (n * (min(r, k) - 1)))

def categorical_correlation(df, categorical_cols):
    # Calculate and plot categorical feature correlation
    cat_corr = pd.DataFrame(index=categorical_cols,
columns=categorical_cols)
    for col1 in categorical_cols:
        for col2 in categorical_cols:
            if col1 == col2:
                cat_corr.loc[col1, col2] = 1.0
            else:
                cat_corr.loc[col1, col2] = cramers_v(df[col1],
df[col2])
    cat_corr = cat_corr.astype(float)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cat_corr, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
    plt.title("Categorical Feature Correlation (Cramér's V)")
    plt.show()
```

Que nos genera la siguiente matriz de correlación:



También podemos observar las palabras más comunes del dataset con un *WordCloud*.

Como podemos observar, las palabras más utilizadas (sin contar *stopping words*) son acerca del presidente.

Análisis profundos

Regresión

Introducción

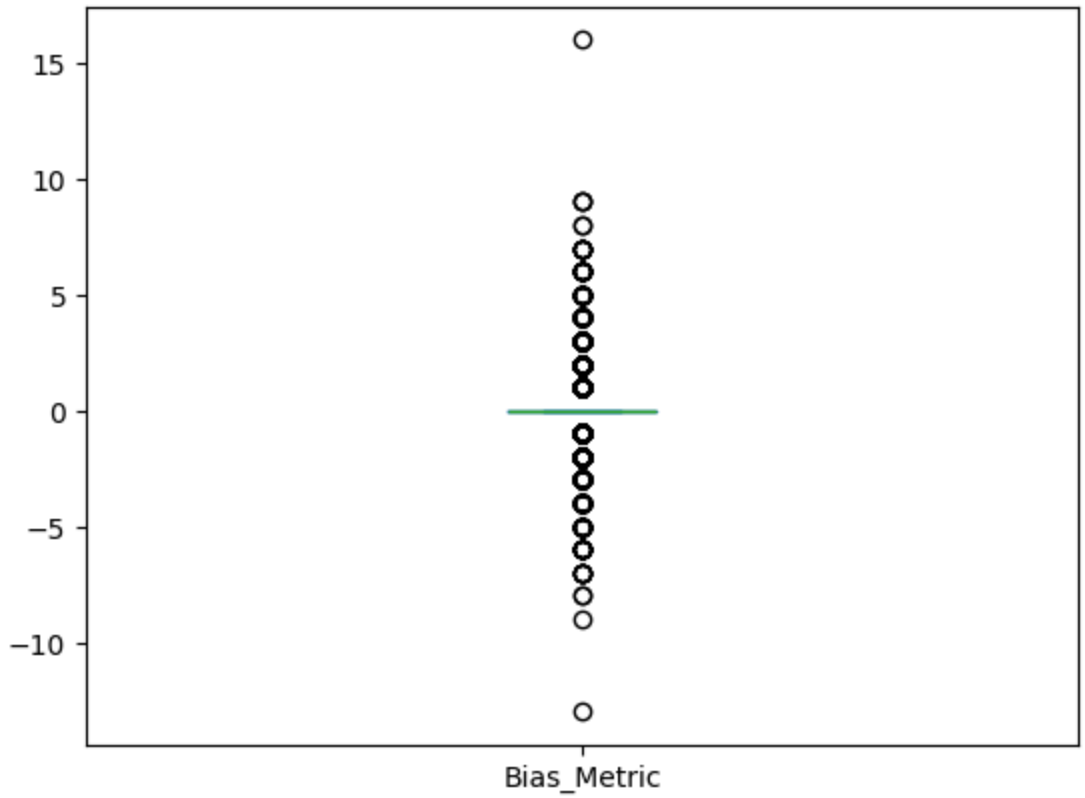
Las noticias que vemos en las redes, excluyendo la posibilidad de ser falsas o contener información errónea, pueden contener sesgo hacia un lado político. Con un modelo de regresión, podemos describir si una noticia contiene sesgo, hacia qué dirección se inclina, y que tanto se inclina.

La habilidad de distinguir si una nota intenta plasmar o exagerar ciertas características o acciones nos permite buscar una visión objetiva de la situación, y permitirnos crear conclusiones propias sin la manipulación política.

Debido a que el lenguaje es algo fácil de manipular, e incluso la selección de dataset puede influir en las decisiones del modelo, esta tarea genera incertidumbre en una meta de precisión por alcanzar.

Data Engineering

Como ya fue explicado en el apartado de la exploración del dataset, definimos la nueva columna de *Bias_Score*. Este es un boxplot de dicha columna.



Como podemos observar, hay un par de valores que se alejan mucho del grupo. Por lo tanto, hemos decidido quitar unos cuantos de estos *outliers*. En específico,

```
from scipy.stats import zscore

# Remove outliers
df[zscore(df['Bias_Metric']) > 4].shape
df_clean = df[zscore(df['Bias_Metric']) < 4]
```

Además, observamos ciertos datos que no concordaban con lo que se define como un medio 'central'. Es decir, noticieros del centro con artículos sesgados. Entonces, nos deshicimos de dichas ocurrencias. Para incorporar más ayuda al modelo, incluimos 2 variables que tienen que ver con el texto, *word_count* y *char_count*. La cantidad de palabras y la cantidad de caracteres en el texto.

Selección del modelo

Se realizaron pruebas con 2 principales estrategias de modelado: un simple Random Forest Regressor y una red neuronal convolucional. Primero, realizamos una vectorización con TfidfVectorizer para el texto. Hemos elegido 5000 max features debido al poco rendimiento que

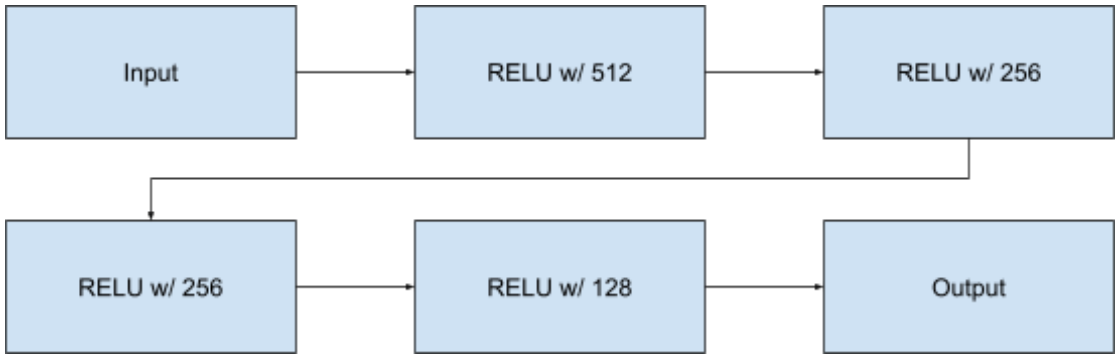
```
X = df_clean[['text', 'outlet', 'topic', 'word_count', 'char_count']]
y = df_clean['Bias_Metric']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Preprocessing phase, to transform text into numerical features and
encode categorical features, was well as normalize numerical features
preprocessor = ColumnTransformer([
    ('text', TfidfVectorizer(max_features=5000), 'text'),
    ('cat', OneHotEncoder(handle_unknown='ignore'), ['outlet',
'topic']),
    ('num', StandardScaler(), ['word_count', 'char_count'])
])

# Pipeline for the model, first preprocessing the data, then define
the model
model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=500,
random_state=42, max_depth=10, min_samples_leaf=4,
min_samples_split=10))
])
model.fit(X_train, y_train)
```

genera si aumentamos esta cantidad. Esto permite tener un modelo ligero. Además, realizamos una codificación para *outlet* y *topic*, y una estandarización de la cantidad de palabras y caracteres. Con un pipeline, conectamos este preprocesamiento al modelo, donde hemos escogido estos hiperparametros por resultado de un grid search. Este modelo nos genera un

resultado r^2 de **0.51** y un MAE de **0.77**. Debido a la naturaleza de lo que deseamos predecir, este resultado no es tan malo como parece.

