Analísis sobre las obras de William Shakespeare

Brian Britos, Centro de Matemática,
Facultad de Ciencias, Universidad de la República
Montevideo, Uruguay
bbritos@cmat.edu.uy
Juan Carlos Pellegrini
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
jpellegrini11@gmail.com

Resumen

En este informe se presenta un estudio sobre el análisis de las obras de William Shakespeare mediante el uso de técnicas de procesamiento de lenguaje natural y aprendizaje automático. El objetivo principal es clasificar los textos y determinar a qué personaje corresponden, dentro de un universo reducido a tres personajes. Para lograrlo, se emplearon diversas herramientas y algoritmos, como la eliminación de stopwords, el uso de conteo de palabras y Tf-idf, el análisis de componentes principales (PCA) para reducir la dimensionalidad, se aplicaron los algoritmos de clasificación *Multinomial Naive Bayes* (multinomialNB), *Support Vector Machines* (SVM), y finalmente FastText para entrenar un modelo para clasificar los textos.

I. Introducción

El análisis de textos literarios ha sido objeto de estudio en diversas disciplinas, y uno de los escritores más influyentes en la historia de la literatura es indudablemente William Shakespeare. Sus obras han sido objeto de análisis y admiración a lo largo de los siglos. El equipo realizó un análisis de las obras de Shakespeare utilizando técnicas de procesamiento de lenguaje natural (NLP) y aprendizaje automático. Para llevar a cabo este análisis, hemos utilizado una fuente de datos diversa y gratuita que contiene la colección completa de obras de Shakespeare. Esta base de datos, contiene una amplia variedad de textos, que abarcan desde tragedias icónicas como ”Hamlet” Romeo y Julieta”hasta comedias como ”Sueño de una noche de verano” ”Mucho ruido y pocas nueces”. La inclusión de esta fuente de datos permitió tener un conjunto completo y representativo de las obras de Shakespeare para este estudio.

El trabajo se divide en dos partes principales. En primer lugar (II-A), se analiza la recopilación y preparación de los datos utilizados en el estudio. Además, se explora la técnica de representación numérica de texto (*bag-of-words* y *tf-idf*) para transformar el contenido textual en un formato adecuado para el análisis computacional. Por otro lado, en la segunda etapa (II-B) se aborda el entrenamiento y la evaluación de modelos, donde se describen los algoritmos utilizados, el proceso de entrenamiento de los modelos y la evaluación de su desempeño mediante medidas como *accuracy* o *F1-score*.

II. Sección Principal

II-A. Parte 1

En la etapa inicial de este estudio, se realiza un proceso de limpieza de los datos para asegurar su calidad y coherencia. Este proceso implica la eliminación de contracciones, la conversión de todo el texto a minúsculas, la eliminación de signos de puntuación y de *Stop Words*. Una vez que los datos están limpios, se procede a seleccionar tres personajes para continuar con el análisis: Antony, Cleopatra y Queen Margaret.

A continuación, se realiza una partición de los datos en conjuntos de entrenamiento (*train*) y prueba (*text*) utilizando la función *train_text_split* de la biblioteca *sklearn*. Se decidió separa el $30\,\%$ de la muestra para *test*.

La figura 1 muestra el balance de párrafos de cada personaje en los conjuntos *train* y *test*.

Luego, se aplican técnicas de procesamiento de texto, como el conteo de palabras, el modelo de *Bag of Words*, los *n-gramas* y el *tf-idf*, con el objetivo de representar los textos de manera numérica y capturar características relevantes. A continuación se explica brevemente en que consiste cada una de estas técnicas.

Bag of Words (BoW): En *BoW*, cada documento se trata como una ”bolsa”que contiene todas las palabras presentes en él, sin tener en cuenta el orden o la estructura gramatical. El enfoque consiste en crear un vocabulario único de todas las palabras

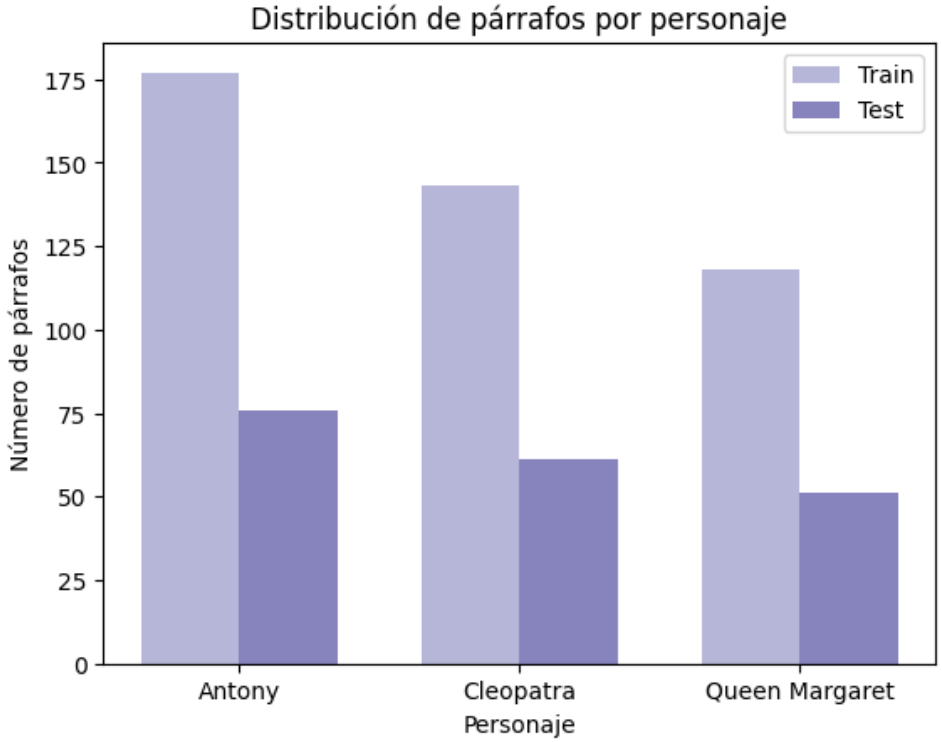


Figura 1: Distribución de párrafos por personaje en los conjuntos Train y Test.

que aparecen en el conjunto de entrenamiento y luego contar la frecuencia de cada palabra en cada documento. El resultado es una *sparse matrix* (es decir, una matriz que tiene solo algunas entradas no nulas) cuyas filas coinciden con los párrafos y sus columnas con el vocabulario presente en todos los párrafos, esto por esto último que es una *sparse matrix*, dado que una palabra suele aparece en una proporción pequeña de párrafos (ya se eliminaron *Stop Words*).

