UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERIA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Universidad de los Andes
Facultad de Ingeniería

TAREA 4 – ISIS 4221

Presented to:
Rubén Francisco Manrique

Presented by
Tomas Acosta Bernal - 202011237
Santiago Pardo Morales - 202013025
Juan Esteban Cuellar Argotty - 202014258
Ayman Benazzouz El Hri - 202424848

BOGOTÁ DC - COLOMBIA

Index

Entrenamiento de Embeddings

1.1. Selección de autores y libros

Hemos escogido los tres autores:
Charles Dickens con sus tres libros: *A Tale of Two Cities*, *Bleak House* y *Oliver Twist*.
Fyodor Dostoyevsky con sus tres libros: *Crime And Punishment*, *The Gambler* y *The Idiot*.
Alexandre Dumas con sus tres libros: *Count of Monte Cristo*, *Twenty Years After* y *The Three Musketeers*.

```
import os
# Directorio donde est n los archivos .txt
data_dir = 'data/books/'
# Leer todos los archivos .txt del directorio
texts = []
for filename in os.listdir(data_dir):
    if filename.endswith('.txt'):
        print(filename)
        with open(os.path.join(data_dir, filename), 'r', encoding='utf-8') as f:
            texts.append(f.read())
            print(f.read()[:2500])
```

Primero hemos guardado los 9 libros (3 de cada autor) en una lista `texts` donde cada elemento presenta el contenido del libro.

1.2. Procesamiento de textos

```
def process_all_texts(processor, texts):
    total = len(texts)
    processed_texts = []
    for index, text in enumerate(texts):
        processed_text = processor.preprocessing_pipeline(text)
        processed_texts.append(processed_text)
    return processed_texts
```

Preparamos los datos que tenemos (el contenido de los libros) usando la función `preprocessing_pipeline()` definida en la clase `processor.py`, que realiza el siguiente preprocesamiento:

- **Gutenberg licensing removal**: limpia los datos de avisos legales y detalles de edición.
- **Tokenization**: divide el texto en tokens.
- **Lowercase conversion**: convierte todas las palabras a minúsculas.
- **Punctuation removal**: elimina la puntuación.
- **Word-to-number conversion**: convierte palabras que representan números en su forma numérica.
- **Digit replacement**: reemplaza los números con la cadena 'NUM'.
- **Stopword removal**: elimina stopwords utilizando el corpus de NLTK.
- **Stemming**: reduce las palabras a su raíz.

1.3. Cálculo del tamaño del vocabulario

```
def get_vocabulary_size(processed_texts):
    vocabulary = set()
    for text in processed_texts:
        vocabulary.update(text)
    return len(vocabulary)

vocab_size = get_vocabulary_size(processed_texts)
print(f"Tama o del vocabulario: {vocab_size}")
```

Entrada: processed_texts (lista de textos que presentan nuestro conjunto de datos).

Esta función se usa para calcular el tamaño del vocabulario en nuestro conjunto de datos, usando la estructura de datos `set` que nos permite solo guardar los valores no repetidos para poder identificar el tamaño del vocabulario.

Salida: no retorna una salida, solo imprime el tamaño del vocabulario.

1.4. Entrenamiento y guardado de modelos

```
def train_and_save_models(processed_texts, dimensions=[100, 200, 300]):
    for size in dimensions:
        print(f"Training model with dimensionality {size}")
        model = Word2Vec(sentences=processed_texts, vector_size=size, window=5, min_count=1)
        model_filename = f"data/answers/Books_{size}_group_code.model"
        model.save(model_filename)
        print(f"Model saved as {model_filename}")

train_and_save_models(processed_texts)
```

Entrada: processed_texts (la lista de los textos que se usarán para entrenar los modelos) y dimensions (las dimensiones de los embeddings para crear los modelos).

Se usa la función `Word2Vec` de la librería **Gensim** para crear y entrenar 3 modelos, usando los textos indicados en los parámetros, con una ventana de 5 palabras, y cada uno con una dimensión de embeddings indicada en los parámetros.

Salida: no retorna nada, sino guarda cada modelo entrenado en un archivo.

Visualización de Embeddings

2.1. Identificación de Protagonistas de Cada Historia

Para visualizar embeddings en dos dimensiones y explorar relaciones entre palabras, hemos utilizado la técnica t-SNE (t-Distributed Stochastic Neighbor Embedding) para reducir la dimensionalidad. Esta técnica permite proyectar los embeddings de Word2Vec que se caracterizan por tener una alta dimensionalidad en un espacio de dos dimensiones para su visualización.
En este caso se utilizó esta visualización para detectar las palabras que podían rodear a los protagonistas principales, el propósito de esto es identificar patrones o analogías de escritura similares que utilizan todos los actores en obras. Esta identificación se realizó a través del hallazgo de los autores principales, los cuales se encontraron mediante una breve revisión de literatura y una consulta sobre quiénes eran los personajes principales de cada obra, basado en esto se encontró que los perosnajes principales de cada obra fueron

- **A Tale of Two Cities - Charles Dickens:** Charles, Sydney, Lucie, Dr Alexandre Manette, Madame Defarge
- **Bleak House - Charles Dickens:** Esther, John, Lady Dedlock, Richard
- **The Count of Monte Cristo - Alexandre Dumas:** Edmond, Mercedes, Fernand, Danglars
- **Crime and Punishment - Fyodor Dostoyevsky:** Rodion, Sonia, Dmitri, Porfiry
- **Oliver Twist - Charles Dickens:** Oliver, Fagin, Nancy, Bill Sikes, Mr Brownlow
- **The Gambler - Fyodor Dostoyevsky:** Alexis, Polina, General
- **The Idiot - Fyodor Dostoyevsky:** Lef, Nastasia, Parfen, Aglaya
- **The Three Musketeers - Alexandre Dumas:** D'Artagnan, Athos, Porthos, Aramis
- **Twenty Years After - Alexandre Dumas:** D'Artagnan, Athos, Porthos, Aramis

2.2. Gráficas t-SNE

Posteriormente, se realizó la gráfica para cada obra, aumentando el tamaño de embeddings para identificar las diferencias entre cada embedding y también identificar patrones similares en la escritura de cada autor.

Nota: Tomamos para el entrenamiento los modelos de 50,100 y 200 embeddings, sin embargo, queriamos apreciar las diferencias con embeddings chiquitos y embeddings con mucho contexto, como el de 5000, para poder observar que tan diferente se agrupaban terminos.