Para el segundo modelo, realizamos muchas pruebas y configuraciones de una red neuronal.

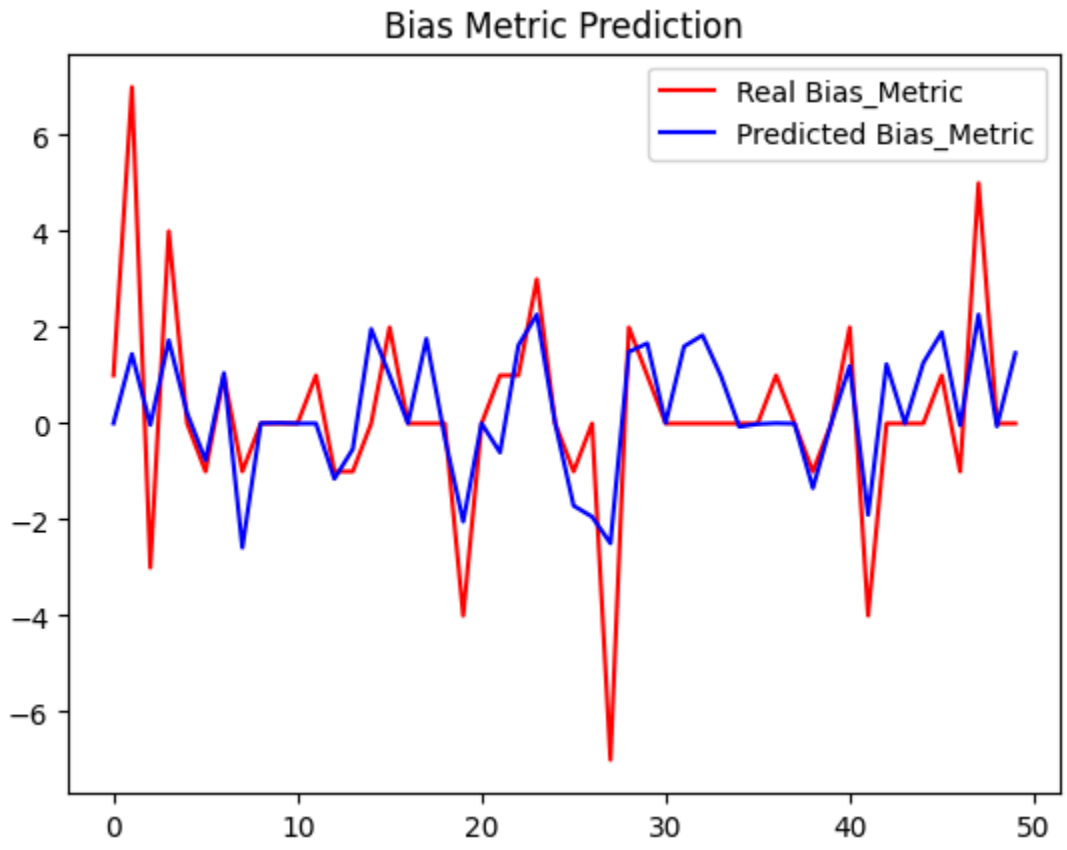
```
def create_nn(optimizer='nadam', neurons1=512, neurons2=256,
dropout_rate=0.1):
    model = Sequential()
    model.add(Dense(neurons1, activation='relu',
input_dim=X_train_transformed.shape[1]))
    model.add(Dense(neurons1 // 2, activation='relu'))
    model.add(Dense(neurons2, activation='relu'))
    model.add(Dense(neurons2 // 2, activation='relu'))
    model.add(Dense(1))  # Output layer
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
```




```
nn_model = KerasRegressor(model=create_nn, verbose=1, batch_size=64,
epochs=100, callbacks=[EarlyStopping(monitor='val_loss',
patience=5)])
```

Hemos incluido un callback para detener el entrenamiento si ya no mejora de manera sustancial. Probamos el uso de dropout entre las capas, pero esto resultó en un empeoramiento del modelo.

Esto nos genera la siguiente predicción para el conjunto de pruebas.



Con una puntuación r^2 de **0.51**, un MSE de **1.45** y MAE de **0.73.**

Interpretaciones

Con nuestra definición de medida del sesgo y los valores resultantes, podemos concluir que estos modelos pueden explicar parcialmente la varianza de los datos, pero aún pueden ser mejorados. Herramientas como BERT pueden resultar útiles para la captura de propiedades del texto y detectar características que TFIDF no logra captar. Sin embargo, BERT requiere demasiados recursos para hacer pruebas continuas, por lo que si se desea seguir usando TFIDF,

otras rutas pueden ser editar las capas de las redes neuronales o utilizar modelos stackeados. El criterio de medida de sesgo también es un tema a debatir, y la elección de esta función puede ser crucial en las predicciones.

Clasificación

Introducción

La abundancia de noticias falsas en el mundo puede tener consecuencias catastróficas tales como lo es la desinformación, los daños a la reputación y el pánico en la sociedad, por mencionar algunas.

Aunque es un tema complejo, abordar este problema es esencial, especialmente con la presencia dominante de las redes sociales en la vida cotidiana, donde cualquier tipo de información puede volverse viral sin importar la fuente.

En este contexto, el presente documento propone un modelo de análisis de sentimientos para evaluar la veracidad de las noticias, con el objetivo de identificar patrones y clasificar noticias falsas y verdaderas con la mayor precisión posible. Nuestra meta es desarrollar un modelo capaz de clasificar noticias de manera eficaz, con la finalidad de utilizarlo en contextos formales, como en la política, los medios de comunicación, entre otros.

Selección del modelo

Dentro de los modelos de clasificación más utilizados se encuentran la Regresión Logística, el Bosque Aleatorio de Clasificación y la Máquina de Soporte Vectorial. Para decidir qué modelo emplear en esta tarea, realizamos una validación cruzada (*cross-validation*) sobre una muestra del 30% de los datos, seleccionados aleatoriamente de un total de 44,678 ejemplares.

Para llevar a cabo dicha evaluación, seleccionamos TfidfVectorizer de *scikit-learn*, ya que para clasificar una noticia según sus características consideramos pertinente ponderar las palabras más frecuentes con valores bajos, mientras que las palabras menos comunes son

asignadas con valores altos; mejorando así la capacidad del modelo de distinguir información relevante.

$$IDF(palabra) = LOG\left(\frac{Número\ total\ de\ documentos\ en\ el\ corpus\ lingüístico}{Cantidad\ de\ documentos\ con\ la\ palabra}\right)$$

Además, TfidfVectorizer nos permite filtrar palabras de parada (*stopwords*) entre el texto que habíamos limpiado previamente con expresiones regulares (*regex*).

