Informe de segunda Tarea

Asignatura:
Introducción a la Ciencia de Datos

Grupo 3

Julio 2024, Montevideo, Uruguay

Integrantes
Dominguez, Leandro
Romani, Tiziana

Indice

Introducción

El proyecto propuesto en esta tarea continúa el análisis iniciado en la Tarea 1, utilizando el mismo dataset basado en la obra completa de William Shakespeare. La tarea implica la aplicación de técnicas avanzadas de análisis de texto y aprendizaje automático, utilizando nuevamente el entorno de Python y Jupyter Notebook.

La primera parte de la tarea se enfoca en la representación numérica de los textos. Esto incluye la creación de un dataset reducido que contiene ciertos personajes, la limpieza de los datos con la función *clean_text()* desarrollada en la Tarea 1, y la división del dataset en conjuntos de *entrenamiento* y *testeo* mediante muestreo estratificado. Posteriormente, se transformarán los textos a representaciones numéricas utilizando técnicas como Bag of Words y TF-IDF, y se realizarán visualizaciones para analizar el balance y la distribución de los datos utilizando PCA y TSNE.

La segunda parte del trabajo se dedica al entrenamiento y evaluación de modelos de clasificación de texto. Se entrenará un modelo Multinomial Naive Bayes y se evaluará su desempeño utilizando métricas como precision y recall además de visualización de matriz de confusión. Además, se explorarán técnicas de validación cruzada para la optimización de hiper-parámetros y se compararán diferentes modelos de aprendizaje automático para la clasificación de párrafos. Finalmente, se discutirá la efectividad de estos modelos en el análisis de texto y se evaluarán técnicas alternativas de extracción de features más complejas que incluyen mecanismos de atención.

Dataset y representación numérica de texto

En esta sección se procede a reducir el dataset únicamente a los 3 personajes requeridos por la letra. Se observa que quedan 626 filas restantes, quedando disponibles los siguientes personajes con sus filas: Anthony (253), Cleopatra (204), Queen Margaret (169).
Puede apreciarse en el siguiente gráfico más claramente cómo las clases están relativamente balanceadas:

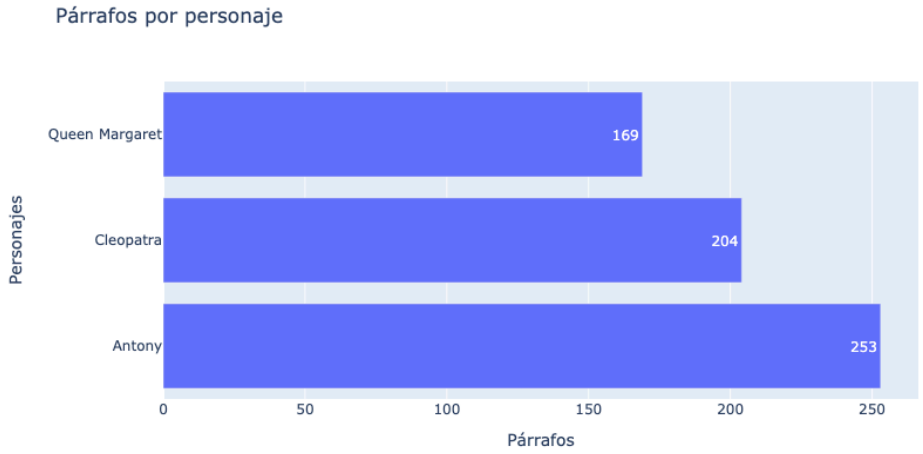

Figura 1: contiene la cantidad de párrafos por personaje.

A continuación se procede a realizar la división en train/test previo a comenzar con los diversos procesos de entrenamiento y uso de técnicas. Haciendo uso del método *train_test_split* dentro de la biblioteca *sklearn* se divide el conjunto de datos con una proporción del 0.7, 0.3, estratificando sobre los personajes. El muestreo estratificado asegura que cada uno de los personajes esté representado con la misma proporción en ambos conjuntos, mejorando así la representatividad y precisión de las estimaciones. Haciendo uso de una *seed*, la *randomicidad* de la función pasa a ser reproducible cada vez que se corre el código en cuestión.