2.2.1. A Tale of Two Cities - Charles Dickens

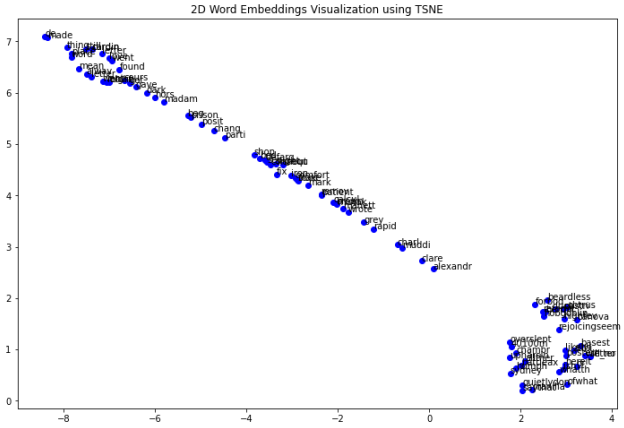

Figura 2.1: Gráfica con 50 embeddings

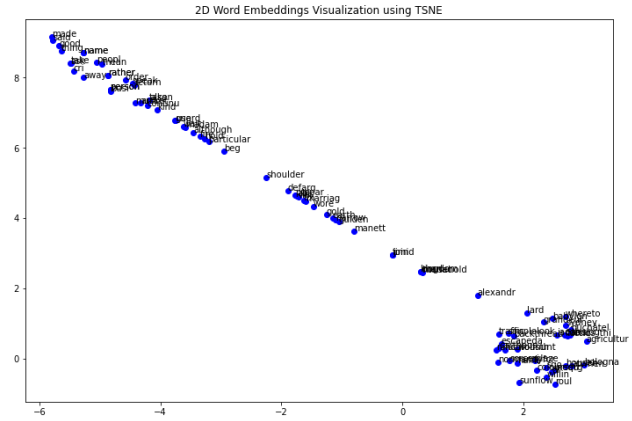

Figura 2.2: Gráfica con 100 embeddings

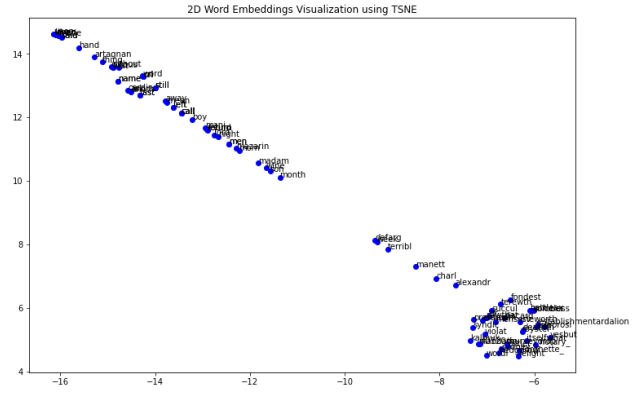

Figura 2.3: Gráfica con 1000 embeddings

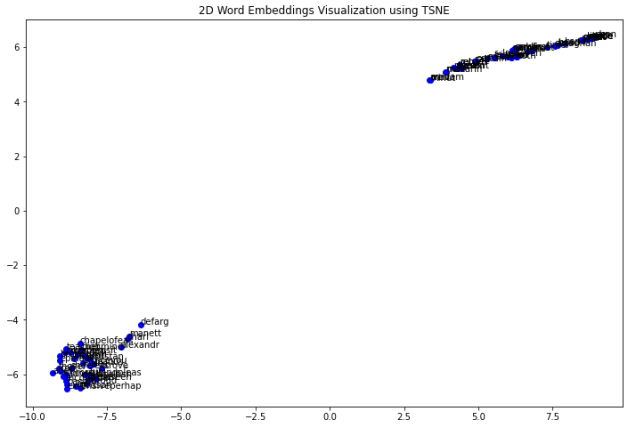

Figura 2.4: Gráfica con 5000 embeddings

2.2.2. Bleak House - Charles Dickens

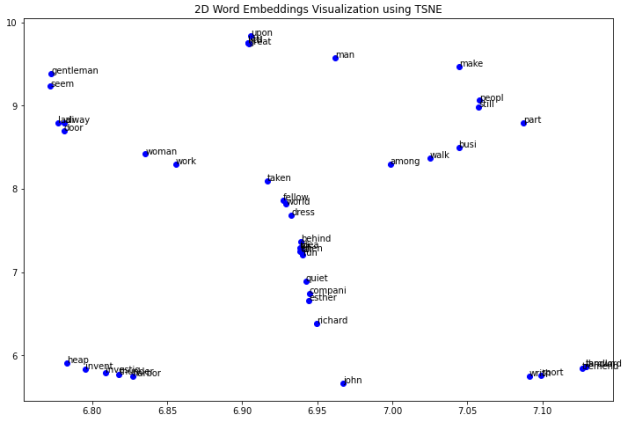

Figura 2.5: Gráfica con 50 embeddings

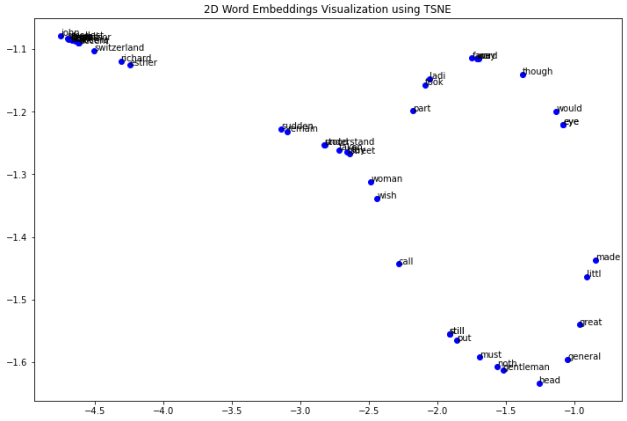

Figura 2.6: Gráfica con 100 embeddings

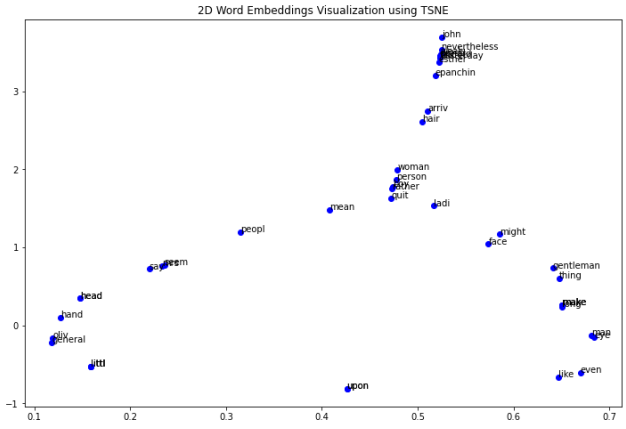

Figura 2.7: Gráfica con 1000 embeddings

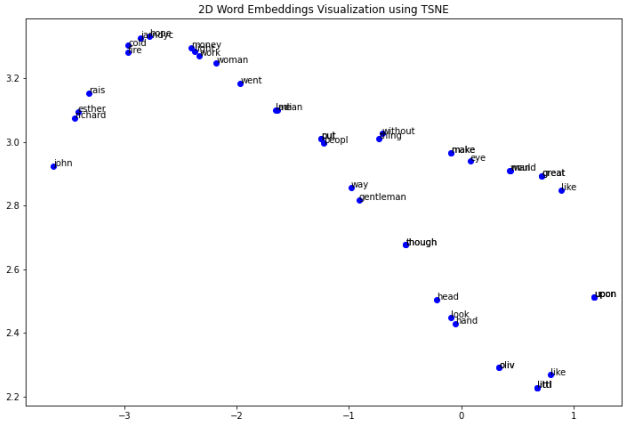

Figura 2.8: Gráfica con 5000 embeddings

2.2.3. Oliver Twist - Charles Dickens

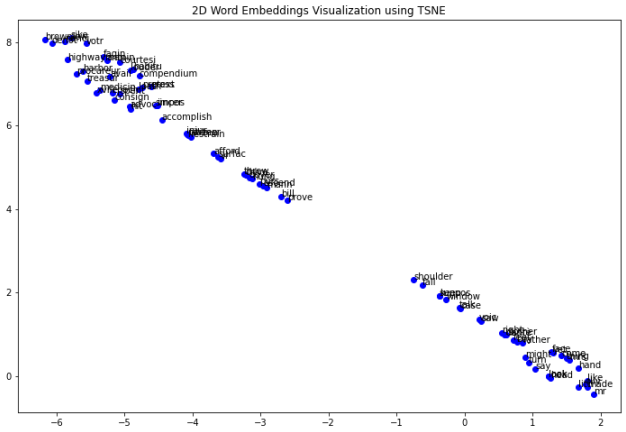

Figura 2.9: Gráfica con 50 embeddings

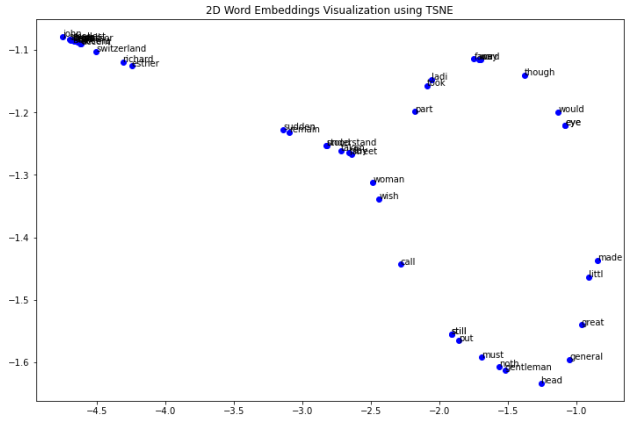

Figura 2.10: Gráfica con 100 embeddings

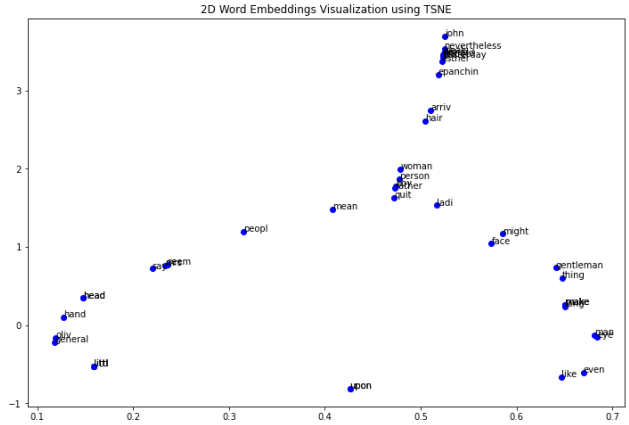

Figura 2.11: Gráfica con 1000 embeddings

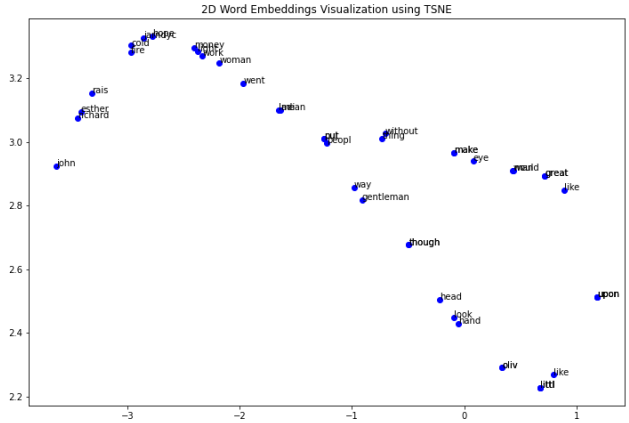

Figura 2.12: Gráfica con 5000 embeddings

2.2.4. Crime and Punishment - Fyodor Dostoyevski

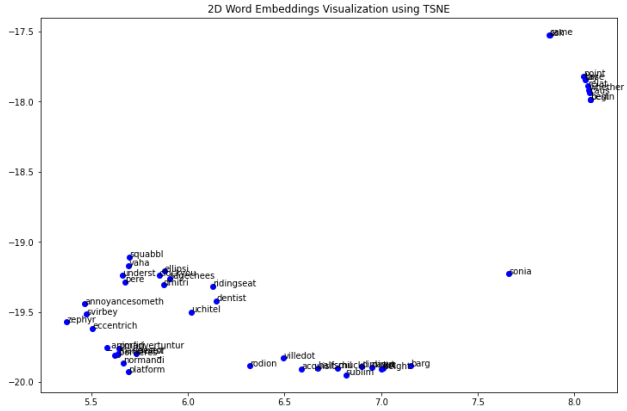

Figura 2.13: Gráfica con 50 embeddings

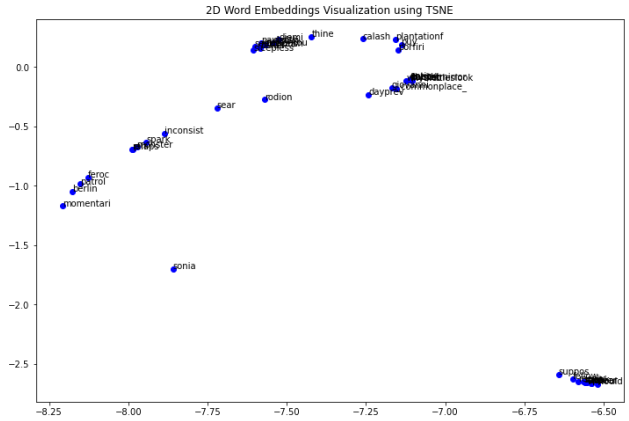

Figura 2.14: Gráfica con 1000 embeddings

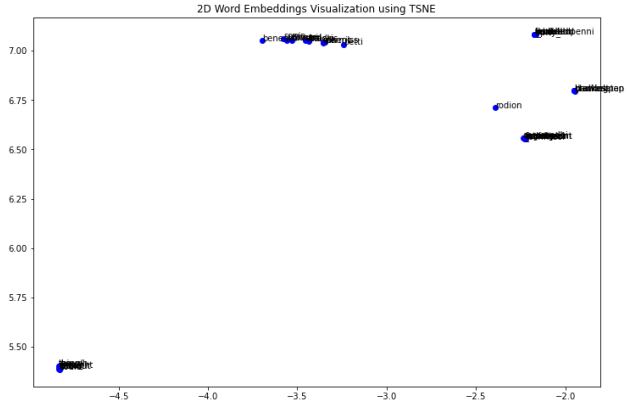

Figura 2.15: Gráfica con 1000 embeddings

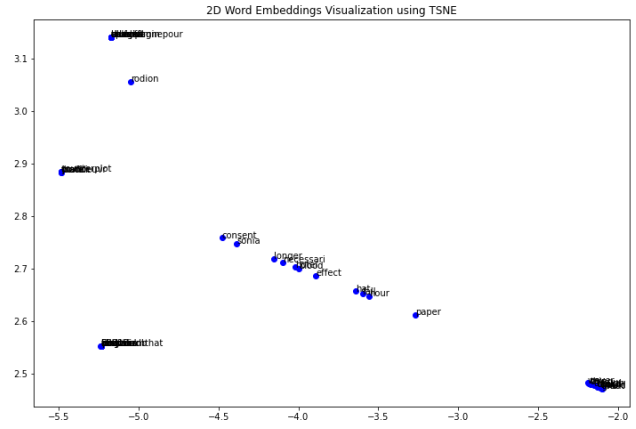

Figura 2.16: Gráfica con 5000 embeddings

2.2.5. The Gambler - Fyodor Dostoyevski

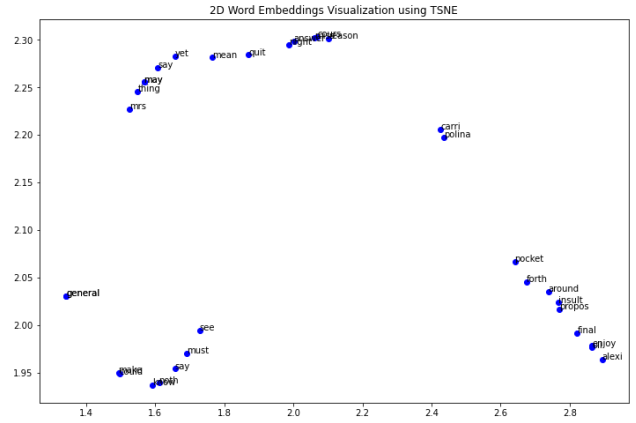

Figura 2.17: Gráfica con 50 embeddings

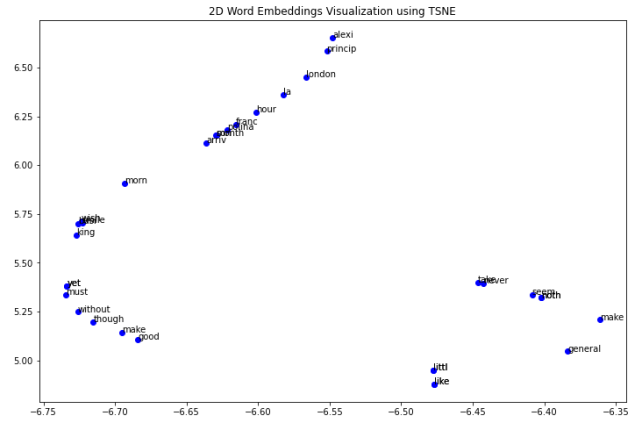

Figura 2.18: Gráfica con 100 embeddings

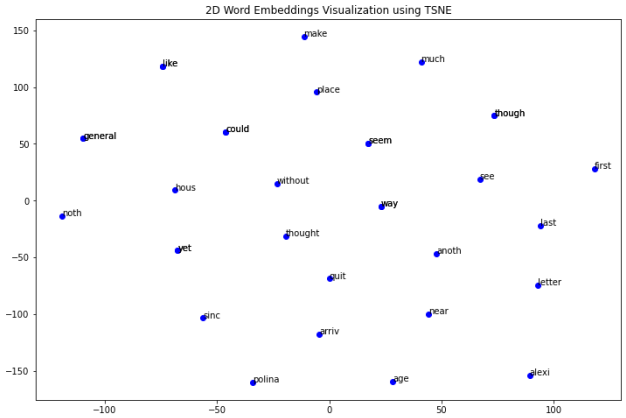

Figura 2.19: Gráfica con 1000 embeddings

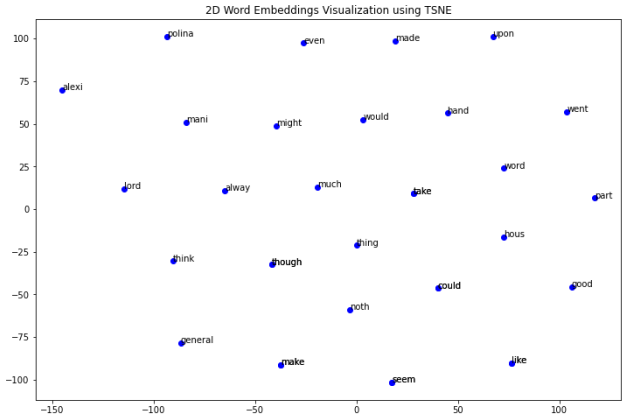

Figura 2.20: Gráfica con 5000 embeddings

2.2.6. The Idiot - Fyodor Dostoyevski

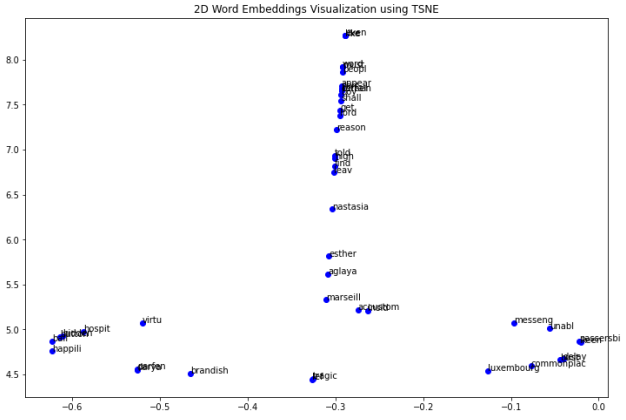

Figura 2.21: Gráfica con 50 embeddings

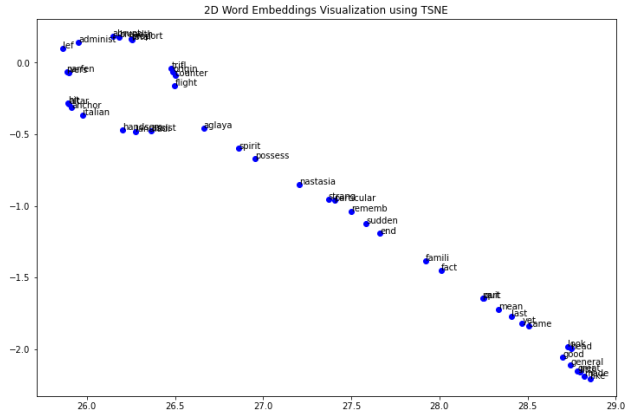

Figura 2.22: Gráfica con 100 embeddings

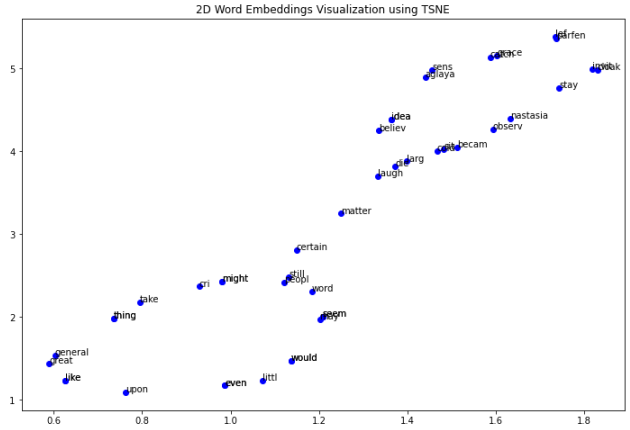

Figura 2.23: Gráfica con 1000 embeddings

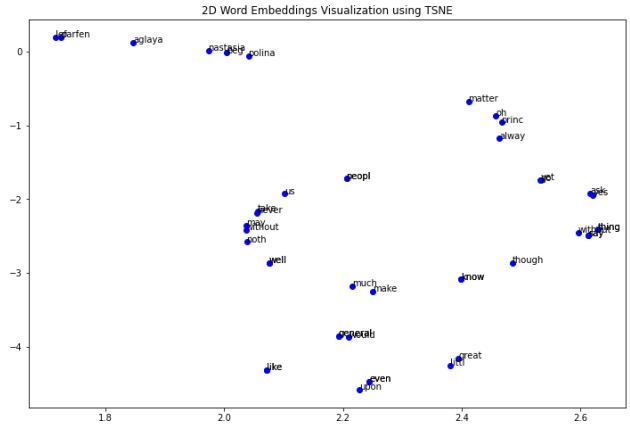

Figura 2.24: Gráfica con 5000 embeddings

2.2.7. The Count of Monte Cristo - Alexandre Dumas

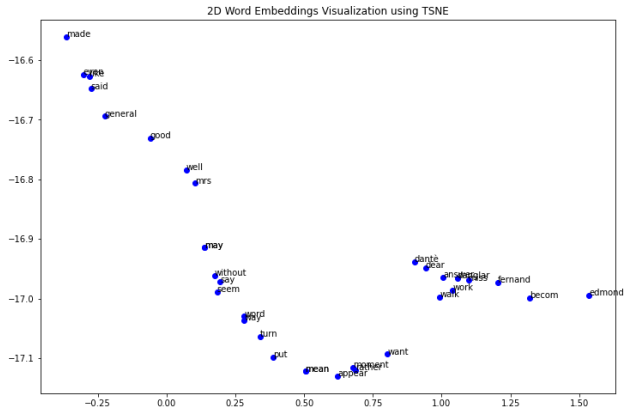

Figura 2.25: Gráfica con 50 embeddings

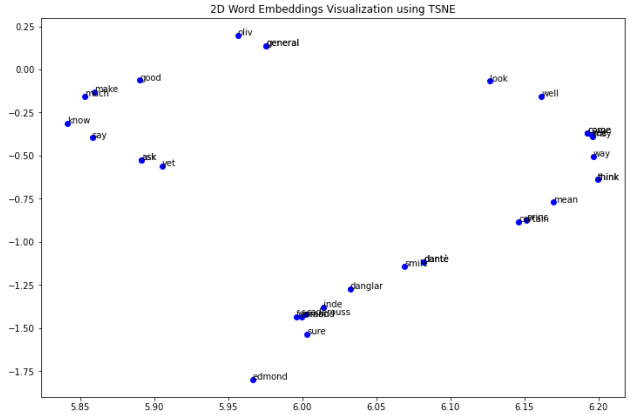

Figura 2.26: Gráfica con 100 embeddings

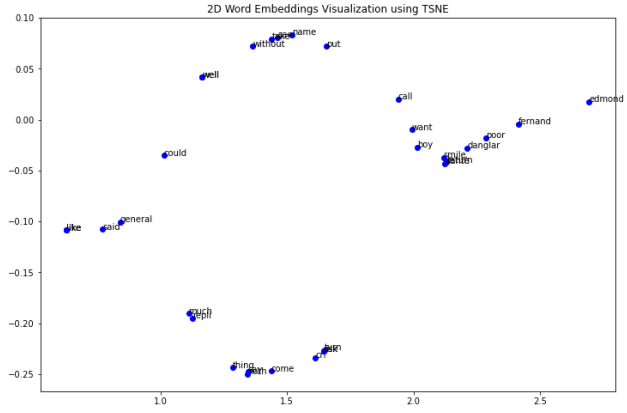

Figura 2.27: Gráfica con 1000 embeddings

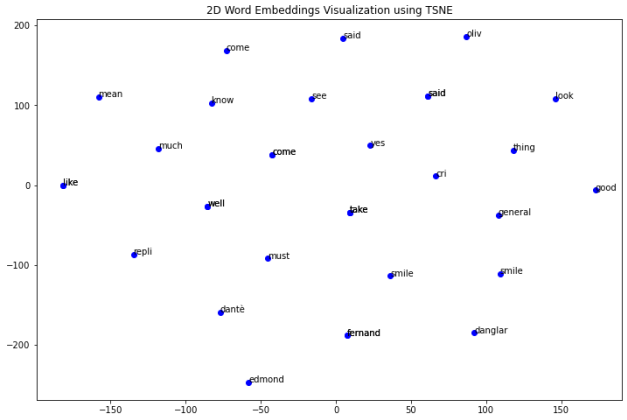

Figura 2.28: Gráfica con 5000 embeddings