Nótese que, una noticia falsa clasificada incorrectamente como verdadera (*precision*) lleva al incremento del pánico en la población, pues se podría propagar desinformación que afecte decisiones y comportamientos; por otro lado, una noticia verdadera clasificada erróneamente como falsa (*recall*), ocasionaría desconfianza en los medios de comunicación y evitaría que información crucial llegue a las manos de las personas que realmente la necesitan.

Dado que ambos errores tienen consecuencias graves, consideramos pertinente evaluar los modelos respecto a una medida que balancee ambos aspectos, es decir, empleamos F1-score como métrica de rendimiento.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Obteniendo los siguientes resultados:

- Support Vector Classification: Accuracy = 0.9851 ± 0.0021
- Random Forest Classifier: Accuracy = 0.9782 ± 0.0026
- Logistic Regression: Accuracy = 0.9785 ± 0.0010

Dado que la máquina de soporte vectorial SVC mostró el mejor desempeño, decidimos continuar con este modelo para la clasificación de noticias.

Support Vector Classification (SVC)

Preprocesamiento

Previo a la creación del modelo, preprocesamos el texto eliminando números y otros caracteres no significativos. Sin embargo, no realizamos lematización ni alguna otra técnica de procesamiento de lenguaje natural, ya que consideramos que ciertos errores gramaticales podrían mejorar la clasificación al reflejar mejor el contexto y la estructura del lenguaje de las noticias. En su lugar, utilizamos TfidfVectorizer para codificar el texto en características numéricas y filtrar *stopwords*.

Configuración de Entrenamiento

Para evaluar el desempeño del modelo, los datos fueron divididos en un conjunto de entrenamiento y prueba utilizando una proporción 70%-30%

Con el objetivo de no solo clasificar, sino también estimar la confianza de la predicción, se habilitó la opción *probability=True* en el SVC, característica que utiliza validación cruzada de 5 pliegues.

