UNIVERSIDAD COMPLUTENSE DE MADRID

INGENIERÍA INFORMÁTICA

UNIVERSIDAD
COMPLUTENSE
MADRID

“PREDICCIÓN DE VALORACIONES DE KINDLE UTILIZANDO APRENDIZAJE AUTOMÁTICO”

Realizado por:

ALMUDENA GÓMEZ-SANCHA

CLARA MARTÍN NAVAS

ALFONSO BARRIGA LUCENA

Índice

1. Resumen

Nuestra práctica consiste en el desarrollo de un modelo de aprendizaje automático que se basa en redes neuronales para predecir la cantidad de estrellas que debería recibir una reseña de un libro en base a su contenido. Cargamos los datos de un database encontrado en kaggle, limpiamos los datos y entrenamos el modelo. Después introducimos una reseña inventada o existente (en inglés) y miramos cuál es su supuesta puntuación.

2. Introducción

Elegimos intentar resolver este problema por varias razones. En primer lugar, a los tres nos gusta mucho la lectura, y por ello nos llamó la atención el dataset inicialmente. Pensamos que era un tema original y que podía ser divertido. Luego se nos ocurrió la idea de entrenar un modelo que en base a una review predijese las estrellas que iba a tener porque nos pareció interesante y entretenido, ya que para hacer el testeo de nuestro modelo podríamos inventarnos nosotros mismos las reseñas e ir viendo que puntuaciones salían.

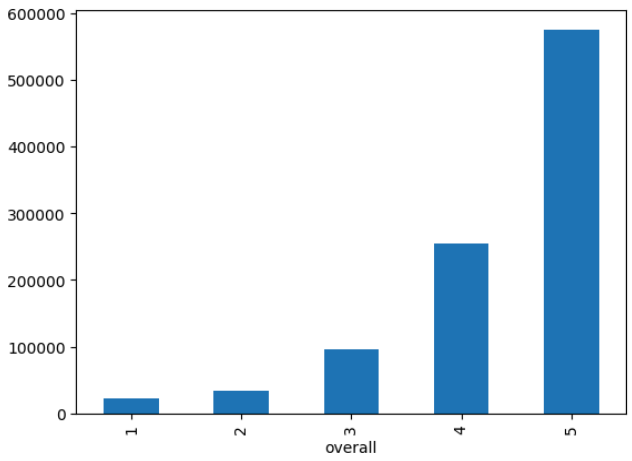
3. Dataset

Hemos seleccionado un dataset con reseñas de productos (libros) de la tienda Kindle de Amazon. Lo primero que hemos hecho es analizarlo para ver cómo son los datos para realizar la limpieza y preprocesado de manera adecuada.

El dataset tiene un total de 982619 con 10 columnas:

```
Unnamed: 0         int64
asin              object
helpful           object
overall            int64
reviewText        object
reviewTime        object
reviewerID        object
reviewerName      object
summary           object
unixReviewTime     int64
dtype: object
```

Las que nos interesan son 'overall' (como objetivo) y 'reviewText' (como entrada). La distribución de overall es así:



Vemos que está muy sesgada a 5 estrellas por lo que tendremos que lidiar con este desequilibrio a lo largo de la práctica.

Por otro lado, vemos las características estadísticas de 'reviewText':

```
count    982597.000000
mean        109.904708
std         130.813633
min           1.000000
25%          33.000000
50%          62.000000
75%         130.000000
max        4385.000000
Name: reviewText_length, dtype: float64
```

Observamos que la longitud media de las reseñas son 110 palabras pero tiene gran variabilidad ya que la desviación típica es 130. Por ello tendremos que filtrar las reseñas muy cortas y muy largas.

4. Descripción de la solución planteada

Nuestra solución consiste en construir un modelo de aprendizaje profundo usando tensorflow, una biblioteca desarrollada para construir y entrenar modelos de este tipo. La hemos elegido porque es útil para trabajar con redes neuronales como la nuestra, y permite trabajar con capas como Embedding o LSTM, ideales para capturar relaciones en secuencias de palabras, que es en lo que convertimos nuestras reseñas para poder trabajar con ellas. Scikit-learn se nos parecía más limitado para este caso.