Como ejemplo, si se consideran los siguientes tres párrafos de subconjunto estudiado: *'pardon pardon'*, *'nay help'* y *'fares lord help lords king dead'*, y se aplican BoW, se obtiene la siguiente matriz

	pardon	nay	help	fares	lord	king	dead
1	2	0	0	0	0	0	0
2	0	1	1	0	0	0	0
3	0	0	1	1	2	1	1

n-grama: Un *n-grama* es una secuencia contigua de n elementos en un texto o corpus. Los elementos pueden ser palabras, caracteres o incluso fonemas, dependiendo del nivel de análisis que se esté considerando. En el contexto de este trabajo, se usara como elemento las palabras.

El valor de n en un *n-grama* indica el número de elementos consecutivos que se agrupan juntos. Por ejemplo un 2-grama (también conocido como bigrama) agrupa dos palabras consecutivas, un 3-grama (trigrama) agrupa tres palabras consecutivas, y así sucesivamente.

La utilidad de estos, es que pueden proporcionar información sobre la estructura y la co-ocurrencia de las unidades lingüísticas en un texto determinado, a diferencia de *BoW* (se puede pensar como que siempre utiliza 1-grama). Si se consideran los mismos tres párrafos que antes, pero ahora utilizando 2-grama, se obtiene

	pardon pardon	nay help	fares lord	lord help	help lord	lord king	king dead
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	1	1	1	1

Tf-idf *Tf-idf* es una medida que cuenta la frecuencia de una palabra en un párrafo ponderando por la cantidad de párrafos donde aparece la misma. El objetivo de usar *Tf-idf* en lugar de simplemente la frecuencia de ocurrencia es reducir el impacto de los *tokens* que ocurren con mucha frecuencia y por ende son empíricamente menos informativos. La ecuación matemática es

$$\textit{tf-idf}(t,d) = tf(t,d) \times idf(t),$$

donde $tf(t,d)$ (*term-frequency*) es la frecuencia de la palabra t en el párrafo d (es decir, la cantidad de ocurrencias de t divido la cantidad de palabras en el párrafo d), mientras que $idf(t)$ (*inverse document-frequency*) mide que tan común es una palabra en la colección completa de los párrafos.

Hay varias maneras de definir $idf(t)$, en particular en el método *TfidfTransformer* [1] se definen como:

$$idf(t) = log\left(\frac{1+n}{1+df(t)}\right) + 1,$$

donde n es la cantidad total de párrafos en la colección, $df(t)$ se denomina *document frequency of t* y es la cantidad de párrafos en la colección que contiene la palabra t.

Además, se aplica el análisis de componentes principales (*PCA*) para reducir la dimensionalidad de los datos. La varianza explicada por cada componente se visualiza para comprender la estructura de los datos y determinar cuánta información se conserva en el proceso de reducción de dimensionalidad. La siguiente figura es un biplot.

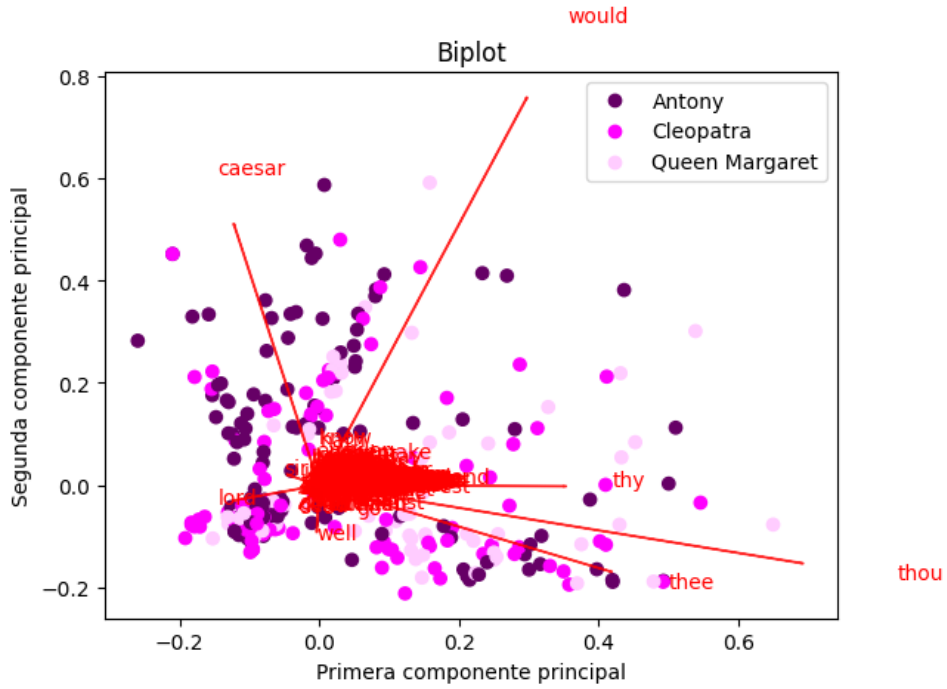


Figura 2: Biplot del resultado obtenido al aplicar *PCA*

Rápidamente se llega a la conclusión que 2 componentes principales son insuficientes para poder separa los personajes. La varianza explicada en este caso es $4,89\,\%$

Con la intención de entender cuantas componentes principales habría que considerar para poder separar los personajes, se realizó calculó la varianza explicada para las 100 primeras componentes principales. Con las primeras 100 componentes principales,

[1] Sklearn - TfidfTransformer

se consigue un $56,2\,\%$ de la varianza explicada. Esto es un indicador de que los datos no serán fácilmente separables en dimensión bajan, al menos, de forma lineal.