También, para obtener los mejores resultados posibles, se realizó una búsqueda de hiperparámetros con GridSearchCV, donde se evaluaron diferentes valores de C (0.1, 1, 10) y dos tipos de kernel (linear y rbf). De modo que, al finalizar la búsqueda, los mejores parámetros encontrados fueron kernel = 'linear' y C = 1, lo que sugiere que los datos son linealmente separables y que el modelo escogió un margen de separación amplio, evitando tanto el sobreajuste (*overfitting*) como el subajuste (*underfitting*).

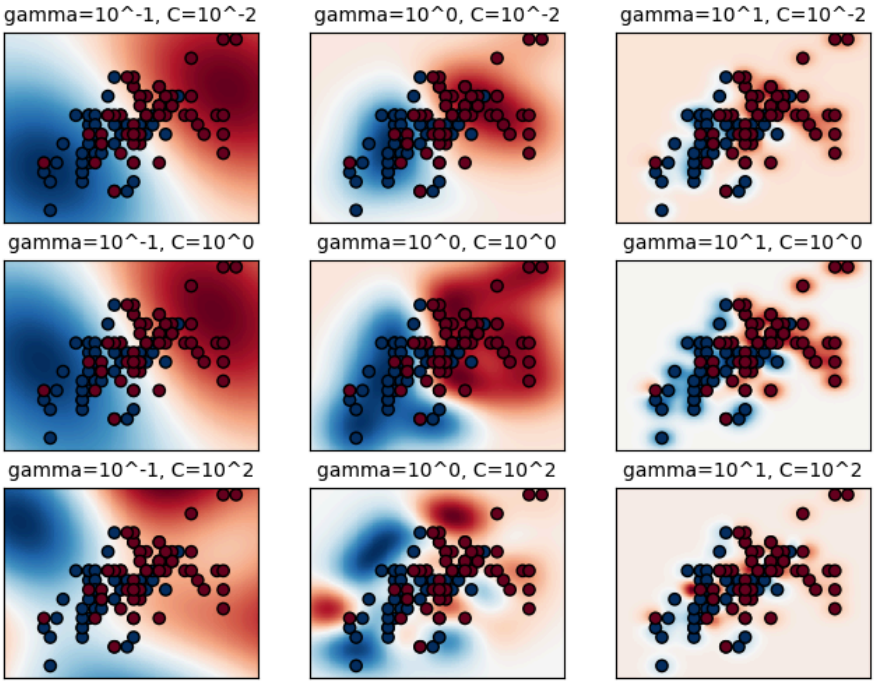
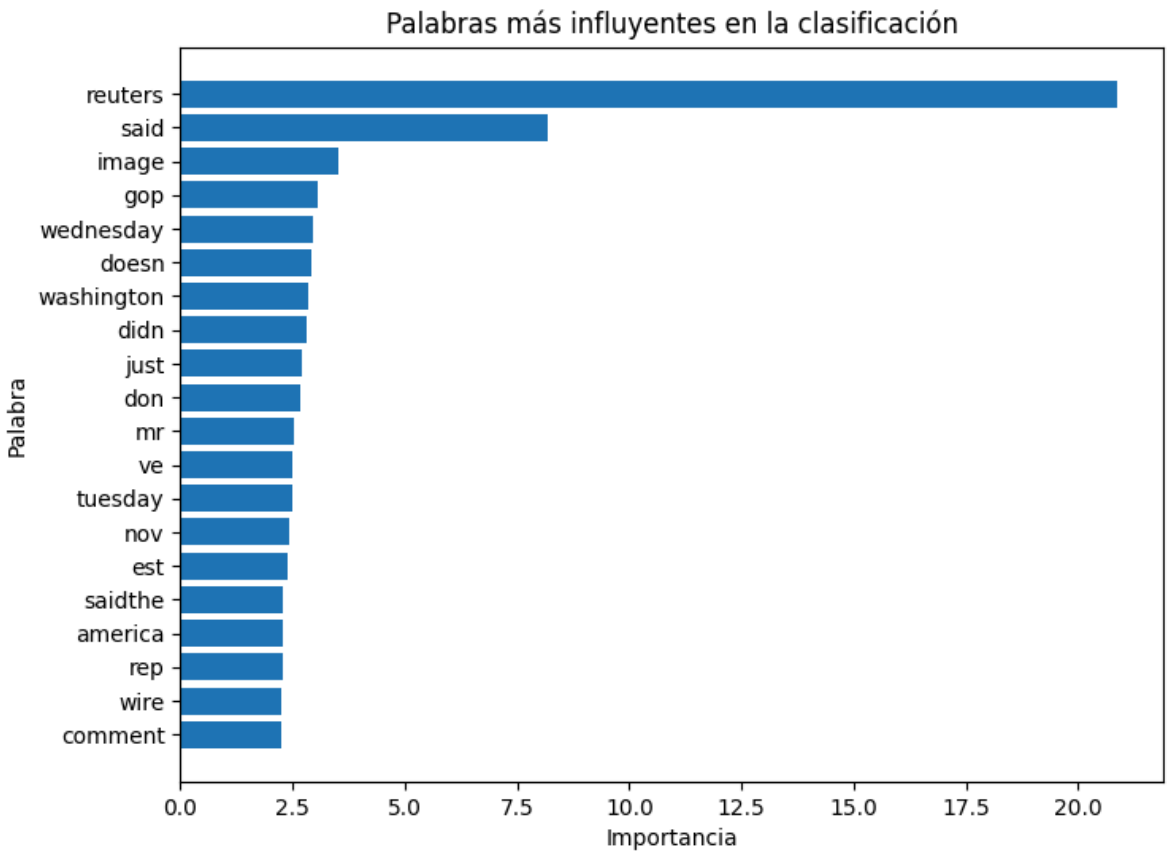


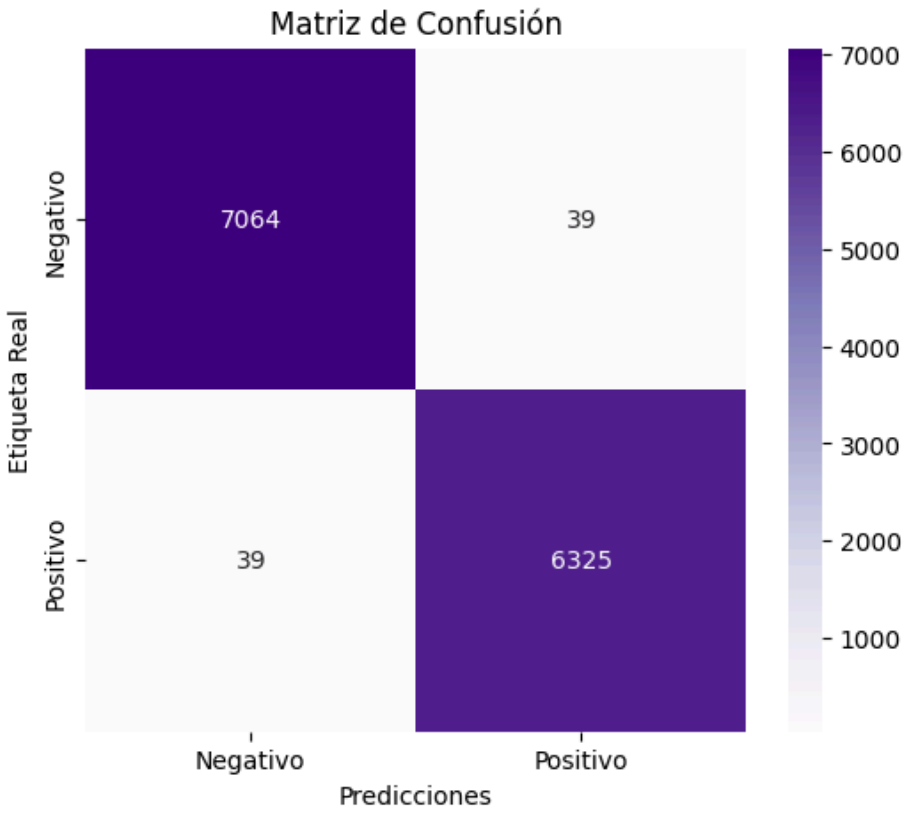
Ilustración para visualizar el comportamiento del parámetro C

Evaluación

Tras un largo tiempo de entrenamiento, los resultados obtenidos fueron los siguientes

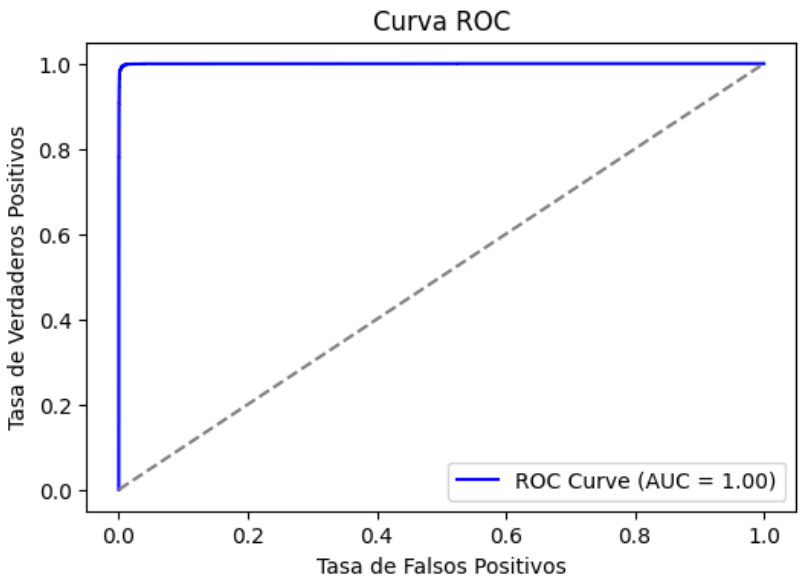


Clase	Precision	Recall	F1-score	Soporte
Negativo	0.99	0.99	0.99	7103
Positivo	0.99	0.99	0.99	6364
Accuracy			0.99	13,467
Macro Avg	0.99	0.99	0.99	13,467
Weighted Avg	0.99	0.99	0.99	13,467



Como se observa en los resultados, el modelo minimiza tanto los falsos positivos como los falsos negativos, obteniendo el balance buscado y demostrando un excelente desempeño con una exactitud del 99.42%.

Para estar más seguros del rendimiento del modelo, presentamos la curva ROC, la cual permite visualizar mejor las tasas de clasificación del modelo.



En este caso, el área bajo la curva (AUC) es prácticamente 1.00. Sin embargo, antes de concluir que el modelo es completamente fiable, es fundamental verificar que no haya sufrido sobreajuste (*overfitting*) a los datos de entrenamiento. Para ello, realizaremos pruebas adicionales en un conjunto de datos independiente, obtenido mediante *web scraping* de la página de noticias de la BBC, conformado por un *dataframe* con 25 entradas nuevas.

Noticias de la British Broadcasting Corporation (BBC)

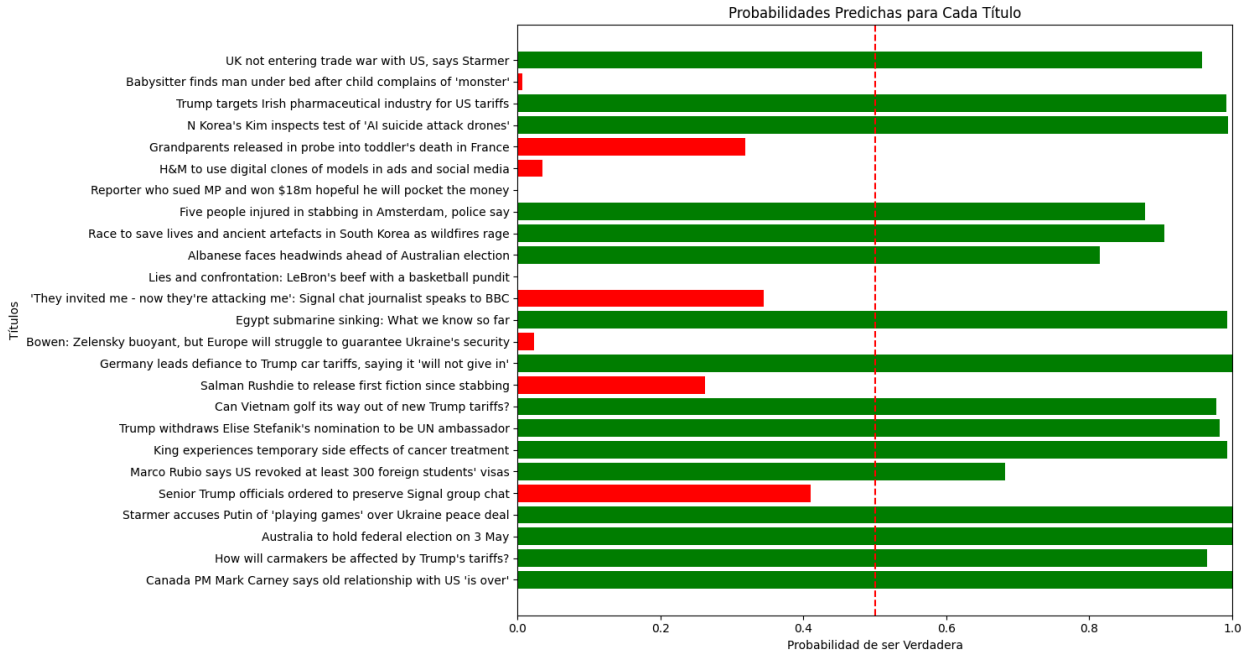
Web Scraping

El *scraping* se llevó a cabo en la página web de BBC News (https://www.bbc.com/news) donde, a través de los hipervínculos, se recopilaron 25 noticias con título, texto, etiquetas, contribuyentes y URL.