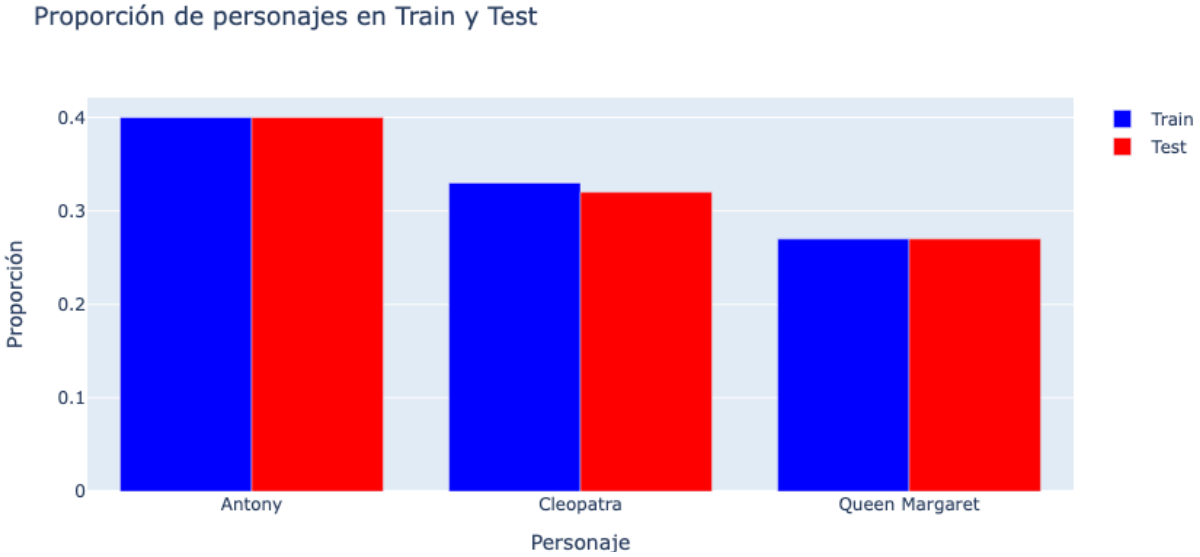

Figura 2: contiene la visualización de la proporción de los personajes sobre los conjuntos de entrenamiento y test.

En la *Figura 2* es posible visualizar cómo la proporción para Anthony así como Queen Margaret es exactamente la misma dentro de ambos conjuntos, mientras que para cleopatra difiere mínimamente (un 0.1). Esta diferencia en las proporciones pueden ser resultado del redondeo y la indivisibilidad de las observaciones individuales durante el muestreo estratificado al asignarse un número entero de observaciones a cada conjunto.

Representación vectorial de texto

Se introducen algunos conceptos que se utilizaron a lo largo de la tarea para representar numéricamente textos. Esto es necesario para que los mismos sean entrada de cualquier algoritmo ya que los mismos trabajan con números y no textos.

Bag of Words

Esta es una técnica empleada para representar texto basándose en la frecuencia de las palabras que contiene. Previo a representar los documentos se debe generar un vocabulario o vector de dimensión base donde cada posición se asocia a una palabra del vocabulario. Luego, para representar un documento, se debe contabilizar cuántas veces ocurre la palabra modificando (o no) cada posición del vector. A continuación se presenta un ejemplo.
En este caso se utilizará como vocabulario únicamente las palabras *left, come & speak*. Por lo que el vector representativo tendrá dimensión 3, donde la primera posición hará referencia al token *left*, la segunda posición a *come* y la última a *speak*.
Haciendo uso de la función *CountVectorizer* de *sklearn*, se calculan los vectores para el conjunto de entrenamiento ya preprocesado, obteniendo las siguientes representaciones:

speak	[0 0 1]
pardon pardon	[0 0 0]
richard	[0 0 0]
moreov hath left walk privat arbour new plant orchard side tiber hath left heir ever common pleasur walk abroad recreat caesar come anoth	[2 1 0]

Tabla 1: contiene la representación vectorial de los párrafos seleccionados utilizando un vocabulario reducido

Algo que se destaca inmediatamente de este ejemplo es que un vocabulario incompleto potencialmente lleve a representaciones idénticas entre entradas diferentes, perdiendo mucha información.