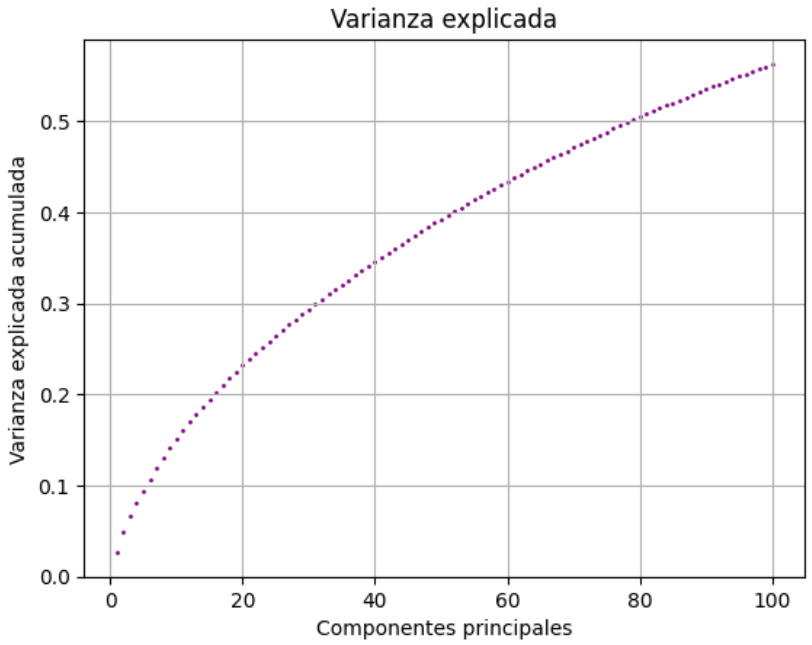


Figura 3: Varianza explicada de las 100 primeras componentes principales

II-B. Parte 2

En la siguiente etapa de este estudio, se lleva a cabo el entrenamiento de dos modelos de clasificación: *Multinomial Naive Bayes* (*MultinomialNB*) y *Support Vector Machine* (*SVM*). Estos modelos se entrenan utilizando el conjunto de datos preparado en la etapa anterior. Además, se emplea la técnica de *Cross-Validation* para buscar el mejor conjunto de parámetros para cada modelo, optimizando así su rendimiento.

Posteriormente, se incorpora la librería FastText, la cual se utiliza para realizar un análisis adicional de los textos. FastText es una poderosa herramienta de procesamiento de lenguaje natural que permite una representación de palabras y textos de alta calidad.

Cross Validation: La técnica *Cross Validation* es utilizada en el aprendizaje automático para evaluar el rendimiento de un modelo utilizando un conjunto de datos limitado. Consiste en dividir el conjunto de datos en subconjuntos de entrenamiento y prueba de manera iterativa, de modo que el modelo se entrene y se evalúe en múltiples combinaciones diferentes.

En resumen, la validación cruzada implica:

1. Dividir el conjunto de datos en k subconjuntos de tamaño similar.
2. Realizar k iteraciones, en cada una de las cuales se elige un subconjunto diferente como conjunto de prueba y los restantes como conjunto de entrenamiento.
3. Entrenar el modelo utilizando el conjunto de entrenamiento y evaluar su rendimiento utilizando el conjunto de prueba.
4. Calcular la métrica de rendimiento para cada iteración.

La validación cruzada es útil para evaluar cómo se desempeñará un modelo en datos no vistos y para evitar problemas de sobreajuste (*overfitting*). Al realizar múltiples iteraciones, se obtiene una estimación más robusta del rendimiento del modelo.

En este trabajo se utilizó $k = 4$ (es decir, 4 subconjuntos) y las métricas utilizadas fueron *Accuracy* para los modelos *MultinomialNB* y *SVM*, mientras que para *FastText* se decidió utilizar la métrica *F1-Score*. Además para discernir cual set de parámetros es mejor se realizaron gráficos de violín, los cuales dan una idea de la densidad de métrica en cuestión. El punto blanco dentro de cada violín es la mediana, mientras que el largo del mismo representa la variabilidad de la métrica. El mejor set de parámetros será aquel que tenga la mediana más alta y al mismo tiempo sea lo menos largo posible (es decir, la métrica

varia poco y por lo tanto tiene cierta estabilidad). En el eje vertical, se muestran los valores de los parámetros siguiendo el patrón **stop_words, ngram, idf**.

Multinomial Naive Bayes

La figura 4 muestra los gráficos de violín obtenidos al iterar la técnica *Cross-Validation* en 28 diferentes sets de parámetros. Se observa que el los parámetros **'None', (1,1), False** es el que da mejores resultados, por lo que se entreno el modelo con estos parámetros.