En primer lugar cargamos y analizamos el dataset, que contiene las reseñas de los libros con sus calificaciones. Hacemos un análisis para identificar qué columnas son realmente importantes y cuales nos dan igual para resolver nuestro problema. La limpieza también elimina las filas duplicadas, filtra outliers en la longitud de las reseñas con rangos intercuartílicos, preprocesa el texto para borrar carácteres que no sean números o letras, quitar palabras inútiles, no diferenciar entre mayúsculas y minúsculas, y se encarga de

manejar las negaciones, ya que no es lo mismo tenerlas en cuenta que no. Si no las tuviéramos en cuenta, al modelo le parecería lo mismo “like” que “not like”, ya que ignorará las negaciones.

Luego entrenamos el modelo, usando una red neuronal con capas de Embedding, LSTM y densas. Regularizamos con Dropout para evitar el sobreajuste, y se ajustan los pesos de clases para manejar desequilibrios en las calificaciones.

Después se evalúa el modelo en un conjunto de pruebas usando métricas: precisión, accuracy y F1-score.

Finalmente tenemos una función que predice la calificación obtenida en base a una reseña introducida por nosotros.

4.1 Limpieza inicial de datos

Primero vamos a eliminar las variables que tienen información sin relación o utilidad en el análisis que añaden ruido. Eliminamos: 'Unamed: 0' ya que es un índice automático; 'helpful' ya que es complejo de procesar ahora, a lo mejor lo reincorporamos luego; 'reviewerName' ya tenemos id y es texto que sobra; 'summary' no nos aporta esta información; y 'unixReviewTime' ya tenemos ‘reviewTime’.

Además, eliminamos las filas repetidas para eliminar el sesgo que supondría el peso extra que tendrían esas reseñas en el entrenamiento y así reducimos el tamaño sin perder información. Suponemos que la fila está repetida si tiene exactamente el mismo texto de reseña ('reviewText') y la misma valoración ('overall').

A continuación, queremos eliminar los outliers para lo cual hemos utilizado el método del rango intercuartílico (IRQ). Este método se suele emplear si los datos no siguen la distribución normal ya que ayuda a definir los límites a partir de los cuales los datos se consideran atípicos. Estos límites se calculan así:

Límite inferior = Q1 — (k * IQR)
Límite superior = Q3 + (k * IQR)

IRQ es la medida de dispersión y es la distancia entre el primer y el tercer cuartil de una distribución. Este se multiplica por el factor k para encontrar los outliers, lo más común es utilizar k=1.5. Q1 es el primer cuartil y Q3 es el tercer cuartil (Brownlee, 2020).

Con esto, eliminamos las reseñas muy largas y muy cortas ya que sólo aportan ruido y pueden hacer que el coste computacional sea mucho mayor. Tras aplicarlos, se eliminan 89428 outliers de longitud que es como un 10% de los datos originales.

Por último, ponemos la fecha como fecha y no como object por si lo usamos en algún momento.

4.2 Preprocesado texto

Tras la limpieza inicial del texto, hemos procedido a procesarlo para que el modelo pueda interpretarlo bien y aprender de este. Para seleccionar qué criterios aplicar hemos investigado y probado las prácticas más comunes. Hay algunas cosas que no hemos aplicado como el stemming ya que reducía de más las palabras y. con esto, salía menor accuracy. Además, sin esto podríamos interpretar un poco mejor los resultados y las palabras más comunes y hacer que sea algo un poco menos “caja negra”.