2.2.8. The Three Musketeers - Alexandre Dumas

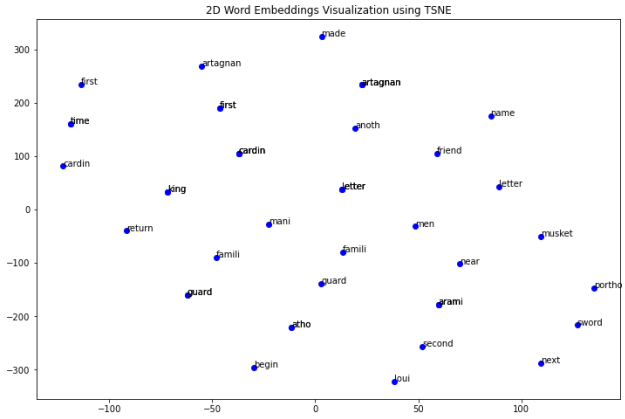

Figura 2.29: Gráfica con 50 embeddings

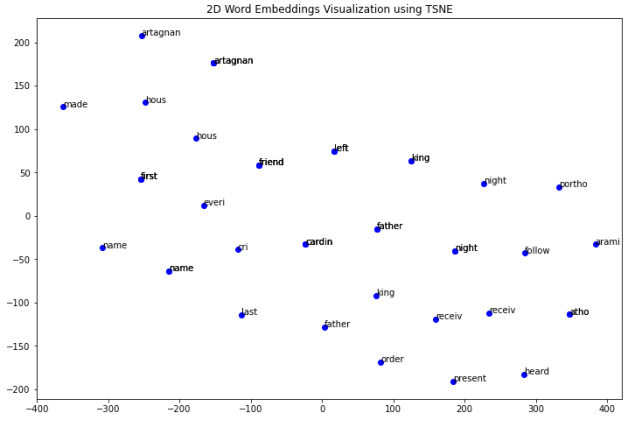

Figura 2.30: Gráfica con 100 embeddings

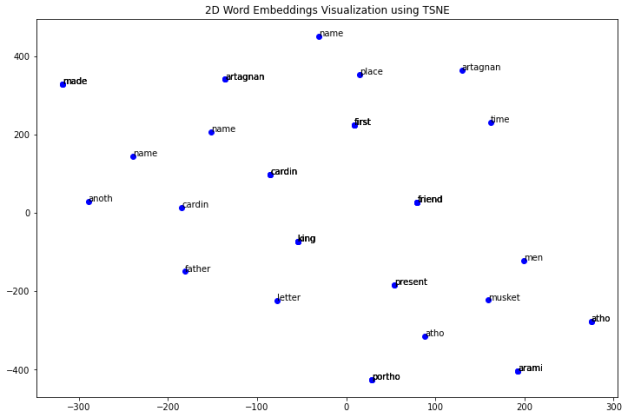

Figura 2.31: Gráfica con 1000 embeddings

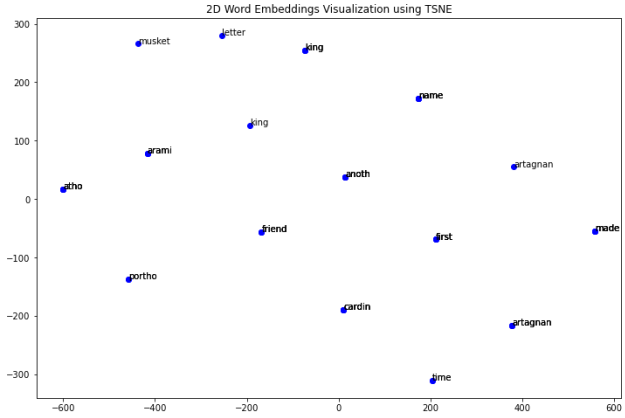

Figura 2.32: Gráfica con 5000 embeddings

2.2.9. Twenty Years After - Alexandre Dumas

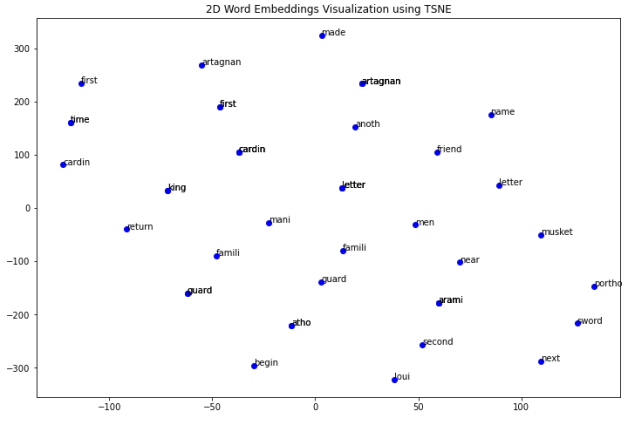

Figura 2.33: Gráfica con 50 embeddings

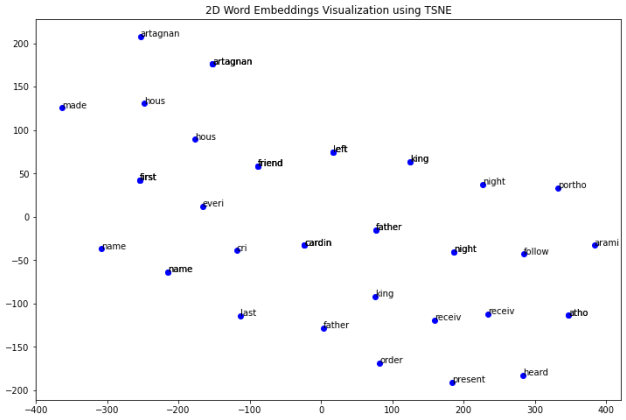

Figura 2.34: Gráfica con 100 embeddings

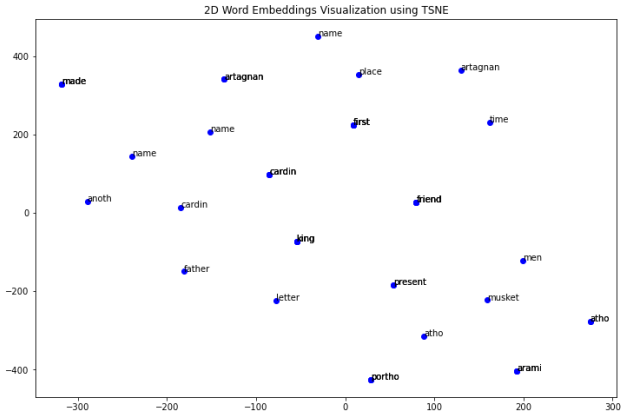

Figura 2.35: Gráfica con 1000 embeddings

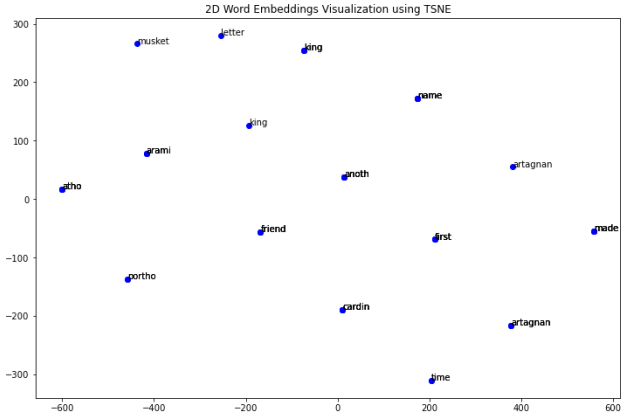

Figura 2.36: Gráfica con 5000 embeddings

2.3. Análisis de los Embeddings

Con respecto al cambio de los embeddings, se puede observar que a medida de que hay un aumento en la cantidad de embeddings para analizar el texto, mayor es la distancia y la densidad de los grupos y menor será la posibilidad de identificar patrones de escritura de cada autor. Evidenciado en 5000 embeddings una representación muy

diferente en términos de agrupación y distancia con respecto a una cantidad de embeddings más baja.

Para métodos prácticos, se utilizará el número de 100 embeddings para realizar el análisis y la identificación de analogías en la escritura de cada autor, las palabras que se utilizan alrededor del protagonistas no fueron bien detectadas dado que es evidente que se realizó la técnica de stemming para realizar el procesamiento de texto.

2.4. Análisis de los Protagonistas en las Obras

Según las gráficas, se puede observar que Charles Dickens tiene una forma muy similar para detallar a sus protagonistas en muchas de sus obras, exaltándolos con características al mencionarlos, esto se puede ver específicamnte en A Tale of Two Cities y en Oliver Twist, en los cuales el comportamiento en la gráfica tNSE es muy similar.

En las gráficas también se puede observar que Dostoyevski suele utilizar un estilo muy variado al describir a los protagonistas de sus historias, esto se ve reflejado en un comportamiento muy diferente y segmentado en sus gráficas.

Por último, se puede observar que Alexandre Dumas no suele ser muy descriptivo a la hora de mencionar los protagonistas en sus obras, ya que no existen agrupaciones tan claras entre palabras, esto sucede principalmente en The Three Musketeers y Twenty Years After, lo cual, este tipo de análisis tiene mucho sentido dado que un libro es la secuela directa de otra, por lo cual maneja los mismos protagonistas y la misma escritura.