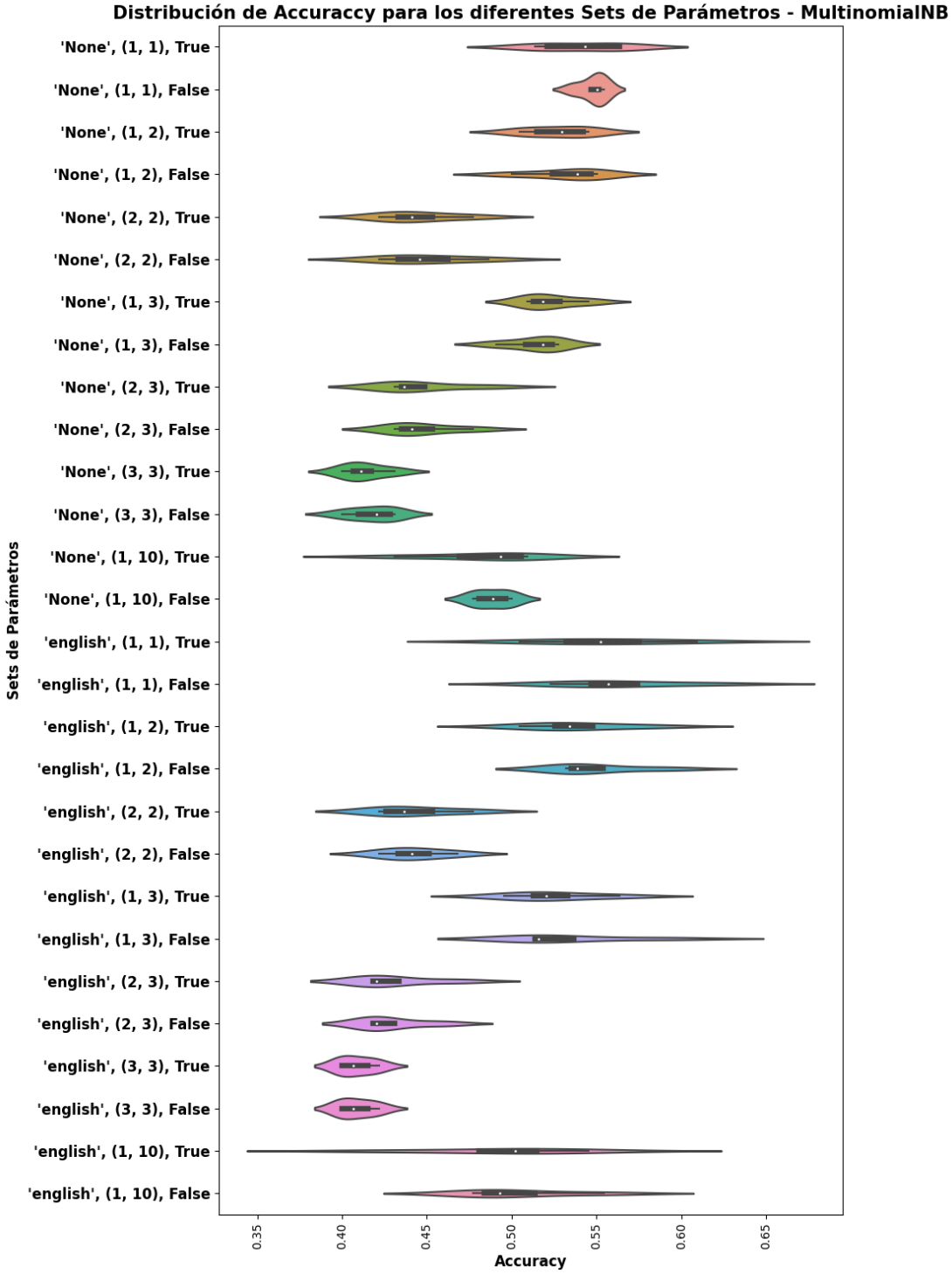


Figura 4: Gráfico de violín para la distribución de Accuracy utilizando *MultinomialNB*

La métrica *Accuracy* en este caso es de $86,5\,\%$. La figura 5 es la matriz de confusión obtenida. Se decidió mostrar los valores porcentuales según la etiqueta real (la fila) en lugar de los absolutos, para que sea más sencillo extraer conclusiones. Se observa que predice relativamente bien la etiqueta correcta (valores en la diagonal), aunque tanto para *Cleopatara* como *Queen Margaret* predijo que el $20\,\%$ de sus párrafos fueron dichos por *Antony*.

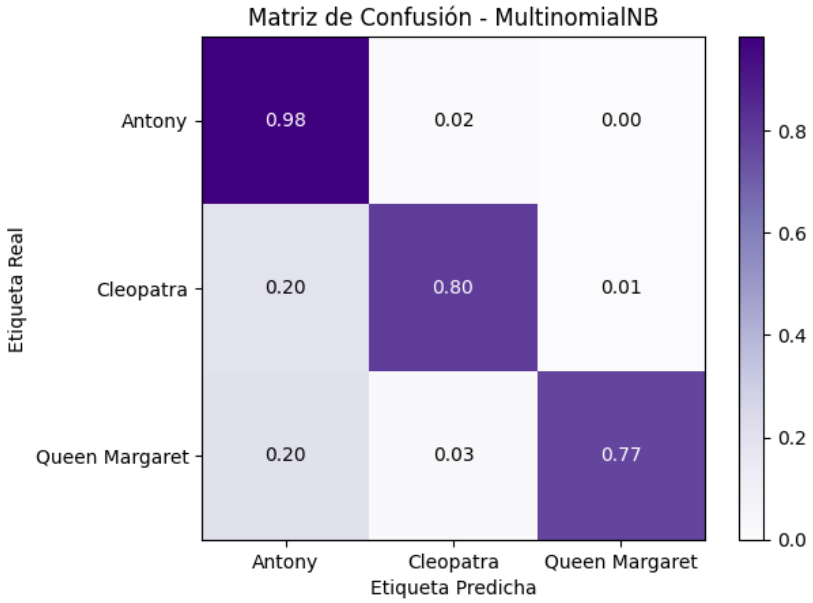


Figura 5: Matriz de Confusión utilizando *MultinomialNB*

El modelo muestra un interés por el personaje de Antony, ya que es con quién comete más errores al predecir. Este hallazgo resulta interesante, especialmente considerando la evidencia presentada en la figura 1, donde se observa que Antony tiene la mayor cantidad de párrafos. Esta observación sugiere la posibilidad de una introducción de sesgo en el modelo debido a la diferencia de párrafos para cada personaje.

Support Vector Machine:

La técnica de *Support Vector Machine* (*SVM*) es un algoritmo de aprendizaje supervisado utilizado tanto para problemas de clasificación como de regresión. El objetivo principal de *SVM* es encontrar un hiperplano que mejor separe las diferentes clases de datos.

En el caso de clasificación, *SVM* busca construir un hiperplano que maximice la separación entre las clases, eligiendo aquel que esté más alejado de los puntos de cada clase. Los puntos más cercanos al hiperplano se llaman vectores de soporte, de ahí el nombre del algoritmo.

En muchos casos los datos no son linealmente separables, por lo que se utiliza una función de kernel para mapear los datos de entrada a un espacio (*llamado espacio de características*) de mayor dimensión (posiblemente infinita), donde se puede encontrar un hiperplano separación.

Una vez entrenado, *SVM* puede clasificar nuevos ejemplos asignándolos a una clase en función de qué lado del hiperplano se encuentren.

Se repitió el procedimiento de *Cross Validation* con los mismos sets de parámetros para entrenar un modelo *SVM*. La figura 6 es un gráfico de violín para encontrar el mejor ser de parámetros. En este caso, el mejor set de parámetros es **'None', (1,1), True**.