Hemos desarrollado una función que limpie el texto siguiendo los siguientes criterios:

- Convertimos las contracciones a palabras por separado (ejemplo: don't -> do not)
- Ponemos todas las letras en minúsculas para que se trate igual la misma palabra (ejemplo: Palabra y palabra).
- Eliminamos los caracteres no alfabéticos
- Quitamos los artículos y preposiciones (stopwords) ya que su alta frecuencia y baja información no ayudan a discriminar
- Dentro de estas stopwords mantenemos not porque queremos mantener el significado de expresiones como "not like" y que no se conviertan en "like". Además, eliminamos palabras que tenían muchísima frecuencia en todas las reseñas y no aportan información: book, story y read.
- Tokenizamos las palabras

Para realizar la limpieza y entrenar al modelo, seleccionamos una muestra del 10% de todas las reseñas que son aproximadamente 90000.

Tras realizarla, vemos que las reseñas mantienen su significado pero están mucho más resumidas y queda la información más relevante por lo que hemos conseguido nuestro objetivo. Además, vemos que hay alguna o algunas reseñas que, tras la limpieza, se quedan con 0 de longitud así que las eliminamos ya que eran solo dos y así evitamos ruido.

overall	reviewText	reviewTime	reviewerID	reviewText_length	clean_review	tok_len
3	I wasn't sure what to expect and was on the fence about reading it boy am I glad for rainy days I read it in one seating and enjoyed it.	2013-02-18	A2KJ18SSPD4ZTS	30.0	not sure expect fence reading boy glad rainy days one seating enjoyed	12

Por último, al tokenizar las palabras para el modelo, tras experimentar, hemos elegido las 15000 más comunes. Y para eliminar ruido hemos usado el percentil 75 como maxlen en el padding ya que de esta manera cubrimos los casos útiles y así las secuencias más cortas no acaban llenas de ceros y se acelera el proceso. Elegimos el 75 y no el 80 o 90 que suele ser más común porque, en nuestro caso, la media de longitud es de 36 pero la más larga es de 220 así que es mejor ser más conservadores.

5. Experimentación

Hemos intentado encontrar el equilibrio entre capacidad y simplicidad durante toda la práctica de manera que el modelo tuviera suficiente capacidad como para resolver el problema pero, a la vez, que fuera fácil de ejecutar y tunear.

5.1 Modelado inicial

Empezamos con el siguiente modelo:

```
model = Sequential([
    #capa de embedding (vectorización de palabras)
    Embedding(input_dim=max_words, output_dim=128),
    #capa LSTM (red neuronal recurrente con dropout para regularización)
    LSTM(128,dropout=0.2, recurrent_dropout=0.2),
    #capa densa (red neuronal totalmente conectada con regularización L2)
    Dense(64, activation="relu", kernel_regularizer=L2(0.01)),
    Dropout(0.5),
    #capa de salida (predicción de la clase con activación softmax, 5 clases para 1-5 estrellas)
    Dense(5, activation="softmax")
])
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
```

La capa de embedding es un método de clustering que va agrupando palabras similares reduciendo así la dimensionalidad y creando un contexto para que la red tenga esa información y no empiece de cero.

Utilizamos una LSTM (Long Short-Term Memory) ya que es una red neuronal capaz de recordar datos relevantes y preservarlos unos instantes para aprender dependencias a largo plazo de datos. Esta capa captura dependencias en la secuencia de texto con un dropout de 0.2 para evitar el sobreajuste.

Después tenemos una capa Dense que reduce la dimensión de lo aprendido de la LSTM con una función de activación ReLU para que aprenda combinaciones no lineales y con regularización L2 para penalizar los pesos muy altos y, en consecuencia, mejorar la generalización. Lo combinamos con una capa de Dropout para introducir aleatoriedad en la activación.

Por último tenemos una capa densa para generar las probabilidades para cada una de las 5 clases. Para la función de pérdida usamos sparse_categorical_crossentropy ya que es más adecuada cuando las clases son enteros. Y usamos adam (Adaptive Moment Estimation) como optimizador ya que se adapta mejor a cada peso del embedding y de la LSTM y no necesita muchos ajustes como otros.