El conjunto de datos reunido incluye 41 etiquetas y 31 contribuyentes en total. Además, la cantidad de palabras por texto en este conjunto fue de 708 palabras de promedio, cantidad similar al promedio de palabras por texto con el que fue entrenado el modelo, que es de 904 palabras.

Evaluación

Al predecir las noticias del dataset formado se obtuvieron los siguientes resultados



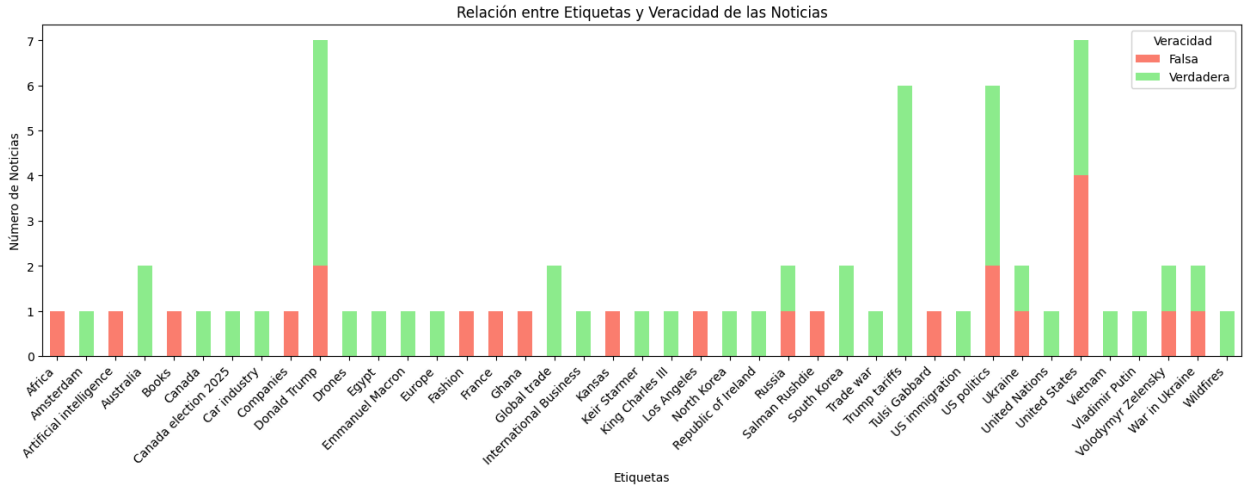
En los resultados obtenidos del conjunto de 25 noticias extraídas de BBC News, el modelo clasificó 16 noticias como verdaderas y 9 como falsas, lo que se traduce en una exactitud del 64%.

Este rendimiento, aunque razonable, plantea algunas preguntas. Una posible explicación radica en que las noticias con las que fue entrenado el modelo fueron mayoritariamente de índole política, lo que podría haber influido en su capacidad para clasificar adecuadamente noticias de otros géneros. Por ejemplo, títulos como "H&M to use digital clones of models in ads and social media" y "Babysitter finds man under bed after child complains of 'monster'" muestran temáticas muy distintas a las noticias políticas que predominaban en el conjunto de entrenamiento.

Otro aspecto relevante es la diferencia de longitud en los textos; por ejemplo, la noticia titulada "Babysitter finds man under bed after child complains of 'monster'" era notablemente más corta que el promedio, con solo 199 palabras, mientras que el promedio era de 708 palabras.

Esta discrepancia podría haber influido en la capacidad del modelo para generar representaciones precisas de textos más breves.

Además, se observó una curiosa discrepancia en la clasificación: la noticia "Starmer accuses Putin of 'playing games' over Ukraine peace deal" presentó una similitud coseno de 0.5 con la noticia "Bowen: Zelensky buoyant, but Europe will struggle to guarantee Ukraine's security". Sin embargo, la primera fue catalogada como verdadera y la segunda como falsa a pesar de su índice de similitud, lo que sugiere un sesgo hacia el tipo de información política a la que estuvo expuesto durante el entrenamiento.



Estas hipótesis explican en parte el comportamiento del modelo y sugieren un sesgo en el corpus de entrenamiento. No obstante, dada la sensibilidad de temas como la guerra en Ucrania o la política de Estados Unidos, la precisión en la clasificación de estas noticias es crucial y no puede tomarse a la ligera.

Por lo tanto, consideramos oportuno probar un enfoque diferente, como el uso de una arquitectura basada en transformers, específicamente BERT (Bidirectional Encoder Representations from Transformers).

BERT (Bidirectional Encoder Representations from Transformers)

¿Qué es?

BERT (Bidirectional Encoder Representations from Transformers) es un modelo de procesamiento de lenguaje natural (*NLP*) desarrollado por Google en 2018, basado en la arquitectura Transformer. Este tipo de arquitectura utiliza un mecanismo de atención que puede considerar una secuencia de palabras al mismo tempo, identificando relaciones entre palabras, incluso si están alejadas entre ellas. En el caso de BERT, se emplea un procesamiento bidireccional, lo que le permite comprender mejor el significado de las palabras basándose en el contexto en el que se encuentran dentro de una oración.

BERT es ampliamente utilizado en aplicaciones como Google Search para mejorar la precisión de las consultas en línea. Además, se aplica en el análisis de sentimientos, como en la clasificación de críticas positivas y negativas de películas, la detección de spam y, en el caso de esta investigación, en la clasificación de noticias falsas y verdaderas.