Clasificación de Autores con Redes Neuronales

3.1. Extracción y preprocesamiento de textos

```
# Directorio donde est n los archivos .txt
data_dir = 'data/books/'
# Lista para almacenar los textos y autores
texts = []
authors = []
# Leer todos los archivos .txt del directorio
for filename in os.listdir(data_dir):
    if filename.endswith('.txt'):
        with open(os.path.join(data_dir, filename), 'r', encoding='utf-8') as f:
            text = f.read()
            texts.append(text)
            print(f"Procesando archivo: {filename}")
            author_match = re.search(r'Author:\s*(.+)', text)
            if author_match:
                author_name = author_match.group(1).strip()
                authors.append(author_name.lower())
            else:
                authors.append("Autor no encontrado")
```

Se guardan los textos de los libros en una lista `texts` y se busca el autor usando una expresión regular. Si no encuentra el autor, guarda .ªutor no encontrado".

3.2. Preprocesamiento de datos

```
def process_all_texts(processor = Processor(), texts: list = []):
    total = len(texts)
    processed_texts = []
    for index, text in enumerate(texts):
        processed_text = processor.preprocessing_pipeline_as_chunks(text)
        processed_texts.append(processed_text)
    return processed_texts

# Preprocesamiento de los textos
processed_texts = process_all_texts(processor_, texts)
```

Entrada: processor (una instancia de la clase `processor` ya definida) y texts (la lista de los textos que vamos a realizar el preprocesamiento).

Preparamos los datos que tenemos (el contenido de los libros) usando la función `preprocessing_pipeline_as_chunks()` que tenemos definida en `processor.py`, la cual realiza el siguiente preprocesamiento:

- **Gutenberg licensing removal**: para limpiar los datos de los avisos legales y los detalles de edición, guardando solo el contenido del libro.
- **Divide to chunks**: Divide el texto en fragmentos de un tamaño específico, predeterminado en 150.
- **Tokenization**

- **Lowercase conversion**: Convierte todas las palabras de la lista a minúsculas.
- **Punctuation removal**: Elimina los signos de puntuación de la lista de palabras.
- **Word-to-number conversion**: Convierte palabras que representan números en su forma numérica.
- **Digit replacement**: Reemplaza los números con la cadena de marcador de posición 'NUM'.
- **Stopword removal**: Elimina los *stopwords* de la lista de palabras utilizando el corpus de *stopwords* de NLTK.
- **Stemming**

Salida: retorna una nueva lista de los textos procesados.