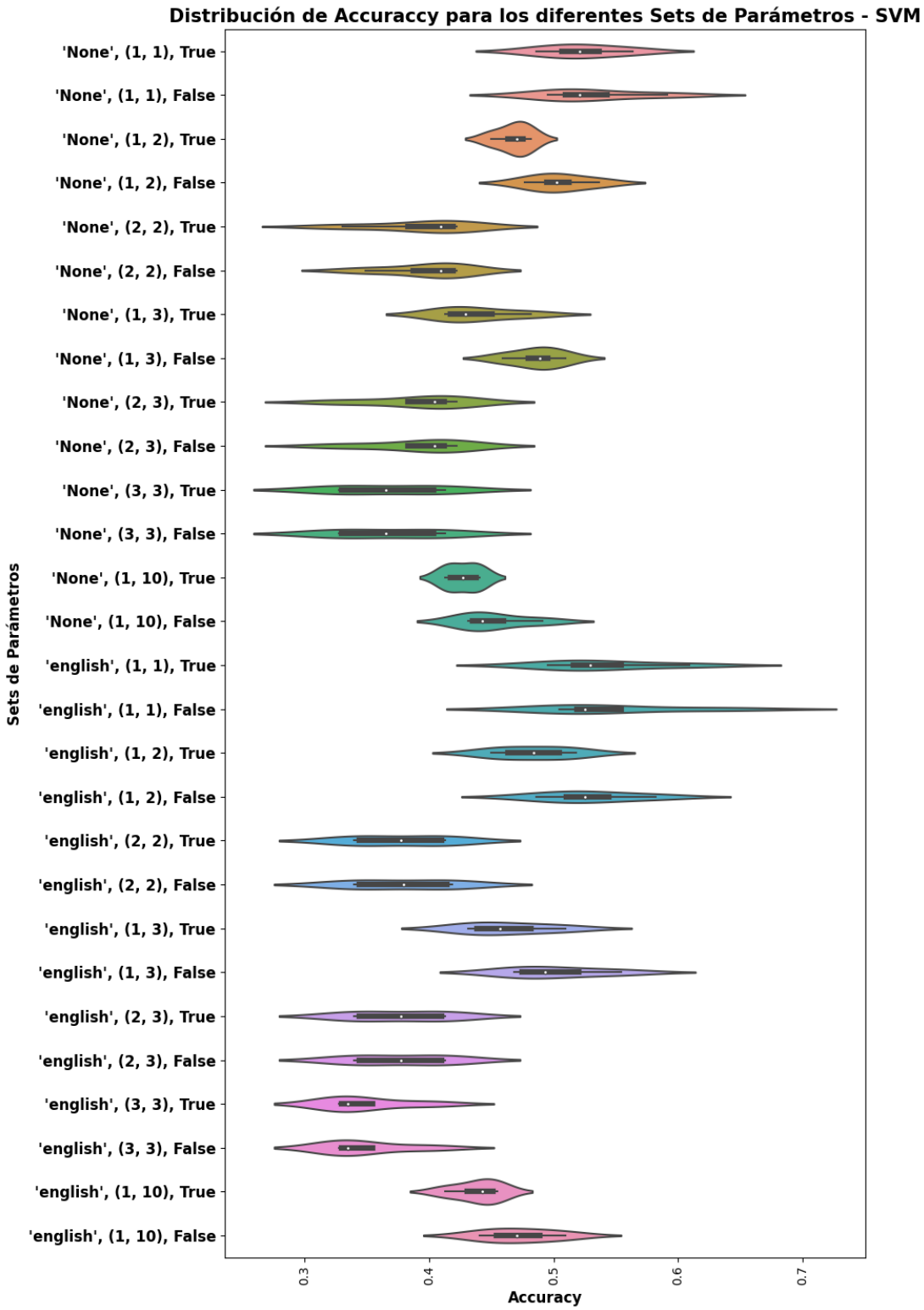


Figura 6: Gráfico de violín para la distribución de Accuracy utilizando *SVM*

Con la técnica *SVM* y los parámetros elegidos se obtuvo un *Accuracy* de $98,2\,\%$, de hecho predijo correctamente el $97\,\%$ de los párrafos. La figura 7 muestra la matriz de confusión obtenida. Es interesante observar que con este modelo, no se tiene un interés particular por ningún personaje, a pesar del pequeño desbalance de párrafos.

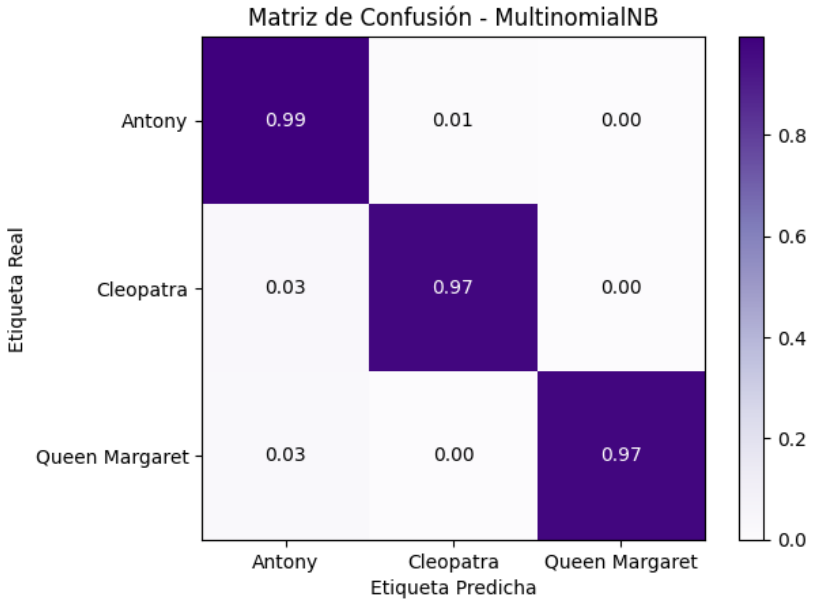


Figura 7: Matriz de Confusión utilizando *SVM*

Cambiando un personaje

A continuación se entrenaron los mismos modelos con los mismos parámetros, pero se cambio el personaje *Antony* por *Hermione*. La figura 8 muestra la distribución por párrafos con el cambio de personaje. Es de esperar que los modelos tenga preferencia por los personajes *Cleopatra* y *Queen Margaret* por tener una cantidad mayor de párrafos que el personaje *Hermione.*