Selección del modelo

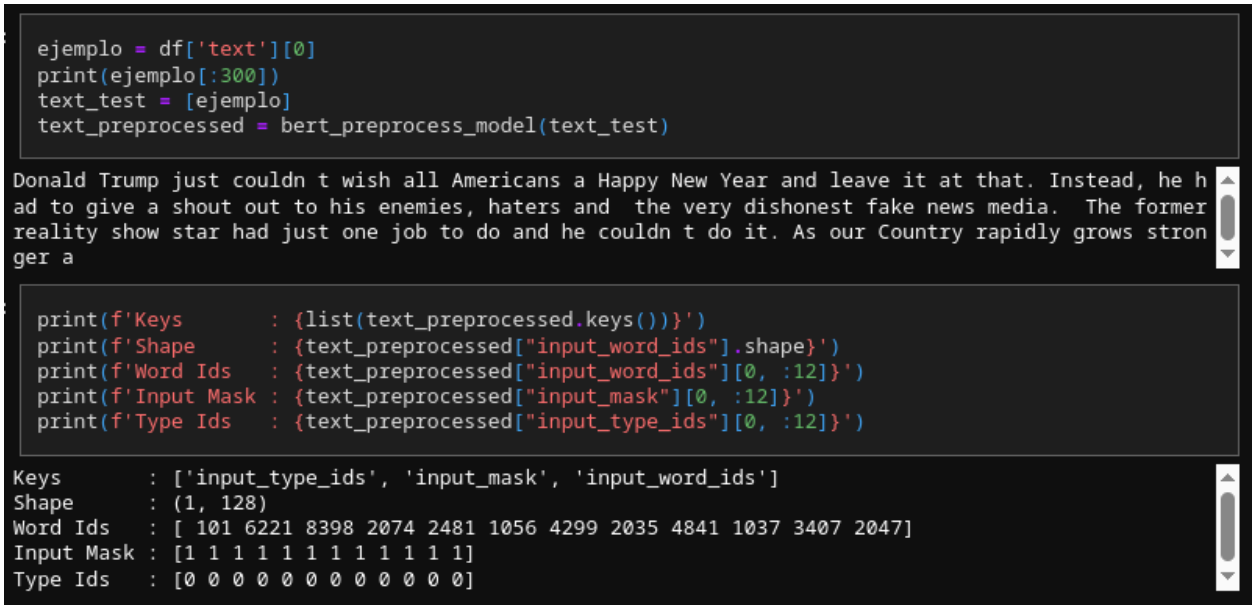
Tensorflow hub aloja diferentes modelos de BERT que han sido preentrenados con grandes corpus lingüísticos, también cuenta con una variedad de modelos de procesamiento. Para esta investigación se escogió el modelo de preprocesamiento https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3 que nos ayudara a transformar el texto crudo a tensores de entrada numéricos que posteriormente serán trasladados al modelo central. Este modelo es *uncased*, por lo que no distingue entre mayúsculas y minúsculas. Además, otro punto importante a mencionar es que la entrada se trunca a 128 *tokens*, lo que podría tener un impacto en el entrenamiento, ya que el conjunto de datos tiene un promedio de

453 tokens por entrada. Sin embargo, la configuración de la longitud de la entrada se deja como trabajo futuro.

Una llamada al modelo de preprocesamiento regresa un diccionario con 3 llaves 'input_type_ids', 'input_mask', 'input_word_ids'

- Input_word_ids: Representa el texto con identificadores numéricos (incluyendo los tokens de inicio [CLS], fin [SEP] y relleno [PAD])
- Input_mask: Indica cuáles son palabras reales y cuáles de relleno
- Input_type_ids: Distingue varios segmentos de texto por entrada, por ejemplo en las oraciones múltiples.

Podemos observar su funcionamiento con el siguiente ejemplo

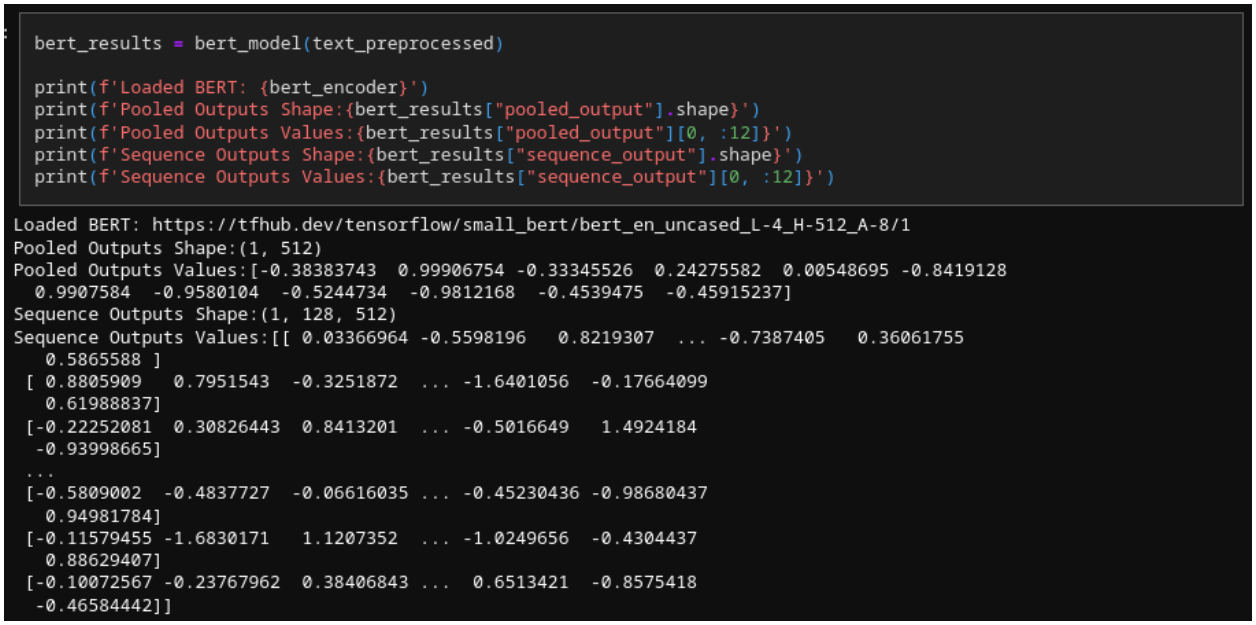


Aparte del modelo de preprocesamiento, se seleccionó un BERT pequeño (*small BERT*) https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1. Su elección se debe a la necesidad de un modelo eficiente que funcione dentro de nuestras restricciones computacionales. El presente modelo cuenta con 4 capas de Transformer, una dimensión de la representación oculta de 512 unidades, lo que significa que cada token es representado en 512

unidades para capturar la información contextual, y 8 cabezas de atención, que permiten al modelo enfocarse simultáneamente en diferentes partes de la oración.

Una llamada al modelo de BERT regresa un diccionario con 3 llaves importantes pooled_output, sequence_output, encoder_outputs. Para esta investigación utilizaremos pooled_output como salida, pues, regresa una representación compacta de cada entrada en 512 dimensiones. De seleccionar cualquier otro tipo de *output,* el tamaño de la salida sería más grande y, por ende, el modelo sería más complejo de entrenar.

A continuación, se muestra un ejemplo que ilustra el funcionamiento del modelo



```
bert_results = bert_model(text_preprocessed)

print(f'Loaded BERT: {bert_encoder}')
print(f'Pooled Outputs Shape:{bert_results["pooled_output"].shape}')
print(f'Pooled Outputs Values:{bert_results["pooled_output"][0, :12]}')
print(f'Sequence Outputs Shape:{bert_results["sequence_output"].shape}')
print(f'Sequence Outputs Values:{bert_results["sequence_output"][0, :12]}')
```

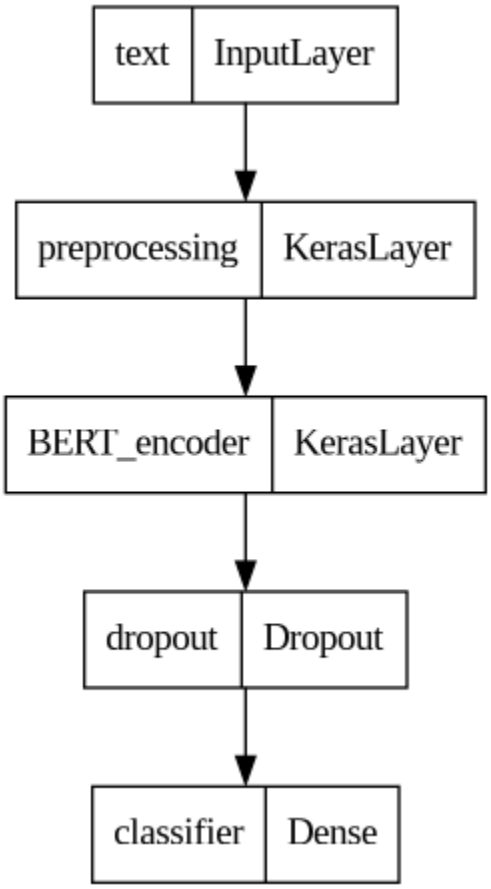
```
Loaded BERT: https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Pooled Outputs Shape:(1, 512)
Pooled Outputs Values:[-0.38383743  0.99906754 -0.33345526  0.24275582  0.00548695 -0.8419128
  0.9907584  -0.9580104  -0.5244734  -0.9812168  -0.4539475  -0.45915237]
Sequence Outputs Shape:(1, 128, 512)
Sequence Outputs Values:[[ 0.03366964 -0.5598196   0.8219307  ... -0.7387405   0.36061755
   0.5865588 ]
 [ 0.8805909   0.7951543  -0.3251872  ... -1.6401056  -0.17664099
   0.61988837]
 [-0.22252081  0.30826443  0.8413201  ... -0.5016649   1.4924184
  -0.93998665]
 ...