3.3. Conjunto de entrenamiento, validación y prueba

Los textos fueron leídos desde un directorio que contenía múltiples archivos en formato `.txt`. Cada archivo representa una obra de un autor, y el nombre del autor fue extraído del contenido del archivo mediante expresiones regulares que buscaban el patrón `Author:`. Si no se encontraba el autor en el archivo, se asignaba el valor de "Autor no encontrado", sin embargo, este no fue el caso, solo fue una precaución tomada. Los textos fueron divididos en fragmentos de 150 palabras con un solapamiento de 25 palabras para preservar el contexto entre los fragmentos. Para no perder información clave de la idea principal.

Una vez procesados los textos y extraídos los autores correspondientes, se construyó un *DataFrame* utilizando la biblioteca `pandas`. En este *DataFrame*, cada fila representa un fragmento de texto (*chunk*) y el autor al que pertenece. Esta estructura fue la base para la posterior partición en conjuntos de entrenamiento, validación y prueba.

3.3.1. División del Dataset

El *dataset* completo fue dividido en tres subconjuntos: entrenamiento, validación y prueba, utilizando una proporción de 70 % para entrenamiento, 20 % para prueba y 10 % para validación. Se utilizó la función `train_test_split` de `scikit-learn` para realizar esta partición de manera estratificada, garantizando que cada autor estuviera representado en las tres particiones de acuerdo con su proporción en el conjunto de datos original.
El proceso se puede resumir en los siguientes pasos:

1. Se dividió el *DataFrame* original en un conjunto de entrenamiento (70 %) y un conjunto de prueba (30 %), estratificando por autor para mantener la distribución de clases.
2. Posteriormente, el conjunto de entrenamiento se dividió de nuevo en un conjunto de entrenamiento (90 %) y un conjunto de validación (10 %), manteniendo la estratificación.

```python
def summary_by_author(train_df, validation_df, test_df):
    """
    Generates a summary table showing the number of samples per author for the training,
        validation, and testing sets.

    Args:
        train_df (pd.DataFrame): Training DataFrame.
        validation_df (pd.DataFrame): Validation DataFrame.
        test_df (pd.DataFrame): Testing DataFrame.

    Returns:
        pd.DataFrame: A summary DataFrame.
    """

    summary_data = {
        'Author': train_df['author'].unique(),
        'Train': train_df['author'].value_counts(),
        'Validation': validation_df['author'].value_counts(),
        'Test': test_df['author'].value_counts()
    }
```

```
    summary_df = pd.DataFrame(summary_data)
    summary_df = summary_df.fillna(0)  # Replace NaN with 0 if no samples exist for some
        authors

    return summary_df
```

Entrada: los conjuntos de entrenamiento, validación y prueba.
Se usa la función `value_counts()` para calcular las muestras en cada conjunto.
Salida: un dataframe que representa el número de las muestras de cada autor por conjunto.
Este es el resultado:

Author	Train	Validation	Test
charles dickens	4744	527	2260
alexandre dumas	3307	368	1575
fyodor dostoyevsky	2592	288	1234

Tabla 3.1:

Como podemos observar en la tabla anterior, las proporciones de los autores en los tres conjuntos son relativamente similares: *Alexandre Dumas* representa aproximadamente un 44 % del total, seguido de *Charles Dickens* con un 31 %, y finalmente *Fyodor Dostoyevsky* con un 25 %. Aunque esta distribución no es perfectamente balanceada, las diferencias no son lo suficientemente significativas como para afectar gravemente el rendimiento del modelo de clasificación, y en caso de eliminar oraciones podriamos estar eliminando factores clave en la diferenciacion entre autores.

Es importante mencionar que un dataset balanceado, donde cada clase (autor, en este caso) tiene una representación equitativa, es ideal para la mayoría de los modelos de clasificación. Sin embargo, dado que en este caso la diferencia entre los autores más representados y menos representados es menor al 20 %, no fue necesario realizar un rebalanceo inmediato de los datos. La ligera diferencia en los porcentajes se mantiene dentro de un margen aceptable teniendo en cuenta el trade-off que podíamos perder con la eliminación de oraciones.

3.4. Creación de Embeddings y Redes Neuronales

En este proyecto, uno de los componentes clave fue la representación de los textos a través de embeddings preentrenados utilizando el modelo Word2Vec. Estos embeddings proporcionan una representación vectorial de las palabras, capturando relaciones semánticas entre ellas, y permiten a la red neuronal aprender patrones útiles para la clasificación. Sin embargo, debido a la naturaleza compleja de la tarea, los resultados obtenidos no fueron los esperados. A continuación, se detallan los pasos seguidos en la creación de los embeddings y las redes neuronales, así como el análisis de los resultados obtenidos.

3.4.1. Creación de la Matriz de Embeddings

Para convertir los fragmentos de texto en representaciones numéricas, se utilizó un modelo preentrenado de *Word2Vec*. La matriz de embeddings se generó mapeando cada palabra del vocabulario a su correspondiente vector en el espacio de embeddings. El tamaño del embedding (*embedding size*) fue de 200 dimensiones, lo cual es estándar para este tipo de tareas. Posteriormente, se preprocesaron los textos del conjunto de datos, tokenizándolos y convirtiéndolos en secuencias de índices correspondientes a las palabras en el vocabulario del modelo.

- Los textos se transformaron en secuencias de tokens, cada uno de los cuales se mapeó a un índice en el vocabulario del modelo Word2Vec.
- Las secuencias se rellenaron (*padding*) para asegurar que todas tuvieran la misma longitud máxima, basada en el texto más largo del conjunto de datos.
- Se codificaron las etiquetas de los autores utilizando un *LabelEncoder* para transformar los nombres de los autores en índices numéricos, lo que permitió que el modelo tratara esta tarea como un problema de clasificación multicategoría.

3.4.2. Redes Neuronales Implementadas

Para abordar el problema de clasificación de autores, se diseñaron y entrenaron tres redes neuronales con distintos niveles de complejidad. A continuación, se detalla cada una de ellas:

RDD1: Red Neuronal Densa

La primera arquitectura (`RDD1`) consistió en una red neuronal completamente densa. Esta red fue la más sencilla de las tres y utilizó una capa de *embedding* no entrenable con los pesos preentrenados de Word2Vec. Después de la capa de *embedding*, los vectores de palabras fueron aplanados (*Flatten*) y pasaron por tres capas densas (*fully connected*) con activaciones *ReLU*. Estas capas densas fueron intercaladas con capas de *dropout* para reducir el riesgo de sobreajuste. Finalmente, se añadió una capa *softmax* con tres unidades para la clasificación de los autores.

- Esta red fue relativamente simple y mostró limitaciones en la captura de patrones complejos en los datos, lo cual se reflejó en un rendimiento bajo en las métricas de precisión y exactitud.

RDD2: Red Neuronal Densa con Capas Densas Profundas

La segunda arquitectura (`RDD2`) es una red neuronal que comienza con una capa de *embedding* no entrenable, utilizando un vocabulario de tamaño 27,220 y un embedding preentrenado de dimensión 100. A diferencia de `RDD1`, `RDD2` introduce una capa `Dense` inicial de 512 neuronas después de la capa de `Flatten`, con una activación `ReLU`. La arquitectura de `RDD2` incluye las siguientes capas:

- Una capa de `Embedding` con una salida de forma `(None, 121, 100)`.
- Una capa de `Flatten` que convierte la salida a un vector unidimensional de tamaño `(None, 12100)`.
- Una primera capa densa (`Dense`) con 512 unidades y activación `ReLU`.
- Una capa de `Dropout` con una tasa de 0.4 para mitigar el sobreajuste.
- Una segunda capa `Dense` con 256 unidades y activación `ReLU`, seguida de otra capa de `Dropout` con una tasa de 0.4.
- Una tercera capa `Dense` con 128 unidades y activación `ReLU`, seguida de una capa de `Dropout` con una tasa de 0.3.
- Finalmente, una cuarta capa `Dense` con 64 unidades y activación `ReLU`, antes de llegar a la capa de salida `Dense` con activación `Softmax` para clasificar entre tres autores.

Este diseño de `RDD2` busca mejorar la capacidad del modelo para capturar relaciones complejas entre las palabras a través de un ajuste más refinado de los pesos. Aunque no incorpora mecanismos de atención ni capas recurrentes, el uso de múltiples capas densas profundas permite mejorar la capacidad de representación del modelo.

RDD3: Red Neuronal con LSTM y Atención

La tercera red (`RDD3`) fue la arquitectura más compleja implementada. Además de las capas *LSTM* bidireccionales, esta red incluyó un mecanismo de atención multi-cabezal (*multi-head attention*) para capturar dependencias de largo alcance entre las palabras del texto. También se añadieron conexiones residuales entre las capas *LSTM* para evitar el problema de los gradientes desvanecientes. Esta red fue diseñada para abordar la complejidad del problema mediante un procesamiento profundo de las secuencias, utilizando técnicas avanzadas de regularización como *spatial dropout*, normalización de capas (*layer normalization*) y múltiples capas densas.

- A pesar de la complejidad de esta red y del uso de mecanismos avanzados como la atención, los resultados obtenidos no fueron significativamente mejores que los de las redes anteriores. Esto sugiere que los embeddings preentrenados de Word2Vec no contenían la suficiente información discriminativa para la correcta clasificación de los autores, posiblemente debido a la naturaleza similar de los textos o a las características estilísticas complejas que no pudieron ser capturadas con este enfoque.

3.4.3. Análisis de Resultados

A lo largo de los experimentos, se probaron varias arquitecturas de redes neuronales, aumentando la complejidad progresivamente. Sin embargo, los resultados no fueron los esperados. Las métricas de precisión, exactitud y recall indicaron que la clasificación de autores basada en los embeddings preentrenados no proporcionaba suficiente información para una clasificación precisa. Esto sugiere que el problema es inherentemente difícil, o que los embeddings utilizados no eran adecuados para capturar las características distintivas de los textos.
Al analizar el rendimiento de cada red, se observó que incluso con la implementación de arquitecturas avanzadas, como el modelo `RDD3` que utilizaba mecanismos de atención y conexiones residuales, los resultados no mejoraron significativamente. Esto refuerza la hipótesis de que el problema radica en los embeddings o en la naturaleza compleja del estilo de los autores, lo que hace que la clasificación sea difícil con las técnicas utilizadas.
Como conclusión, se podría considerar explorar otros enfoques para mejorar la clasificación, como la utilización de modelos basados en *transformers* o el entrenamiento de embeddings más especializados en este dominio, para capturar mejor las características únicas de cada autor.

3.5. Resultados de las Arquitecturas con Embedding 100

En esta sección, se explican las dimensiones de cada capa de las tres arquitecturas evaluadas (RDD1, RDD2 y RDD3) utilizando un embedding de dimensión 100. Las arquitecturas presentan diferencias significativas en su estructura y complejidad.