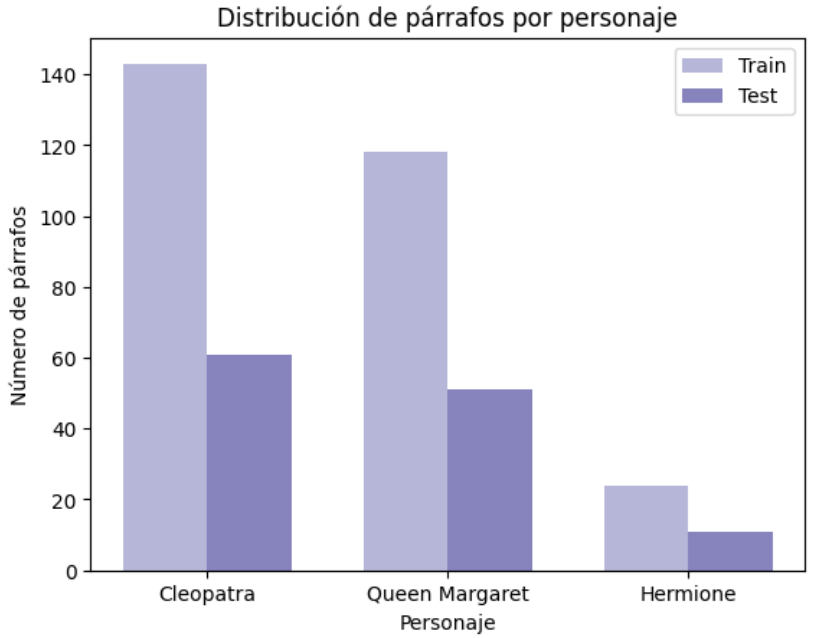
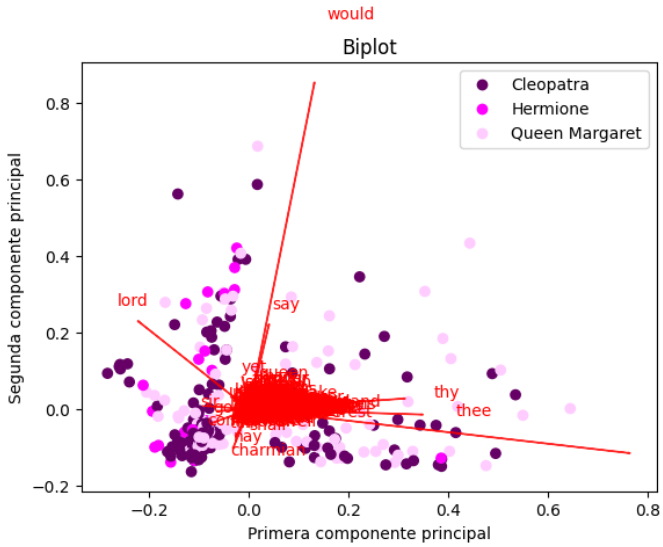


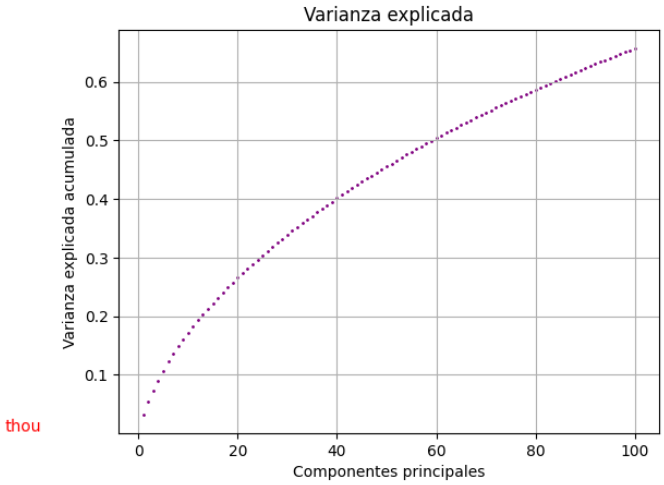
Figura 8: Distrubución de párrafos por personaje en los conjuntos Train y Test.

Al aplicar *PCA*, se obtuvo la figura 9a. A diferencia de lo que ocurría con *Antony*, parece ser que las palabras utilizadas por *Hermione* y *Cleopatra* tienen cierta correlación, dado que se agrupan en una misma zona.

Por otro lado, la figura 9b muestra la varianza explicada para este caso, al utilizar 100 componentes principales, la varianza explicada haciende a $65,7\,\%$.



(a) Matriz de Confusión para el modelo *MultinomialNB*



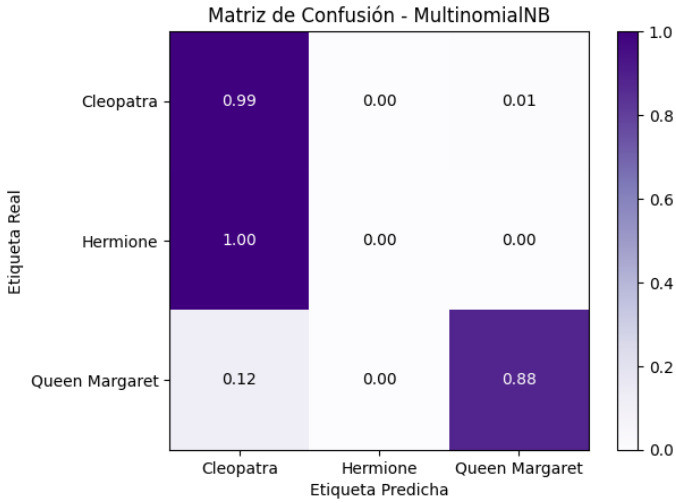
(b) Matriz de Confusión para el modelo *SVM*

Figura 9: Matrices de confusión para ambos modelos

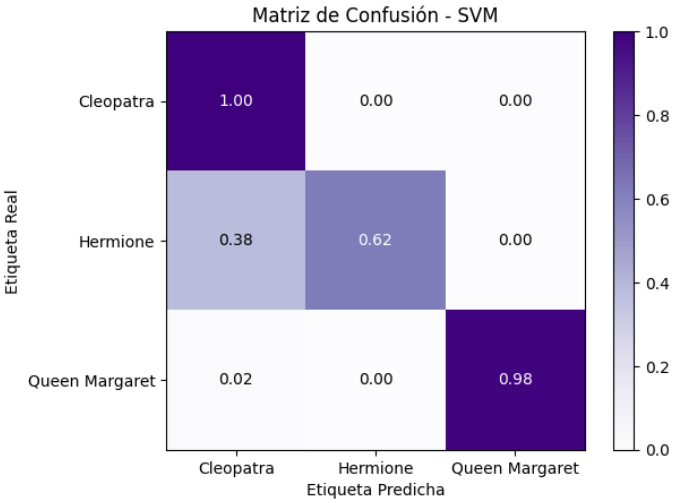
Multinomial Naive Bayes - SVM : Hermione

Se utilizaron los mismos parámetros encontrados para la parte anterior, en este caso la métrica *Accuracy* fue de $86,3\,\%$. Por otro lado, al entrenar el modelo *SVM* se obtuve un *Accuracy* de $96,1\,\%$.