 [-0.5809002  -0.4837727  -0.06616035 ... -0.45230436 -0.98680437
   0.94981784]
 [-0.11579455 -1.6830171   1.1207352  ... -1.0249656  -0.4304437
   0.88629407]
 [-0.10072567 -0.23767962  0.38406843 ...  0.6513421  -0.8575418
  -0.46584442]]
```

Modelo de Clasificación

Capas

Una vez seleccionados el modelo de preprocesamiento y de codificación, pasamos a la construcción del modelo. Primero preprocesamos el texto a una representación que BERT pueda entender. Luego, esta información pasa al codificador BERT y seleccionamos solamente la salida *pooled_output*. Para evitar *overfitting*, se usa una capa *Dropout* con una tasa de abandono

del 10%, que desactiva aleatoriamente algunas conexiones. Finalmente, una capa densa procesa la información y genera la clasificación final.



Configuración

Dado que nos enfrentamos a un problema de clasificación binaria, se utilizó la función de pérdida *BinaryCrossentropy* para medir qué tan lejos están las predicciones del modelo respecto a las etiquetas reales. Y, se usó la métrica *BinaryAccuracy* para indicar el porcentaje de predicciones correctas.

También, se estableció un entrenamiento de 5 épocas (*epochs*), donde se calcula el número total de pasos por época a partir del tamaño del conjunto de entrenamiento. A partir de esto, se obtiene el número total de pasos de entrenamiento y los pasos de calentamiento

(*warmup*), que equivalen al 10% del total. Esta fase de warmup ayuda a que el modelo se estabilice al inicio del aprendizaje.

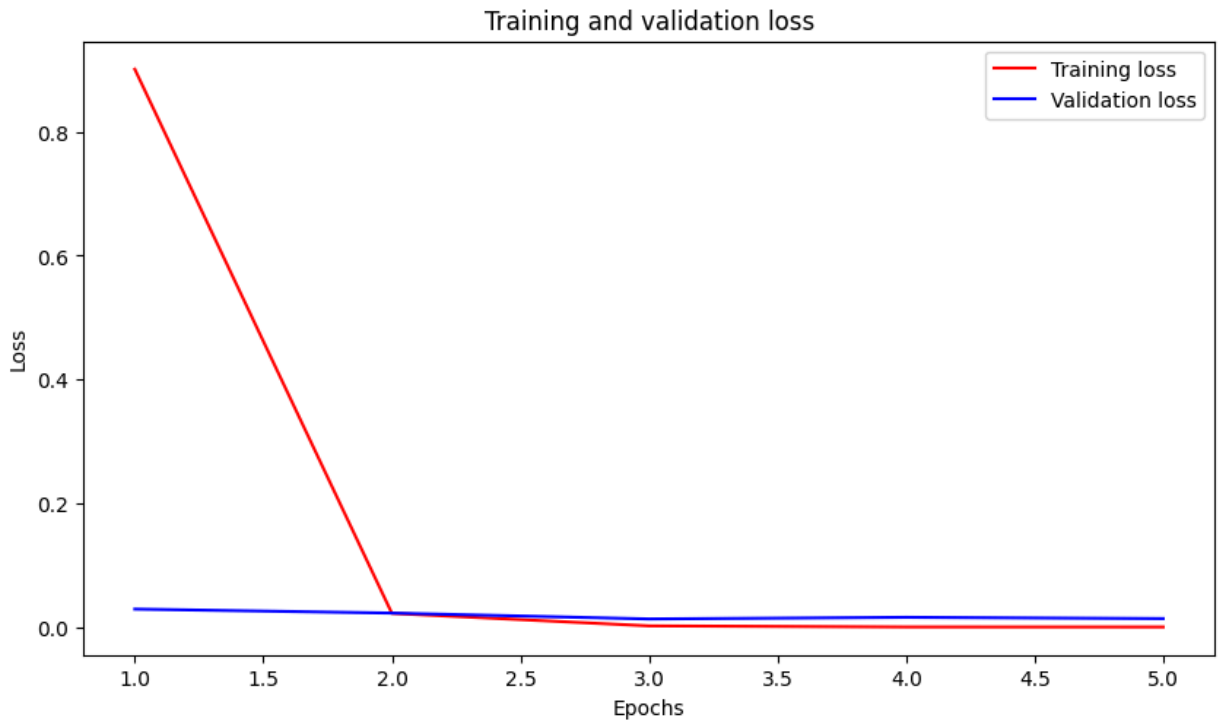
Luego se crea un optimizador *AdamW* (una versión mejorada del clásico Adam con regularización), con una tasa de aprendizaje inicial de 3e-5, lo cual es típico para modelos basados en Transformers como BERT.