3.5.1. RDD1

RDD1 comienza con una capa de `Embedding`, que tiene un vocabulario de 27,220 palabras y genera un embedding de dimensión 100. La salida de la capa de `Embedding` tiene la forma `(None, 121, 100)`, donde 121 es la longitud máxima de las secuencias de entrada y 100 es la dimensión de los vectores de palabras.
Posteriormente, se aplica una capa de `Flatten`, que convierte la salida en un vector de tamaño `(None, 12100)`. Después de la capa `Flatten`, la arquitectura incluye varias capas densas (`Dense`) intercaladas con capas de `Dropout`. Los resúmenes del modelo no especifican las dimensiones exactas de estas capas densas, lo que se indica con `0 (unbuilt)` en los parámetros del modelo.
El número total de parámetros en esta arquitectura es 2,722,000, los cuales no son entrenables debido a la naturaleza preentrenada del embedding.

3.5.2. RDD2

La arquitectura **RDD2**, al igual que RDD1, comienza con una capa de `Embedding` que genera una salida de forma `(None, 121, 100)` y contiene 2,722,000 parámetros no entrenables debido al embedding preentrenado.
Después de la capa `Flatten`, que convierte la salida en un vector de tamaño `(None, 12100)`, RDD2 difiere de RDD1 al incluir una capa densa (`Dense`) de 512 unidades con activación `ReLU`, lo que proporciona una mayor capacidad de procesamiento inicial. Luego, se añade una capa de `Dropout` con un valor de 0.4 para prevenir el sobreajuste. A esta capa le sigue una segunda capa `Dense` de 256 unidades, también intercalada con `Dropout` de 0.4, y una tercera capa `Dense` de 128 unidades, con `Dropout` de 0.3. Finalmente, se añade una capa `Dense` de 64 unidades antes de la capa de salida.
La arquitectura termina con una capa de salida `Dense` de 3 unidades (correspondientes a las clases de autores), utilizando la función de activación `softmax`. Esta estructura permite un mayor grado de procesamiento en comparación con RDD1, gracias a la mayor capacidad de sus capas densas.

3.5.3. RDD3

RDD3 es considerablemente más compleja que RDD1 y RDD2. Comienza también con una capa de `Embedding` de dimensión 100, pero a esta se le aplica una capa de `SpatialDropout1D`, manteniendo la salida con forma `(None, 121, 100)`. Luego, se utiliza una capa `Bidirectional` que genera una salida de `(None, 121, 512)`, lo que indica que la capa tiene 512 unidades bidireccionales.
A esta capa se le aplica una normalización y se introduce una `MultiHeadAttention`, lo que mantiene la forma `(None, 121, 512)`. Finalmente, se pasa por varias capas `Bidirectional`, seguidas de una concatenación de resultados de capas de `GlobalAveragePooling`, lo que resulta en una forma `(None, 896)` antes de entrar a las capas densas finales.

La arquitectura tiene un total de 5,429,715 parámetros, de los cuales 2,705,923 son entrenables y 2,723,792 no entrenables (relacionados con el embedding).

3.6. Resultados Embedding 50

Se evaluaron tres arquitecturas (RDD1, RDD2, y RDD3) con un embedding de dimensión 100. Los resultados obtenidos en términos de *accuracy*, *precision* y *recall* se muestran en la siguiente tabla:

Modelo	Test Accuracy	Test Precision	Test Recall
RDD1	0.4458	0.1988	0.4458
RDD2	0.4458	0.1988	0.4458
RDD3	**0.4528**	**0.3245**	**0.4528**

Tabla 3.2: Resultados de las arquitecturas con embedding de dimensión 100.

3.7. Explicación de las Dimensiones de las Arquitecturas - Embedding 200

En esta sección, se explican las dimensiones de cada capa de las tres arquitecturas evaluadas (RDD1, RDD2 y RDD3) utilizando un embedding de dimensión 200. Las arquitecturas tienen diferencias importantes en la estructura y complejidad.

3.7.1. RDD1

RDD1 comienza con una capa de `Embedding`, que tiene un vocabulario de 27,220 palabras y genera un embedding de dimensión 200. La salida de la capa de `Embedding` tiene la forma `(None, 121, 200)`, donde 121 es la longitud máxima de las secuencias de entrada y 200 es la dimensión de los vectores de palabras.
Posteriormente, se aplica una capa de `Flatten`, que convierte la salida en un vector unidimensional de tamaño `(None, 24200)`. Luego, la arquitectura incluye tres capas `Dense` intercaladas con capas de `Dropout`. Los resúmenes del modelo no especifican las dimensiones exactas de estas capas densas, lo que se indica con `0 (unbuilt)` en los parámetros del modelo.
El número total de parámetros en esta arquitectura es 5,444,000, los cuales no son entrenables debido a la naturaleza preentrenada del embedding.

3.7.2. RDD2

La arquitectura **RDD2**, al igual que RDD1, comienza con una capa de `Embedding` que genera una salida de forma `(None, 121, 200)` y con un total de 5,444,000 parámetros no entrenables debido a la naturaleza del embedding. Después de la capa `Flatten`, que convierte la salida en un vector de tamaño `(None, 24200)`, se añade una capa `Dense` con 512 unidades y activación `ReLU`. Esta capa permite realizar un procesamiento más profundo de los datos, seguida por una capa de `Dropout` con un valor de 0.4 para prevenir el sobreajuste. A continuación, se agregan capas `Dense` adicionales con 256, 128 y 64 unidades, cada una intercalada con capas `Dropout` de 0.4 y 0.3, lo que refuerza la capacidad del modelo para aprender características representativas sin sobreajuste.
Finalmente, el modelo termina con una capa de salida `Dense` con 3 neuronas, correspondientes a las clases de autores. El número total de parámetros en RDD2 es mayor que en RDD1 debido a las capas densas adicionales, lo que le otorga una mayor capacidad de procesamiento y generalización.

3.7.3. RDD3

RDD3 es considerablemente más compleja que RDD1 y RDD2. Comienza también con una capa de `Embedding` de dimensión 200, pero a esta se le aplica una capa de `SpatialDropout1D`, manteniendo la salida con forma `(None, 121, 200)`. Luego, se utiliza una capa `Bidirectional` que genera una salida de `(None, 121, 512)`, lo que indica que la capa tiene 512 unidades bidireccionales.

A esta capa se le aplica una normalización y se introduce una `MultiHeadAttention`, lo que mantiene la forma `(None, 121, 512)`. Finalmente, se pasa por varias capas `Bidirectional`, seguidas de una concatenación de resultados de capas de `GlobalAveragePooling`, lo que resulta en una forma `(None, 896)` antes de entrar a las capas densas finales.
La arquitectura tiene un total de 8,356,515 parámetros, de los cuales 2,910,723 son entrenables y 5,445,792 no entrenables (relacionados con el embedding).

3.8. Resultados de las Arquitecturas con Embedding 200

A continuación se presentan los resultados de las tres arquitecturas utilizando embeddings de dimensión 200, evaluadas en términos de *accuracy*, *precision* y *recall*.

Modelo	Test Accuracy	Test Precision	Test Recall
RDD1	0.4458	0.1988	0.4458
RDD2	0.4458	0.1988	0.4458
RDD3	**0.4553**	**0.4213**	**0.4553**

Tabla 3.3: Resultados de las arquitecturas con embedding de dimensión 200.

3.9. Explicación de las Dimensiones de las Arquitecturas - Embedding 50

En esta sección, se explican las dimensiones de cada capa de las tres arquitecturas evaluadas (RDD1, RDD2 y RDD3) utilizando un embedding de dimensión 50.

3.9.1. RDD1

La arquitectura **RDD1** comienza con una capa de `Embedding`, que tiene un vocabulario de 27,220 palabras y genera un embedding de dimensión 50. La salida de la capa de `Embedding` tiene la forma `(None, 121, 50)`, donde 121 es la longitud máxima de las secuencias de entrada y 50 es la dimensión de los vectores de palabras.
Posteriormente, se aplica una capa de `Flatten`, que convierte la salida en un vector unidimensional de tamaño `(None, 6050)`. Luego, la arquitectura incluye tres capas `Dense` intercaladas con capas de `Dropout`. Los resúmenes del modelo no especifican las dimensiones exactas de estas capas densas, lo que se indica con `0 (unbuilt)` en los parámetros del modelo.
El número total de parámetros en esta arquitectura es 1,361,000, los cuales no son entrenables debido a la naturaleza preentrenada del embedding.

3.9.2. RDD2

La arquitectura **RDD2** sigue una estructura similar en las capas iniciales a RDD1, comenzando con una capa de `Embedding` que produce una salida de forma `(None, 121, 50)` y con un total de 1,361,000 parámetros no entrenables.
Sin embargo, la principal diferencia radica en las capas densas. Después de la capa `Flatten` que convierte la salida en un vector de tamaño `(None, 6050)`, se añade una capa `Dense` con 512 unidades y activación `ReLU`, seguida por una capa de `Dropout` con un valor de 0.4. Luego, se agregan capas `Dense` adicionales con 256, 128 y 64 unidades, intercaladas con capas `Dropout` con valores de 0.4 y 0.3, respectivamente.
Finalmente, el modelo termina con una capa de salida `Dense` con 3 neuronas para la clasificación de autores. Esta arquitectura tiene un total considerablemente mayor de parámetros en comparación con RDD1 debido a la adición de capas densas con muchas unidades.

3.9.3. RDD3

La arquitectura **RDD3** es más compleja que RDD1 y RDD2. Al igual que en los otros modelos, comienza con una capa de `Embedding` que genera una salida de forma `(None, 121, 50)`. Esta capa tiene el mismo número de parámetros no entrenables que en los otros modelos.

Después de la capa de `Embedding`, RDD3 incorpora una capa de `SpatialDropout1D` que mantiene la misma forma de salida. Luego, se aplica una capa `Bidirectional LSTM` que expande la salida a `(None, 121, 512)`. A continuación, se emplea una capa de `MultiHeadAttention`, que también tiene una salida de `(None, 121, 512)`. Estas salidas pasan por varias capas adicionales de normalización y `GlobalAveragePooling1D`, las cuales convierten las salidas en vectores de tamaños `(None, 512)`, `(None, 256)` y `(None, 128)`, respectivamente.
La arquitectura finaliza con una capa de `Concatenate`, que combina todas las salidas anteriores en un vector de tamaño `(None, 896)`. El modelo luego incluye capas `Dense` con 512, 256, y 128 unidades, terminando con una capa de salida de 3 unidades para la clasificación.
El número total de parámetros en RDD3 es significativamente mayor que en RDD1 y RDD2, debido a la complejidad de las capas bidireccionales y de atención.

3.10. Resultados de las Arquitecturas con Embedding 50

Se evaluaron tres arquitecturas (RDD1, RDD2, y RDD3) con un embedding de dimensión 50. Los resultados obtenidos en términos de *accuracy*, *precision* y *recall* se muestran en la siguiente tabla:

Modelo	Test Accuracy	Test Precision	Test Recall
RDD1	0.4458	0.1988	0.4458
RDD2	0.4458	0.1988	0.4458
RDD3	**0.4553**	**0.4213**	**0.4553**

Tabla 3.4: Resultados de las arquitecturas con embedding de dimensión 50.