En la figura 10 se aprecian ambas matrices de confusión. Se aprecia que el modelo *MultinomialNB* predijo cero vez al personaje *Hermione*, aunque era esperado que se equivoque mayormente en este personaje, por el gran desbalance en las cantidades de párrafos, sorprendio que no predijera ninguna vez. Específicamente, los párrafos dichos por esta personaje los clasifico como de *Cleopatra*, que es quién más párrafos dice. Por otro lado, al utilizar *SVM* se aprecia que a diferencia del modelo *MultinomialNB*, el $62\,\%$ de los párrafos dichos por *Hermione* los clasifico correctamente, aunque sigue habiendo un porcentaje alto (el restante $38\,\%$ que los asigno a *Cleopatra*). Es interesante notar que a diferencia de cuando se utilizó *Antony* en lugar de *Hermione*, donde los párrafos estaban mejor balanceados, al aplicar *SVM* sigue habiendo una preferencia por el personaje que más párrafos dice.



(a) Matriz de Confusión para el modelo *MultinomialNB*



(b) Matriz de Confusión para el modelo *SVM*

Figura 10: Matrices de confusión para ambos modelos

FastText

FastText[2] es una librería gratuita y open-source que permite aprender representaciones de texto y clasificaciones de texto. Se puede usar en hardware genérico, sin necesidad de GPU dedicadas.

En esta parte se utilizan nuevamente los personaje *Antony*, *Cleopatra* y *Queen Margaret*. También aprovecharemos que el método devuelve las métricas *Precision* y *Recall* para evaluar el redimiendo del modelo con el indicador *F1-Score*. A continuación se recuerda como se calculan estás métricas.

Si VP, FP, VN y FN son la cantidad de *Verdaderos Positivos*, *Falsos Positivos*, *Verdaderos Negativos* y *Falso Negativos* respectivamente, entonces

$$Precision = \frac{VP}{VP + FP} \qquad y \qquad Recall = \frac{VP}{VP + FN}.$$

Es decir, la medida *Precision* representa qué proporción de los identificados como positivos son realmente positivos. Mientras que la medida *Recall* da una idea de qué proporción de positivos fueron identificados correctamente.

Finalmente, el *F1-Score* permite incorporar ambas medidas anteriores en una única métrica y se calcula como

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall}.$$

La librería de Python utilizada permite elegir ciertos parámetros, los utilizados para este trabajo fueron **lr** (*Learning Rate*), **epoch** y **ngram**. La figura 12 muestra los distintos valores obtenidos de *F1-Score* para cada set de parámetro. Se obtuvo un *Accuracy* de $69,7\,\%$. La figura 11 muestra la matriz de confusión obtenida. Aunque el modelo acierta con menor frecuencia que antes, es interesante notar que sigue prefiriendo al personaje con más párrafos (*Antony*).

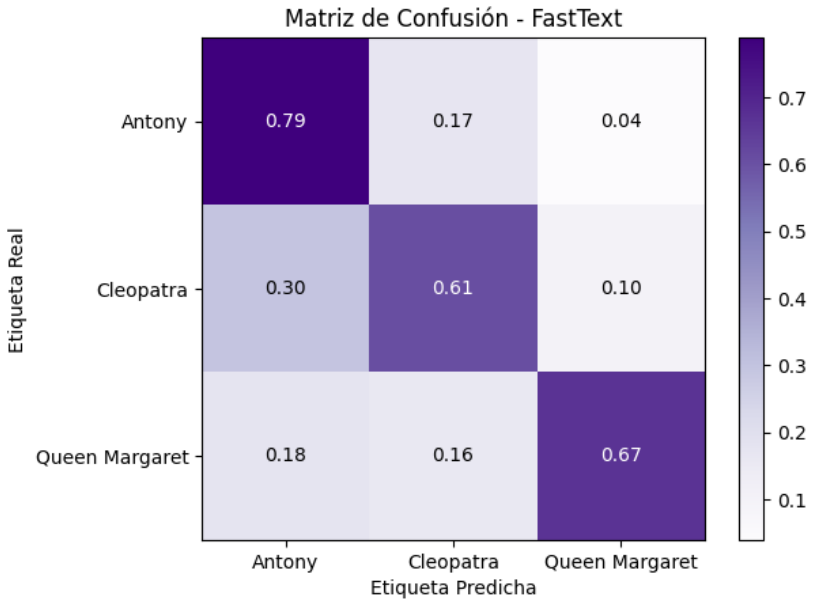


Figura 11: Matriz de Confusión para el modelo *FastText*.

[2]FastText

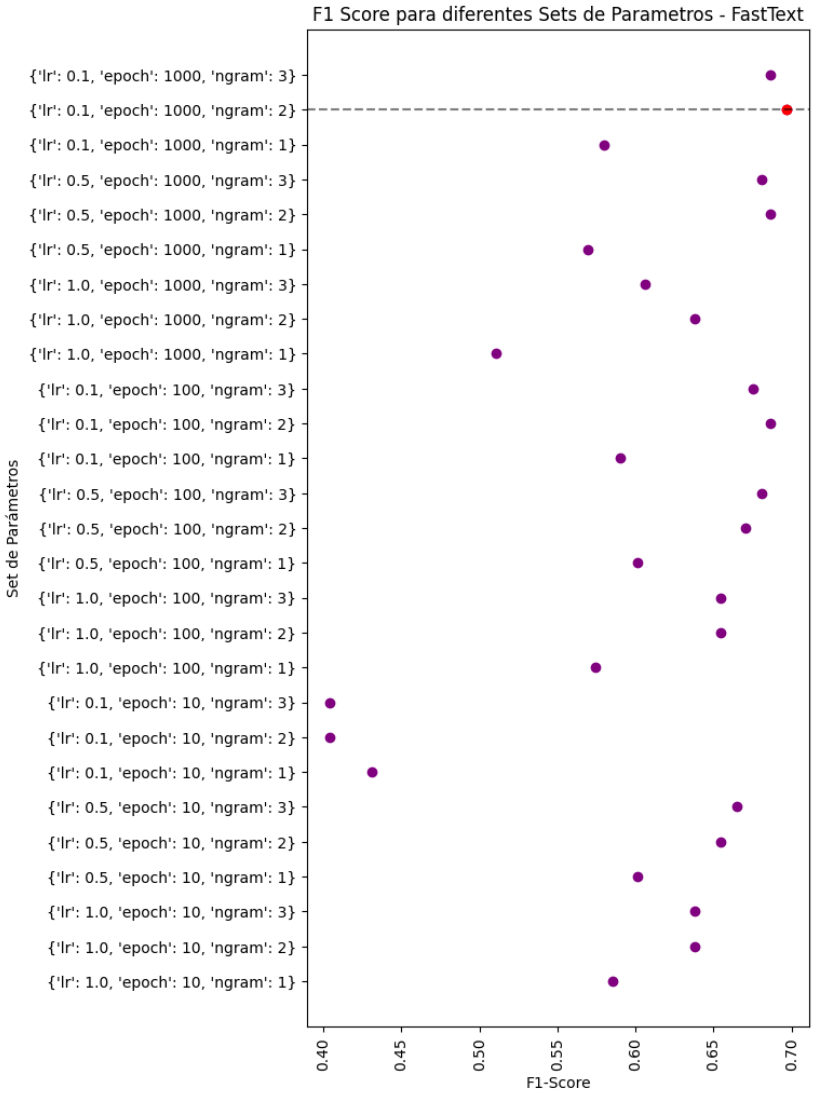


Figura 12: Valores de la métrica *F1-Score* para distintos sets de parámetros. En rojo se marca el set de parámetro que dio mejor resultado.