Lo hemos entrenado de manera que pare si deja de aprender o hay mucha pérdida de validación para no perder mucho tiempo o sobreentrenar.

```python
early = EarlyStopping(
    monitor="val_loss",
    patience=2,
    restore_best_weights=True
)
reduce_lr = ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.5,
    patience=1,
    min_lr=1e-6
)

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=10,
    batch_size=64,
    callbacks=[early, reduce_lr],
    class_weight=class_weight_dict
)
```

Tras entrenarlo y analizarlo nos hemos encontrado con un problema ya que el modelo sabe predecir muy bien si son 5 estrellas pero porque cómo hay tantas ha aprendido que suele acertar si predice 5 estrellas. Para arreglar este sesgo hemos conseguido equilibrarlo un poco al incluir los pesos pero no mucho. También hemos probado a hacer oversampling pero no funcionaba. Por ello, decidimos probar a balancear la muestra cogiendo cómo guía la clase con menos reseñas (1 estrella). Tras volver a probar el modelo con la nueva muestra balanceada vemos cómo el accuracy y la gráfica de confusiones mejoran bastante.

A continuación intentamos probar a hacer el modelo un poco más complejo pero no conseguimos que mejorara. Por ello decidimos probar a encontrar las relaciones entre palabras antes de entrenarlo para ayudar al modelo a aprender mejor y más rápido.

5.2 Segundo preprocesado y nuevo modelado

Decidimos extraer e incluir bigramas y trigramas antes de entrenar al modelo para capturar mejor los patrones de palabras, especialmente de negaciones, y así mantener el significado de las reseñas y para reforzar la señal de las frases al añadirlo al embedding. Usamos CountVectorizer para encontrar los más importantes e incluirlos en los datos de manera que ayuden a mejorar el modelo. Estos ngramas aportan un contexto que el embeddding entrenado de cero y la LSTM no podrían capturar en la misma magnitud.

Hicimos varias pruebas de cómo encontrarlos e integrarlos en los datos y así quedó tras encontrar una buena solución:

clean_review	tok_len	length	ngram_review_v2
worst mess seen grammar check poorly written even free get money back mess	13	26	worst mess seen grammar check poorly_written even free get_money_back mess
interesting short well written rumors ghosts college campuses one seemed bring ghost stories life	14	30	interesting short well_written rumors ghosts college campuses one seemed bring ghost stories life

De esta manera vemos que el significado de las frases permanece. Por ejemplo, poorly_written es distinto a well_written cuando antes lo tenía por separado y tenía que intentar relacionarlas en el entrenamiento.

Una vez terminado el nuevo preprocesamiento probamos el modelo anterior que teníamos pero el accuracy se estancaba alrededor del 40%. Esto nos llevó a pensar que el modelo se estaba quedando corto y que teníamos que hacer cambios. Decidimos probar distintas combinaciones, entre ellas, poner el LSTM bidireccional y añadir bloques convolucionales y de pooling para que detecte mejor patrones de palabras de distinta granularidad y para aligerar la carga del LSTM.

Con este modelo vimos que el accuracy se disparaba a 0.97 pero, a la vez, el accuracy de validación se estancaba en 0.45 lo cual significaba que habíamos sobre ajustado y la red estaba memorizando el train set. Por ello fuimos corrigiendo hasta dejar un solo bloque de convolución y pooling y añadimos capas de regularización (SpatialDropout y Dropout).

El modelo resultante fue este:

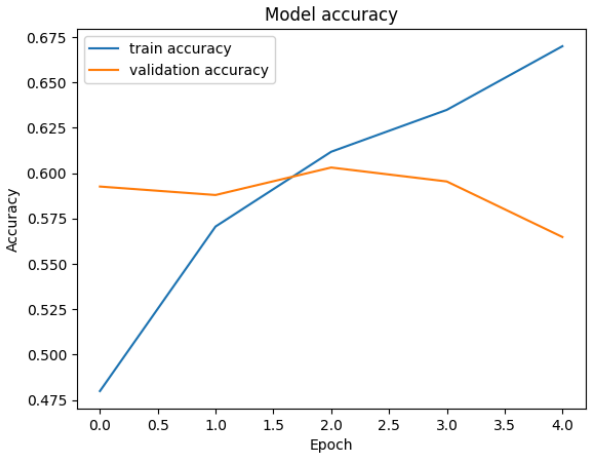
```
model = Sequential([
    Embedding(max_words, 128, input_length=max_len),
    SpatialDropout1D(0.3),

    Conv1D(64, 3, activation="relu", padding="same"),
    BatchNormalization(),
    MaxPooling1D(2),
    Dropout(0.3),

    Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2)),
    Dense(64, activation="relu", kernel_regularizer=L2(0.01)),
    Dropout(0.5),
    Dense(5, activation="softmax")
])
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
```

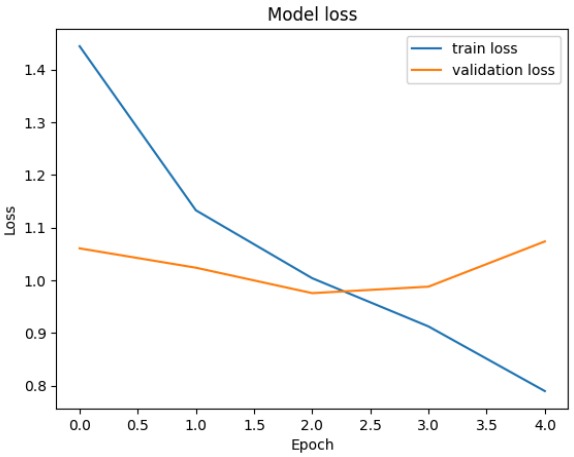
Con todo esto conseguimos que el accuracy y el balanced accuracy subieran pero, sobre todo, conseguimos que el modelo reconociera mejor las reseñas de tres estrellas además de las de una y cinco.

6. Resultados

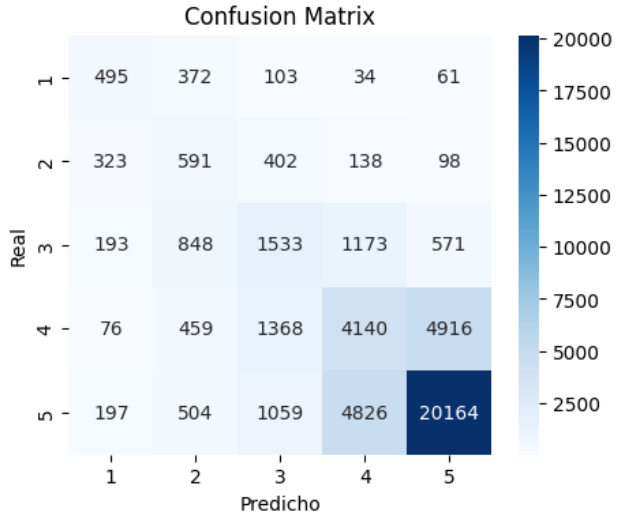
Hemos comprobado los resultados obtenidos por nuestro modelo de tres formas: con gráficas que nos indican la precisión de entrenamiento o la de validación, la matriz de confusión, y con las puntuaciones finales que da el modelo tras introducir una reseña.