3.11. Conclusión

Tras evaluar las arquitecturas RDD1, RDD2 y RDD3 utilizando tres tipos de embeddings de diferentes dimensiones (50, 100 y 200), se pueden extraer las siguientes conclusiones:

- Las arquitecturas RDD1 y RDD2, que comparten una estructura similar, muestran resultados muy consistentes a lo largo de las tres configuraciones de embeddings. En todos los casos, la *accuracy* fue de 0.4458, con una *precision* de 0.1988 y un *recall* de 0.4458. Esto indica que estos modelos tienen una capacidad limitada para mejorar el rendimiento de clasificación, independientemente de la dimensión de los embeddings utilizados.

- La arquitectura RDD3, que es más compleja e incluye mecanismos de atención y capas bidireccionales, demostró ser significativamente más efectiva en comparación con RDD1 y RDD2. Con embeddings de mayor dimensión (50, 100 y 200), la *accuracy* de RDD3 aumentó ligeramente en cada caso, alcanzando un máximo de 0.4553 con embeddings de dimensión 50 y 200. El incremento en la *precision* fue más notable, alcanzando un valor de 0.4213 con embeddings de dimensión 50 y 200, y 0.3245 con embeddings de dimensión 100, lo que sugiere que RDD3 puede manejar mejor la complejidad del problema.

- En general, se observa que la arquitectura RDD3 responde mejor a embeddings de mayor dimensión, pero también es evidente que no hay una mejora significativa entre las dimensiones 50, 100 y 200 en términos de *accuracy*. Sin embargo, la *precision* en los embeddings de dimensión 50 y 200 fue consistentemente superior, lo que indica que la arquitectura RDD3 mejora en la clasificación de las clases relevantes con mayor certeza en estos casos.

La elección de una arquitectura más compleja como RDD3 ofrece una ventaja clara en cuanto a rendimiento, especialmente cuando se utilizan embeddings de mayor dimensión. No obstante, el aumento en la dimensión del embedding no siempre garantiza una mejora lineal en todas las métricas, lo que sugiere que la selección de la arquitectura puede ser más crucial que el tamaño del embedding en sí.

Uso de Embeddings Preentrenados

En este punto, implementamos y evaluamos el uso de embeddings preentrenados de GloVe (*Global Vectors for Word Representation*) para mejorar la clasificación de autores. Utilizamos embeddings GloVe de tres tamaños: 50, 100 y 200 dimensiones, y comparamos su rendimiento con los embeddings entrenados previamente con Word2Vec.

4.1. Carga de los embeddings preentrenados

Para realizar este análisis, cargamos los modelos preentrenados de GloVe utilizando la biblioteca `gensim.downloader`. Esto nos permite trabajar con embeddings ya entrenados en grandes corpus de texto, proporcionando una rica representación semántica de las palabras. Los embeddings GloVe de diferentes dimensiones se cargan de la siguiente manera:

```
import gensim.downloader as api

# Cargar los modelos preentrenados de GloVe
glove_models = {
    'glove_50': api.load('glove-wiki-gigaword-50'),   # 50 dimensiones
    'glove_100': api.load('glove-wiki-gigaword-100'),  # 100 dimensiones
    'glove_200': api.load('glove-wiki-gigaword-200'),  # 200 dimensiones
}
```

4.2. Preparación de la matriz de embeddings

El proceso de creación de la matriz de embeddings para los modelos de GloVe sigue los mismos pasos descritos previamente, donde se construyeron las matrices de embeddings para los modelos de Word2Vec. La función `create_embedding_matrix()` se adapta para trabajar con los modelos de GloVe, asegurando que las palabras de los textos procesados se mapeen correctamente a sus representaciones vectoriales.

4.3. Preparación de los datos

El preprocesamiento y la conversión de los textos en secuencias de índices siguen exactamente el mismo procedimiento descrito en el Punto 3 del informe, donde se utilizó la función `prepare_data()` y el relleno de secuencias (*padding*) con `pad_sequences()` para garantizar que todas las secuencias tengan la misma longitud. No es necesario repetir estos pasos en detalle aquí, pero se aplican de manera idéntica usando los embeddings GloVe en lugar de Word2Vec.

4.4. Entrenamiento y evaluación de los modelos

El entrenamiento de las arquitecturas de redes neuronales RDD1, RDD2 y RDD3, así como su evaluación, sigue los mismos principios explicados en el Punto 3, utilizando las métricas de precisión, *recall* y *accuracy* para medir el rendimiento del modelo en el conjunto de prueba. La única diferencia aquí es que las capas de embeddings ahora utilizan las representaciones preentrenadas de GloVe.
A continuación, presentamos el código utilizado para entrenar los modelos con los embeddings GloVe y realizar las evaluaciones correspondientes:

```
# Entrenar y evaluar los modelos
for glove_name, glove_model in glove_models.items():
    embedding_size = int(glove_name.split('_')[1])  # Extraer el tama o de los embeddings
    print(f"Training models with {glove_name} embeddings...")

    # Crear la matriz de embeddings y preparar datos
    embedding_matrix = create_embedding_matrix(
        train_df, val_df, test_df, glove_model, embedding_size
    )
    X_train_padded = pad_sequences(prepare_data(train_df, processor_, glove_model), maxlen=
        max_length, padding='post')
    X_val_padded = pad_sequences(prepare_data(val_df, processor_, glove_model), maxlen=
        max_length, padding='post')
    X_test_padded = pad_sequences(prepare_data(test_df, processor_, glove_model), maxlen=
        max_length, padding='post')

    # Entrenar y evaluar los modelos RDD1, RDD2, RDD3
    for model_name, model in models.items():
        history = train_model(
            model=model,
            train_data=(X_train_padded, y_train),
            val_data=(X_val_padded, y_val),
            batch_size=32,
            epochs=20
        )
        accuracy, precision, recall = evaluate_model(
            model=model, test_data=(X_test_padded, y_test)
        )
        # Guardar resultados
        results[model_name] = {
            'val_accuracy': max(history.history['val_accuracy']),
            'train_accuracy': max(history.history['accuracy']),
            'test_accuracy': accuracy,
            'test_precision': precision,
            'test_recall': recall
        }
```

4.5. Explicación de las Dimensiones de las Arquitecturas con GloVe 50

A continuación se describe cada una de las arquitecturas RDD1, RDD2 y RDD3, las cuales fueron entrenadas utilizando embeddings preentrenados de GloVe con una dimensión de 50. El vocabulario usado en estas arquitecturas es de 400,001 términos y la longitud máxima de secuencia es de 121 palabras.

4.5.1. RDD1_glove_50

La arquitectura **RDD1_glove_50** comienza con una capa de `Embedding` que tiene una salida de forma `(None, 121, 50)`, donde 121 es la longitud de la secuencia y 50 es la dimensión del vector de embedding. Esta capa contiene 20,000,050 parámetros, que corresponden a los embeddings preentrenados de GloVe. Luego, se utiliza una capa `Flatten`, que convierte la salida en un vector unidimensional con forma `(None, 6050)`. Después de la capa de aplanamiento, se incluyen tres capas `Dense` intercaladas con capas de `Dropout`, pero las dimensiones exactas de estas capas no se muestran en el resumen del modelo. Esta arquitectura tiene un total de 20,000,050 parámetros no entrenables, ya que los embeddings de GloVe no son ajustados durante el entrenamiento.

4.5.2. RDD2_glove_50

La arquitectura **RDD2_glove_50** es muy similar a RDD1, comenzando también con una capa de `Embedding` con la misma salida de forma `(None, 121, 50)` y con 20,000,050 parámetros. La capa `Flatten` convierte esta salida en un vector de tamaño `(None, 6050)`. Al igual que en RDD1, se incluyen varias capas `Dense` y `Dropout`, aunque no se especifican sus dimensiones exactas. Esta arquitectura también contiene un total de 20,000,050 parámetros no entrenables, y su entrenamiento no modifica los embeddings de GloVe.

4.5.3. RDD3_glove_50

La arquitectura **RDD3_glove_50** es más compleja que las anteriores, comenzando con una capa de `Embedding` con la misma salida de forma `(None, 121, 50)`. Luego, se aplica una capa de `SpatialDropout1D`, que mantiene la misma forma. A continuación, se utiliza una capa `Bidirectional LSTM`, que expande la salida a `(None, 121, 512)`. Posteriormente, se añade una capa de `MultiHeadAttention`, que mantiene las dimensiones de `(None, 121, 512)`. Esta arquitectura incluye capas de `GlobalAveragePooling1D` para reducir dimensionalidad, con salidas de tamaño `(None, 512)`, `(None, 256)`, y `(None, 128)`. La salida final de las capas de `Pooling` se combina en una capa `Concatenate`, con una salida de `(None, 896)`. Finalmente, el modelo contiene tres capas `Dense` con 512, 256 y 128 neuronas, seguidas por una capa de salida de tamaño 3 para la clasificación. En total, este modelo tiene 22,605,365 parámetros, de los cuales 2,603,523 son entrenables.

4.6. Resultados del Modelo con Embedding GloVe 50

Los resultados de las tres arquitecturas utilizando embeddings de GloVe 50 se evaluaron en términos de *accuracy*, *precision* y *recall*. Los resultados indican que la arquitectura más compleja (RDD3) tuvo un mejor desempeño que las otras dos, aprovechando las capas adicionales de atención y procesamiento bidireccional.

4.7. Explicación de las Dimensiones de las Arquitecturas con GloVe 100

En esta sección se describen las dimensiones de cada una de las capas de las tres arquitecturas (RDD1, RDD2 y RDD3), entrenadas con los embeddings preentrenados de GloVe de dimensión 100. El tamaño del vocabulario utilizado es de 400,001 términos, y la longitud máxima de la secuencia es de 121 palabras.

4.7.1. RDD1_glove_100

La arquitectura **RDD1_glove_100** comienza con una capa de `Embedding`, que toma como entrada secuencias de longitud 121 y genera un espacio de embeddings de dimensión 100, resultando en una salida de forma `(None, 121, 100)`. Esta capa contiene 40,000,100 parámetros que provienen del uso de los embeddings preentrenados de GloVe. Posteriormente, la capa `Flatten` convierte la salida en un vector unidimensional de tamaño `(None, 12100)`. Después de la capa de `Flatten`, el modelo incluye tres capas `Dense` intercaladas con capas de `Dropout`, pero las dimensiones de estas capas no se especifican en el resumen del modelo. Esta arquitectura tiene un total de 40,000,100 parámetros, que no son entrenables, ya que los embeddings de GloVe no se modifican durante el entrenamiento.

4.7.2. RDD2_glove_100

La arquitectura **RDD2_glove_100** comienza con una capa de `Embedding` no entrenable que utiliza los embeddings preentrenados de GloVe. Esta capa tiene una salida de forma `(None, 121, 100)` y un total de 40,000,100 parámetros no entrenables. Luego, la salida pasa a través de una capa `Flatten`, que convierte el tensor resultante en un vector unidimensional de tamaño `(None, 12100)`.