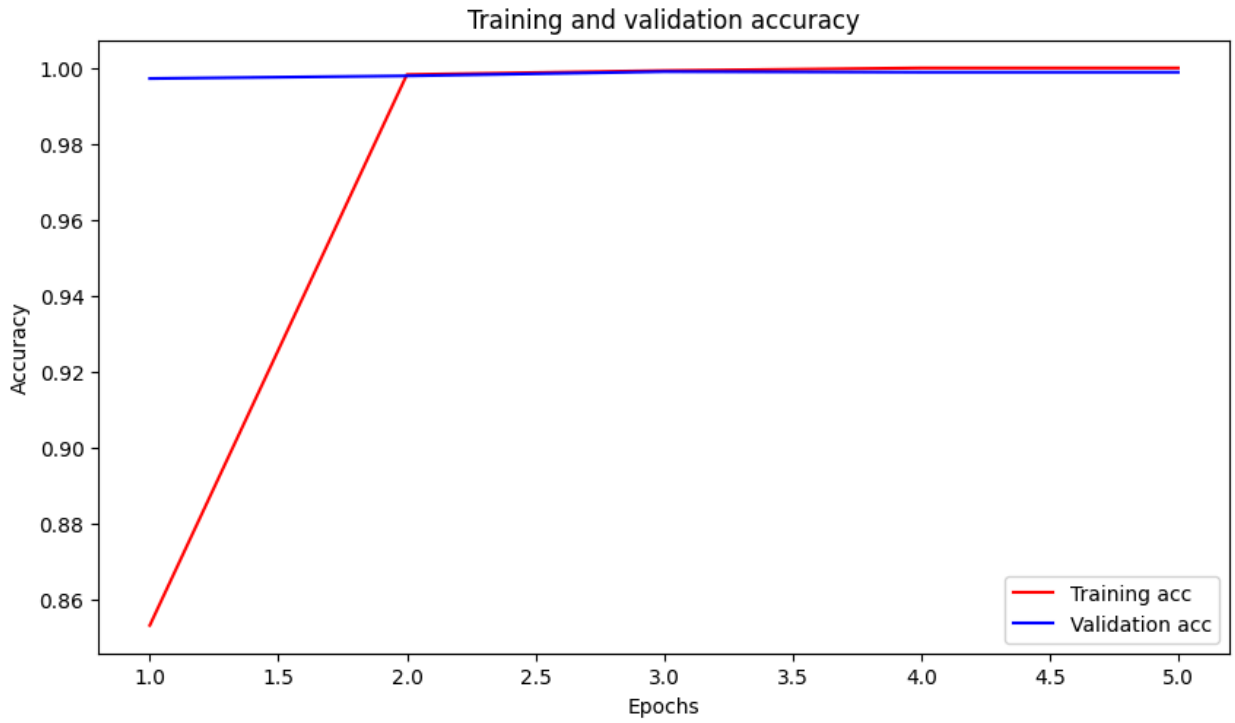
Por último, el modelo se compila. Esto deja listo al modelo para comenzar a aprender de los datos.

Entrenamiento

Al entrenar el modelo, se nota que aprendió muy rápido. En la primera época ya tenía una precisión del 85%, y para la segunda ya casi llegaba al 100%. La pérdida bajó mucho también, lo que significa que el modelo estaba entendiendo bien los datos.

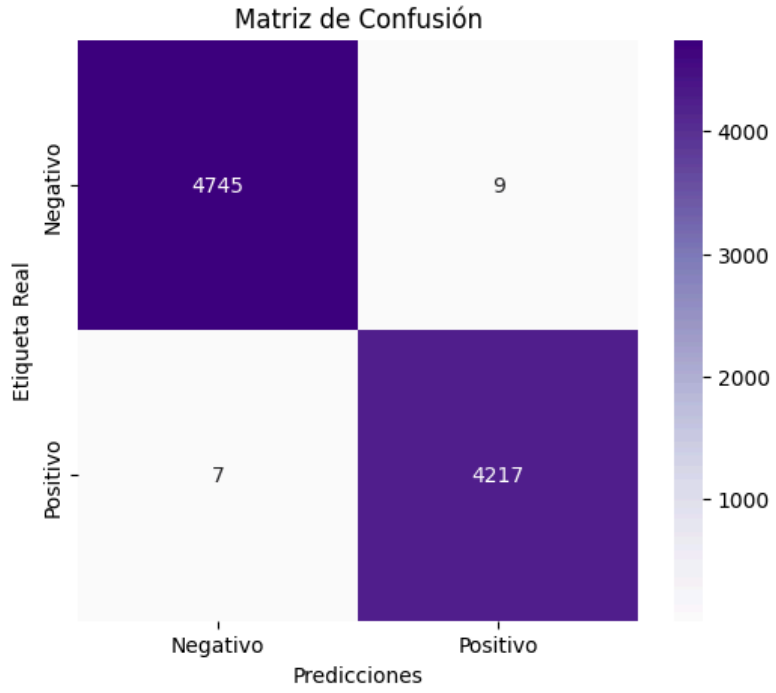
En cuanto a la validación, desde el inicio tuvo un rendimiento muy alto, arriba del 99%, y se mantuvo estable durante todo el entrenamiento. A su vez, la pérdida desde el inicio se mantuvo en valores bajos para bajar aún más.


Training and validation loss
Training loss
Validation loss
Loss
0.0
0.2
0.4
0.6
0.8
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs


Training and validation accuracy
Training acc
Validation acc
Accuracy
0.86
0.88
0.90
0.92
0.94
0.96
0.98
1.00
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epochs

Evaluación

El modelo logró una exactitud de 0.9982. Además, tanto la precisión, el recall y el f1-score están aproximadamente en 1.00, lo que indica que el modelo es muy confiable para ambos tipos de noticias. La matriz de confusión también lo confirma:



- De 4754 noticias negativas, solo 9 fueron clasificadas mal.
- De 4224 noticias positivas, solo 7 se confundieron.

Esto nos dice que el modelo no solo aprendió bien, sino que también generaliza muy bien con nuevos datos. Es una herramienta bastante sólida para detectar noticias falsas.

Conclusiones

Para el modelado del sesgo, los modelos de regresión, podemos concluir que estos aún tienen espacio por mejorar. Con una puntuación r^2 de **0.51**, esto nos da cierta explicación en el comportamiento del sesgo según el texto y otras características del medio publicador. Usar modelos con mayor complejidad como BERT podría mejorar este valor, a cambio de poder computacional.

El modelo SVC demostró un alto desempeño clasificando noticias políticas, alcanzando una exactitud del 99.42%. Sin embargo, al aplicarlo a noticias de otras temáticas, su rendimiento disminuyó significativamente, revelando un posible sesgo en los datos de entrenamiento. Esto evidenció la necesidad de modelos más robustos y generalizables, como BERT, que comprenden mejor el contexto y diversidad del lenguaje. Construyendo así una herramienta bastante sólida para detectar noticias falsas.

Si se desea consultar más a fondo el desarrollo del proyecto, consultar el siguiente repositorio https://github.com/jzarcoo/FakeNewsDetection

References

Emine Bozkus. (2023). *Fake news detection datasets* [Data set]. Kaggle. https://www.kaggle.com/datasets/emineyetm/fake-news-detection-datasets

Google, (2025). *Classification: Accuracy, recall, precision, and related metric*s. Google. Recuperado el 13 de marzo del 2025 de https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=es-419

Scikit-learn, (s.f.) *RBF SVM parameters*. Recuperado el 3 de abril del 2025 de https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Shaikh, J., & Patil, R. (s.f.). *Fake news detection using machine learning*. Departamento de Electrónica y Telecomunicaciones, Facultad de Ingeniería K.J. Somaiya, Mumbai, India. Recuperado el 1 de febrero del 2025 de https://svu-naac.somaiya.edu/C3/DVV/3.4.5/Confernce+and+Book+Chapter/234.pdf

Spinde, T., Plank, M., Krieger, J.-D., Ruas, T., Gipp, B., & Aizawa, A. (2021). *Neural media bias detection using distant supervision with BABE - Bias Annotations By Experts*. In *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp.

TensorFlow Core, (2020). *Making BERT Easier with Preprocessing Models From TensorFlow Hub*. TensorFlow. Recuperado el 15 de marzo del 2025 de https://blog-tensorflow-org.translate.goog/2020/12/making-bert-easier-with-preprocessing-models-from-tensorflow-hub.html?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

TensorFlow, (2024). *Solve GLUE tasks using BERT on TPU*. TensorFlow. Recuperado el 15 de marzo del 2025 de https://www.tensorflow.org/text/tutorials/bert_glue

TensorFlow, (2024). *Classify text with BERT*. TensorFlow. Recuperado el 15 de febrero del 2025 de https://www.tensorflow.org/text/tutorials/classify_text_with_bert#about_bert1166–1177). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.findings-emnlp.101