6.1 Primer modelo

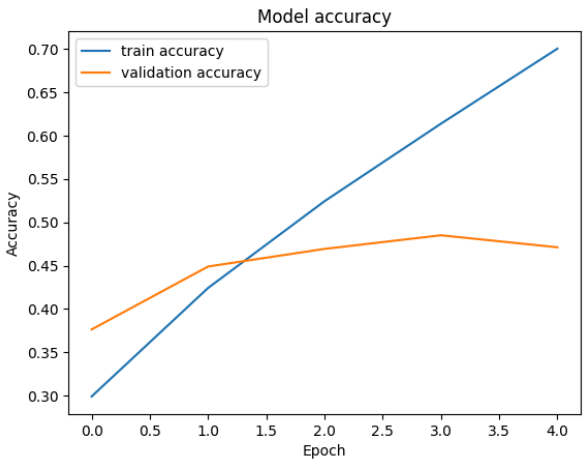


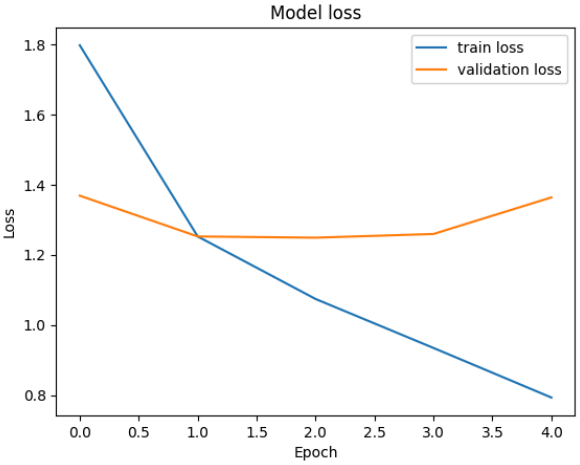


	precision	recall	f1-score	support
1★	0.386	0.465	0.421	1065
2★	0.213	0.381	0.273	1552
3★	0.343	0.355	0.349	4318
4★	0.402	0.378	0.389	10959
5★	0.781	0.754	0.767	26750
accuracy			0.603	44644
macro avg	0.425	0.466	0.440	44644
weighted avg	0.616	0.603	0.609	44644