A diferencia de `RDD1`, `RDD2_glove_100` incluye una primera capa `Dense` con 512 neuronas y activación `ReLU`, seguida de una capa `Dropout` con una tasa del 40 % para prevenir el sobreajuste. Posteriormente, se añaden capas densas adicionales con 256, 128, y 64 neuronas, cada una intercalada con capas `Dropout` para mejorar la generalización del modelo.

Finalmente, la arquitectura termina con una capa de salida `Dense` con tres unidades y activación `Softmax`, adecuada para realizar la clasificación entre los tres autores. En total, el modelo tiene 40,000,100 parámetros, todos ellos no entrenables debido al uso de embeddings preentrenados.

4.7.3. RDD3_glove_100

La arquitectura **RDD3_glove_100** es considerablemente más compleja que las dos anteriores. Inicia con una capa de `Embedding` con salida de forma `(None, 121, 100)`, y luego pasa por una capa de `SpatialDropout1D`

que mantiene la misma forma de salida. Después de esta capa, se añade una capa `Bidirectional LSTM`, que expande la salida a `(None, 121, 512)`. A continuación, una capa `MultiHeadAttention` mantiene la salida en `(None, 121, 512)`. Tras aplicar una capa de normalización, el modelo utiliza tres capas de `Bidirectional LSTM`, que reducen progresivamente las dimensiones de la salida a `(None, 121, 256)` y `(None, 121, 128)`. Luego, se aplican capas de `GlobalAveragePooling1D`, que convierten las salidas en `(None, 512)`, `(None, 256)` y `(None, 128)`. Las salidas de estas capas se combinan utilizando una capa de `Concatenate`, que da como resultado una salida final de `(None, 896)`. Finalmente, el modelo tiene tres capas `Dense` con 512, 256, y 128 neuronas respectivamente, seguidas por una capa de salida de tamaño 3 para la clasificación. El total de parámetros de esta arquitectura es 42,707,815, de los cuales 2,705,923 son entrenables.

4.8. Resultados del Modelo con Embedding GloVe 100

Los resultados de las tres arquitecturas utilizando embeddings de GloVe 100 se evaluaron en términos de *accuracy*, *precision* y *recall*. Como se observa, la arquitectura más compleja (RDD3) tuvo el mejor rendimiento en comparación con las otras dos, aprovechando las capas adicionales de procesamiento y atención para mejorar los resultados en la clasificación.

4.9. Explicación de las Dimensiones de las Arquitecturas - Embedding GloVe 200

En esta sección se describen las dimensiones de las capas de las tres arquitecturas (RDD1, RDD2 y RDD3) utilizando un embedding de GloVe con dimensión 200. A continuación se explica cada arquitectura:

4.9.1. RDD1_glove_200

La arquitectura **RDD1_glove_200** inicia con una capa de `Embedding` que tiene un vocabulario de 400,001 palabras y genera embeddings con una dimensión de 200. La salida de esta capa tiene la forma `(None, 121, 200)`, donde 121 es la longitud máxima de las secuencias de entrada, y 200 es la dimensión de los vectores de palabras.
Después, se aplica una capa `Flatten`, que convierte la salida en un vector de tamaño `(None, 24200)`. A continuación, la arquitectura incluye tres capas `Dense`, intercaladas con capas de `Dropout` para prevenir el sobreajuste, aunque las dimensiones de las capas densas no se detallan en el resumen del modelo.
El número total de parámetros en esta arquitectura es 80,000,200, todos ellos no entrenables debido a que los embeddings son preentrenados.

4.9.2. RDD2_glove_200

La arquitectura **RDD2_glove_200** también comienza con la misma capa de `Embedding` de GloVe, que contiene 80,000,200 parámetros no entrenables y genera una salida de forma `(None, 121, 200)`. Posteriormente, la salida pasa a través de una capa `Flatten`, que convierte el tensor resultante en un vector unidimensional de tamaño `(None, 24200)`.
A diferencia de `RDD1`, `RDD2_glove_200` incluye una capa densa inicial de `512 neuronas` con activación `ReLU`, seguida de una capa `Dropout` con una tasa de desactivación del 40 % para prevenir el sobreajuste. Luego, se añaden dos capas `Dense`, una con 256 neuronas y otra con 128 neuronas, ambas intercaladas con capas `Dropout` con tasas del 40 % y 30 %, respectivamente, para mejorar la capacidad de generalización del modelo.
Finalmente, la arquitectura incluye una capa `Dense` con 64 neuronas y activación `ReLU`, antes de llegar a la capa de salida `Dense` de tres neuronas con activación `Softmax`, la cual realiza la clasificación en tres categorías de autores.
El número total de parámetros sigue siendo 80,000,200, todos no entrenables debido a los embeddings preentrenados de GloVe.

4.9.3. RDD3_glove_200

La arquitectura **RDD3_glove_200** utiliza un enfoque más avanzado. Comienza con la capa `Embedding` que tiene una salida de forma `(None, 121, 200)` y un total de 80,000,200 parámetros no entrenables. Luego, se aplica

una capa `SpatialDropout1D`, que mantiene la misma forma de salida.
A continuación, se introduce una capa `Bidirectional LSTM` que produce una salida de tamaño `(None, 121, 512)`. Esta capa es seguida por una capa de `LayerNormalization` y una capa de `MultiHeadAttention`, ambas con una salida de tamaño `(None, 121, 512)`. La arquitectura utiliza varias capas bidireccionales adicionales y luego pasa a tres capas de `GlobalAveragePooling1D` con tamaños de salida `(None, 512)`, `(None, 256)`, y `(None, 128)`.
Finalmente, los resultados de estas capas se combinan mediante una capa `Concatenate`, con una salida de `(None, 896)`. La arquitectura termina con capas densas (`Dense`) de 512, 256, y 128 neuronas, intercaladas con `BatchNormalization` y `Dropout`, finalizando con una capa de salida `Dense` de tres unidades con activación `Softmax`.
El número total de parámetros en esta arquitectura es 82,912,715, de los cuales 2,910,723 son entrenables.

4.10. Resultados preliminares

A continuación, se presenta un resumen preliminar de los resultados obtenidos con los diferentes tamaños de embeddings preentrenados de GloVe:

Modelo	Mejor Val Accuracy	Mejor Train Accuracy	Test Accuracy	Test Precision
RDD1_GloVe_50	0.6686	0.7434	0.6798	0.6787
RDD1_GloVe_100	0.6788	0.6507	0.6995	0.7056
RDD1_GloVe_200	0.6255	0.5837	0.6258	0.5119
RDD2_GloVe_50	0.6602	0.6571	0.6765	0.6718
RDD2_GloVe_100	0.6162	0.5267	0.6187	0.4859
RDD2_GloVe_200	0.6340	0.6295	0.6398	0.5002
RDD3_GloVe_50	0.9155	0.8950	0.9128	0.9139
RDD3_GloVe_100	0.9070	0.8956	0.9126	0.9124
RDD3_GloVe_200	0.9341	0.9433	0.9300	0.9319

Tabla 4.1: Resultados preliminares de los modelos entrenados con embeddings GloVe.

Estos resultados serán analizados y comparados con los obtenidos con los embeddings entrenados con Word2Vec para determinar qué tipo de embeddings proporciona un mejor rendimiento en la tarea de clasificación de autores.

Comparación de resultados

En el análisis de las diferentes arquitecturas y embeddings utilizados, se observa claramente que los embeddings preentrenados de GloVe ofrecen un mejor rendimiento en comparación con los embeddings creados por el grupo a partir de los libros. Esto es evidente en las métricas de evaluación, donde los embeddings de GloVe logran una mayor consistencia y capacidad para capturar mejor la información contextual de los textos, permitiendo que los modelos de clasificación sean más precisos y eficientes en la tarea.

5.1. Impacto del tamaño de los embeddings

Al comparar los diferentes tamaños de embeddings de GloVe (50, 100 y 200 dimensiones), los resultados muestran que los embeddings de mayor dimensión tienden a ofrecer un mejor rendimiento, especialmente en arquitecturas más complejas. Esto es particularmente claro en el modelo **RDD3_GloVe_200**, que obtuvo las mejores métricas generales con una precisión de prueba de 0.9300 y una precisión en validación de 0.9341. Este patrón sugiere que los embeddings con mayor número de dimensiones capturan mejor las relaciones semánticas y proporcionan más información útil para las tareas de clasificación de autores.

5.2. Comparación entre modelos

Entre las arquitecturas evaluadas, **RDD3** consistentemente superó a **RDD1** y **RDD2**, independientemente del tamaño de los embeddings. Esto destaca el poder de los modelos que utilizan LSTM bidireccionales y mecanismos avanzados como la atención y las conexiones residuales para capturar dependencias a largo plazo en los textos. En particular, **RDD3_GloVe_200** alcanzó un excelente equilibrio entre la precisión de entrenamiento (0.9433) y la precisión de validación (0.9341), sin signos de sobreajuste, lo que sugiere que el modelo pudo generalizar bien los patrones en los datos de prueba.

El modelo **RDD1**, a pesar de ser más simple en su arquitectura, también mostró mejoras notables al utilizar embeddings GloVe. Con un *Test Accuracy* de 0.6798 en **RDD1_GloVe_50** y 0.6995 en **RDD1_GloVe_100**, se observa que incluso un modelo menos complejo puede beneficiarse significativamente de mejores representaciones semánticas proporcionadas por embeddings preentrenados que sean bastante grandes.

Por otro lado, el rendimiento de **RDD2** fue intermedio, con un *Test Accuracy* de 0.6765 en **RDD2_GloVe_50** y 0.6398 en **RDD2_GloVe_200**, lo que indica que esta arquitectura, aunque más avanzada que **RDD1**, no fue tan efectiva como **RDD3** para esta tarea de clasificación.

5.3. Conclusiones clave

- **Embeddings preentrenados mejoran el rendimiento**: Los resultados muestran que los modelos que utilizaron embeddings preentrenados de GloVe superaron consistentemente a los que usaron los embeddings entrenados localmente con Word2Vec. Este aumento en el rendimiento se debe a la capacidad de GloVe de capturar relaciones semánticas más ricas, entrenadas en grandes corpus de texto.
- **El tamaño de los embeddings importa**: Los embeddings de mayor dimensión (GloVe 200) proporcionaron mejores resultados en general, sugiriendo que un mayor número de dimensiones ofrece una representación

más detallada y útil para la tarea de clasificación.

- **Arquitectura avanzada produce mejores resultados**: La arquitectura **RDD3** destacó entre las tres evaluadas, obteniendo las mejores métricas tanto en precisión como en generalización, especialmente al utilizar embeddings GloVe de mayor dimensión.

En definitiva, este análisis sugiere que, aunque las arquitecturas de los modelos son importantes, el uso de embeddings preentrenados puede marcar una diferencia significativa en el rendimiento. En particular, los resultados obtenidos con GloVe resaltan la importancia de utilizar representaciones semánticas robustas en tareas de clasificación de texto, como la clasificación de autores. A medida que los modelos se vuelven más complejos, los embeddings más ricos permiten explotar mejor su potencial, como se observó en el caso de **RDD3_GloVe_200**, que alcanzó las mejores métricas generales en esta tarea.