III. Conclusiones y trabajo futuro

Los resultados obtenidos muestran que los datos son demasiados complicados para poder entenderlos con dos componentes principales, lo cuál era de esperarse dado la gran complejidad del lenguaje. Por otro lado, al momento de clasificar el algoritmo *MultinomialNB* alcanzó una precisión del $86,5\,\%$, mientras que *SVM* obtuvo una precisión del $97,7\,\%$. Sin embargo, al reemplazar el personaje de *Antony* por *Hermione*, los resultados disminuyeron ligeramente, con una precisión de $86,3\,\%$ para el modelo *MultinomialNB* y $96,1\,\%$ para *SVM*. Es interesante destacar que, tanto en el escenario con *Antony* como en el caso de *Hermione*, la aplicación de *SVM* arrojó resultados superiores a los obtenidos mediante el uso de *MultinomialNB*, en particular se observa una menor predisposición por el personaje con más párrafos.

Además, se aplicó el algoritmo *FastText*, el cual logró una precisión del $69,7\,\%$. Aunque este resultado es inferior en comparación con los otros algoritmos, puede deberse a las características específicas de las obras de Shakespeare y la naturaleza compleja del lenguaje utilizado.

En general, utilizar FastText en lugar de algoritmos de clasificación como *MultinomialNB* o *SVM* puede proporcionar ventajas en términos de representaciones de palabras enriquecidas y manejo de palabras fuera del vocabulario. Sin embargo, también existen desventajas relacionadas con el consumo de memoria, complejidad del modelo y dificultad para capturar el contexto a largo plazo. Estas consideraciones deben tenerse en cuenta al elegir la combinación de técnicas más adecuada para un problema de clasificación de texto específico.

Cuando se realizó el cambio de *Antony* por *Hermione* en el estudio, se observó que los datos quedaron desbalanceados. Esta situación tuvo un impacto significativo en los resultados, ya que los modelos de clasificación comenzaron a mostrar una preferencia desproporcionada hacia ciertos los otros dos personajes en lugar de mantener un equilibrio adecuado. Para abordar este problema, se exploraron técnicas de manejo de datos desbalanceados, como el sobre-muestreo y el submuestreo.

El sobre-muestreo consiste en generar nuevas instancias sintéticas de la clase minoritaria para equilibrar la proporción con la clase mayoritaria. Esto se logra mediante técnicas como la duplicación de muestras existentes o la generación de muestras sintéticas basadas en las características de las instancias existentes. Por otro lado, el submuestreo implica reducir la cantidad de instancias de la clase mayoritaria para igualarla a la cantidad de instancias de la clase minoritaria. Esto se puede lograr eliminando muestras aleatoriamente o mediante la selección de instancias representativas.

Al utilizar estas técnicas de sobre-muestreo o submuestreo, se busca lograr un conjunto de datos más equilibrado que permita a los modelos de clasificación abordar de manera efectiva el desafío de la clasificación de los personajes de las obras de William Shakespeare. Es importante tener en cuenta que la elección de la técnica adecuada dependerá del contexto específico y de las características de los datos. Un enfoque riguroso de evaluación y validación de los modelos también es fundamental para garantizar resultados confiables y generalizables.

En general, este estudio demuestra que las técnicas de procesamiento de lenguaje natural y aprendizaje automático pueden ser efectivas para clasificar los párrafos de las obras de Shakespeare y determinar a qué personaje pertenecen, al menos con los cuatro personajes utilizados.

Como una mejora a este trabajo, se plantea la incorporación de la técnica *Word2vec*[3], que es un algoritmo utilizado para generar representaciones vectoriales de palabras en un corpus de texto. Su objetivo principal es capturar el significado semántico y las relaciones entre las palabras.. Estos vectores representan la semántica de las palabras en función de su contexto y relaciones con otras palabras. Las similitudes y relaciones entre palabras se reflejan en la proximidad y direccionalidad de los vectores en el espacio vectorial.

Al usar *Word2vec* en lugar de métodos tradicionales como *BoW* o *Tf-idf*, se pueden obtener diferencias significativas en los resultados. En lugar de tratar cada palabra como una entidad independiente, el método *Word2vec* captura la similitud semántica y el contexto entre palabras. Esto permite una mejor representación de las relaciones entre palabras y mejora el rendimiento en tareas como clasificación de texto, agrupamiento o recuperación de información.

Word2Vec se basa en la idea de que las palabras que aparecen en contextos similares tienen significados similares. El algoritmo utiliza un modelo de aprendizaje no supervisado para aprender los patrones de co-ocurrencia de palabras en un corpus y generar representaciones vectoriales densas para cada palabra.

Hay dos arquitecturas principales utilizadas en *Word2Vec*: *Skip-gram* y Continuous Bag-of-Words (*CBOW*). Ambas arquitecturas involucran una red neuronal de una capa oculta.

- **Skip-gram:** En esta arquitectura, el modelo intenta predecir las palabras de contexto (palabras cercanas) dado una palabra de entrada. El objetivo es maximizar la probabilidad de predecir correctamente las palabras de contexto. Esto permite que el modelo capture las relaciones entre la palabra de entrada y sus palabras vecinas.
- **CBOW:** En esta arquitectura, el modelo intenta predecir la palabra de entrada a partir de las palabras de contexto que la rodean. El objetivo es maximizar la probabilidad de predecir correctamente la palabra de entrada. Esto permite que el modelo capture el significado de una palabra basado en el contexto en el que se encuentra.

[3]Word2vec - Tensorflow