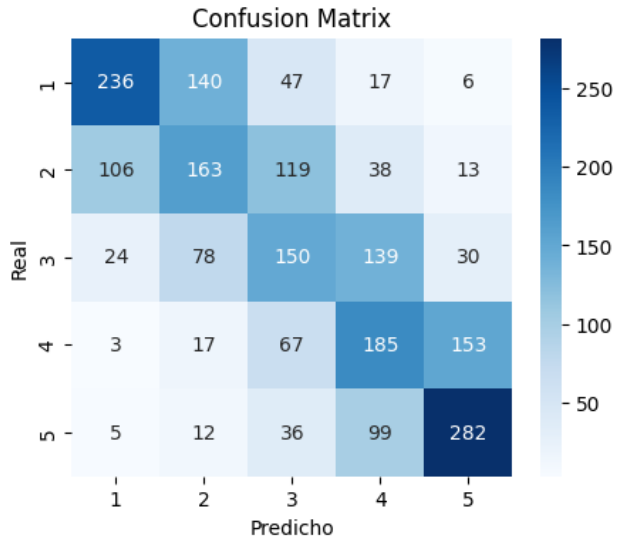


6.2 Primer modelo con muestra balanceada



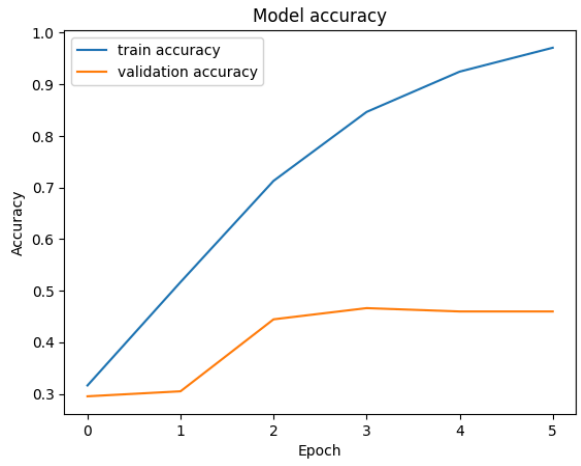


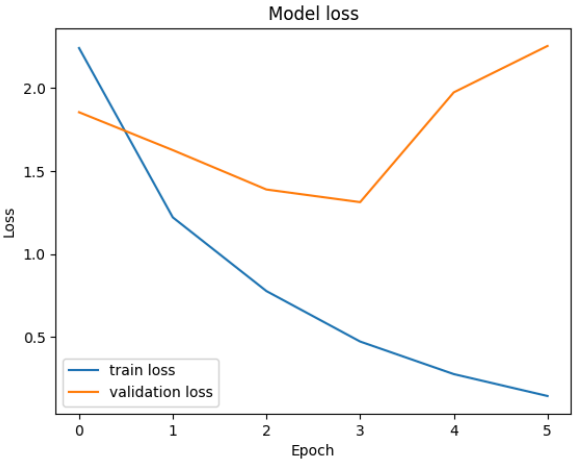
	precision	recall	f1-score	support
1★	0.631	0.529	0.576	446
2★	0.398	0.371	0.384	439
3★	0.358	0.356	0.357	421
4★	0.387	0.435	0.410	425
5★	0.583	0.650	0.614	434
accuracy			0.469	2165
macro avg	0.471	0.468	0.468	2165
weighted avg	0.473	0.469	0.469	2165



6.3 Segundo modelo (sobre ajustado)

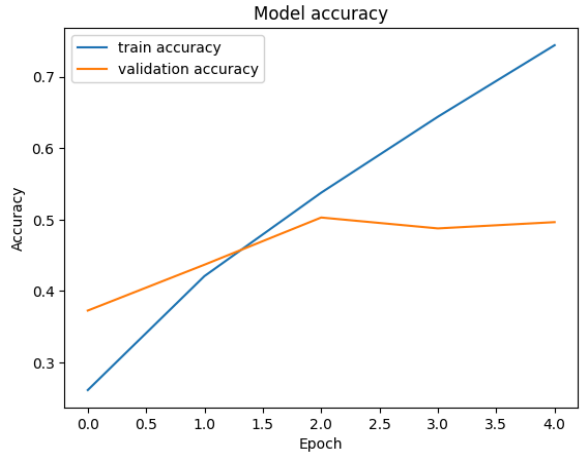
```
136/136 ━━━━━━━━━━━━━━━━━━━━ 15s 34ms/step - accuracy: 0.2555 - loss: 2.7313 - val_accuracy: 0.2952 - val_loss: 1.8525
Epoch 2/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.5051 - loss: 1.2893 - val_accuracy: 0.3048 - val_loss: 1.6248
Epoch 3/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.7240 - loss: 0.7814 - val_accuracy: 0.4443 - val_loss: 1.3878
Epoch 4/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 0.8450 - loss: 0.4811 - val_accuracy: 0.4661 - val_loss: 1.3117
Epoch 5/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 0.9267 - loss: 0.2817 - val_accuracy: 0.4596 - val_loss: 1.9721
Epoch 6/10
136/136 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.9722 - loss: 0.1519 - val_accuracy: 0.4596 - val_loss: 2.2510
```

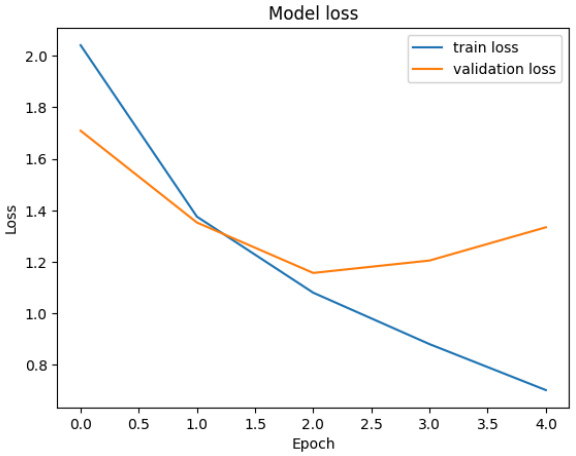




6.4 Modelo definitivo

GRÁFICA DE PRECISIÓN Y PÉRDIDA

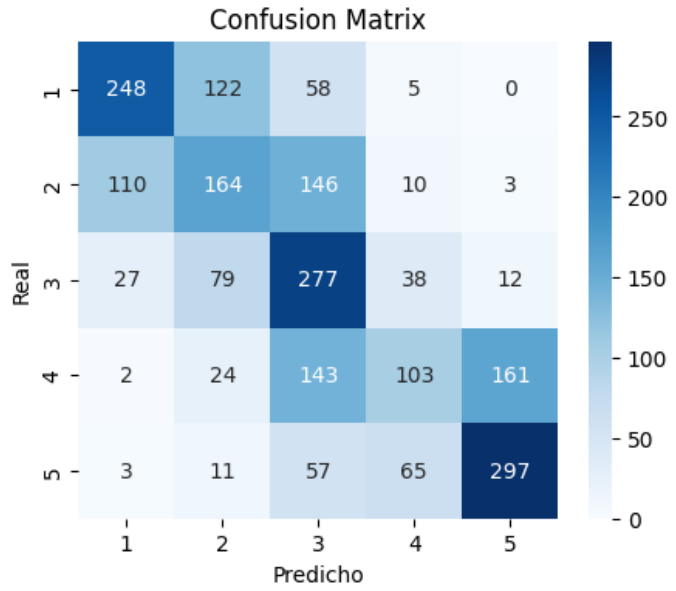




MATRIZ DE CONFUSIÓN

Vemos que la precisión ha aumentado en general en todas las clases y métricas comparado con el primer modelo con la muestra balanceada. Además, vemos en la matriz de confusión que hemos conseguido que acierte en las reseñas de tres estrellas.

	precision	recall	f1-score	support
1★	0.636	0.573	0.603	433
2★	0.410	0.379	0.394	433
3★	0.407	0.640	0.497	433
4★	0.466	0.238	0.315	433
5★	0.628	0.686	0.656	433
accuracy			0.503	2165
macro avg	0.509	0.503	0.493	2165
weighted avg	0.509	0.503	0.493	2165



PREDICCIONES FINALES

Finalmente estas son algunas de las predicciones que ha hecho el modelo en base a algunas de las reseñas que nos hemos inventado. Podemos observar que son bastante acertadas. El modelo actúa mejor si las reseñas son explícitas, es decir, si contienen palabras que caracterizan a su puntuación, por ejemplo, si detecta palabras negativas como hate, boring, etc. le dará peor puntuación, pero si son positivas como love, amazing, great, le premiará.

```
new_review = "This book was awful and it disappointed me! It was super slow and i was falling sleep"
```

predicción: 2 estrellas

```
new_review = "This book was great and it was great! It was super fun and i was loving it"
```

predicción: 5 estrellas

7. Conclusiones y posibles mejoras

El modelo desarrollado con mayor accuracy y más balanceado es la combinación de un preprocesado más elaborado y un modelo más complejo que el inicial. Con esta combinación hemos conseguido llegar a un modelo con una matriz de confusión bastante buena teniendo en cuenta los límites y la simplicidad de este trabajo. También hemos visto

la importancia de los datos de entrada y el impacto que tiene la limpieza de datos y el preprocesado del texto para mejorar el aprendizaje. Por último, hemos aprendido que algo clave es encontrar el equilibrio entre complejidad y sencillez.

Algunas posibles mejoras a futuro serían:

- Aplicar embeddings preentrenados: esto permitiría una representación semántica mucho más realista y con términos poco frecuentes.
- Usar arquitecturas más complejas: como modelos Transformer que son muy efectivos en procesamiento de lenguaje natural.
- Regularización y ajuste de hiper parámetros: se podrían probar diversas combinaciones para encontrar modelos más robustos.

Referencias y bibliografía

https://www.kaggle.com/datasets/bharadwaj6/kindle-reviews

Brownlee, J. (2020). *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery.

Anexo

Tareas realizadas por cada miembro del grupo:
Almudena: limpieza y tokenizacion de los datos, arreglos y mejoras del código, redacción de memoria.
Clara: entrenamiento del modelo, arreglos y mejoras de código, redacción de memoria.
Alfonso: título del proyecto, PowerPoint y portada del word.