La matriz resultante tiene dimensiones *N x size(vocabulary)*, siendo *N* la cantidad de ejemplos en el conjunto de datos. Cada vez que un token no aparece en la oración, se incluye un 0 en la posición asignada. Es posible observar cómo, a menos que la completitud de entradas hagan uso de todo el vocabulario (que no es usual), una gran cantidad de posiciones serán 0. Esto define una matriz dispersa (*sparse matrix*), donde la mayoría de sus entradas son 0.

Con esta forma de representar los datos, podemos ver lo ineficiente que podría llegar a ser el almacenado si no se utilizara una forma eficiente para hacerlo. Supongamos un conjunto de 50.000 palabras, y 1.000.000 de documentos. Asumiendo que cada entrada de la matriz ocupa 4 bytes (int32), esto daría un total de ≈186 GB de tamaño en memoria.
Para evitar este problema, la librería utilizada (*sklearn*) utiliza los tipos Compressed Sparse Row, lo que solamente guarda las coordenadas de cada elemento distintos de cero junto con su valor.

N-gramas

Los n-gramas son secuencias de *n* palabras (o tokens) que se utilizan en el procesamiento del lenguaje natural para captar contextos más amplios que una única palabra en los textos. En muchos casos, dos palabras juntas tienen un significado distinto que dos por separado, como lo es “machine learning” y “machine” “learning”. En muchos casos mejora la representación de los embeddings.

Algunos ejemplos en el conjunto de datos podrían ser

- **Unigrama (1-gram):** Secuencias de una sola palabra. Ejemplo: "good", "sir", "lord".
- **Bigrama (2-gram):** Secuencias de dos palabras. Ejemplo: "good sir", "my lord".
- **Trigrama (3-gram):** Secuencias de tres palabras. Ejemplo: "Romeo and Juliet".

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) es una técnica utilizada en el procesamiento de lenguaje natural para evaluar la importancia de una palabra en un documento dentro de una colección de documentos y se compone de dos partes:

- **TF (Frecuencia de Término):** Mide cuántas veces aparece una palabra en un documento. Más apariciones significan mayor relevancia dentro de ese documento.
- **IDF (Frecuencia Inversa de Documentos):** Mide cuán común o rara es una palabra en todos los documentos de la colección. Las palabras que aparecen en muchos documentos tienen un IDF bajo.

La fórmula TF-IDF combina estas dos medidas para dar más peso a las palabras que son importantes en un documento pero no comunes en todos los documentos, ayudando a identificar términos relevantes y reducir la influencia de palabras muy comunes como podrían ser las stopwords.

En nuestro caso restringido a los tres personajes en train, contamos con un vocabulario de 9010 palabras y 438 documentos, lo que daría una matriz de TF-IDF con 3946380 elementos, pero sólo se guardan 13715. Los elementos no nulos representan un total del 0.35% del total.

Visualizaciones

PCA

PCA (Análisis de Componentes Principales) es una técnica utilizada para reducir la cantidad de variables en un conjunto de datos mientras se conserva la mayor cantidad de información posible. Es útil cuando se trabaja con datos de alta dimensión y se desea visualizar estos datos en 2D o 3D.
En nuestro caso, los vectores resultantes de TF-IDF tienen dimensión 9010, lo que resulta imposible visualizar. A continuación se grafican los embeddings con sus dimensiones reducidas a dos utilizando PCA para las 4 posibles combinaciones de IDF True y False y Bi-gramas y sin N-Gramas:

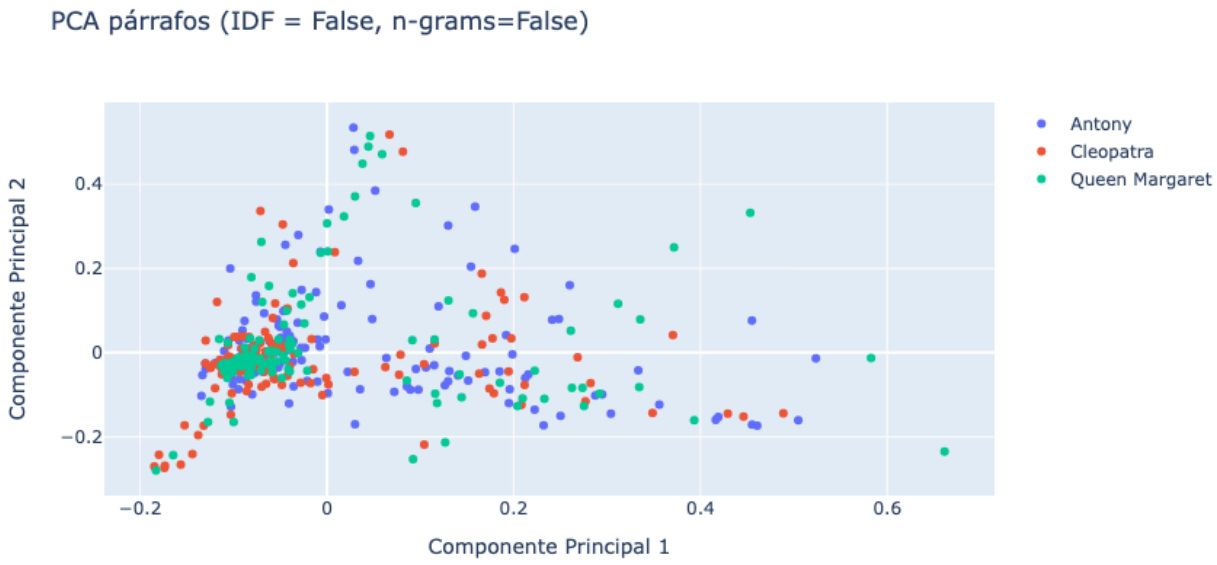

Figura 3: contiene los vectores TF, con n=1 para n gramas

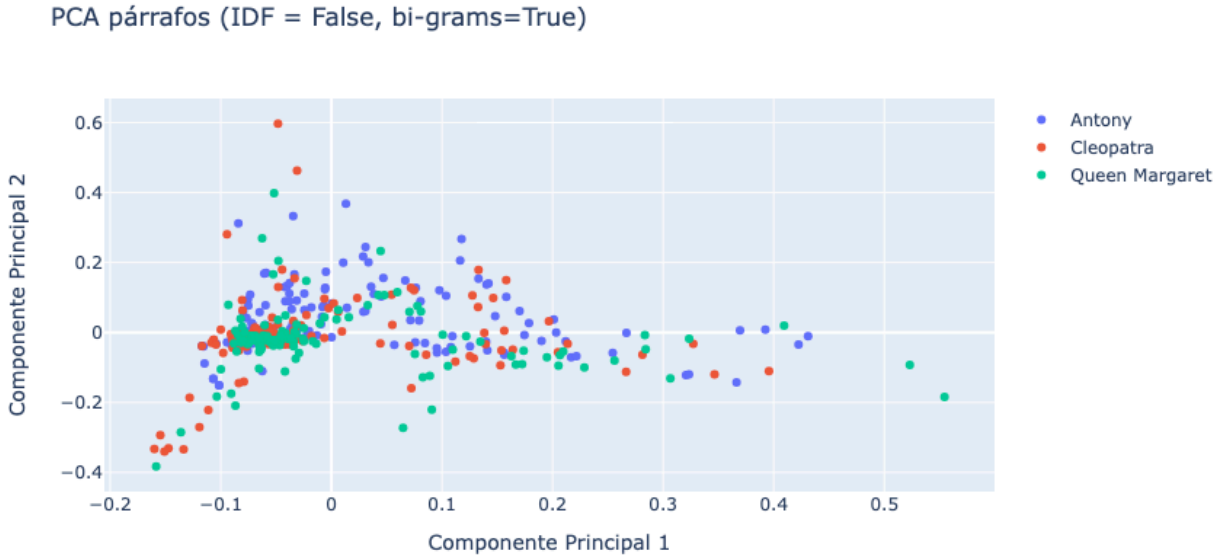

Figura 4: contiene los vectores TF, con n=2 para n gramas.

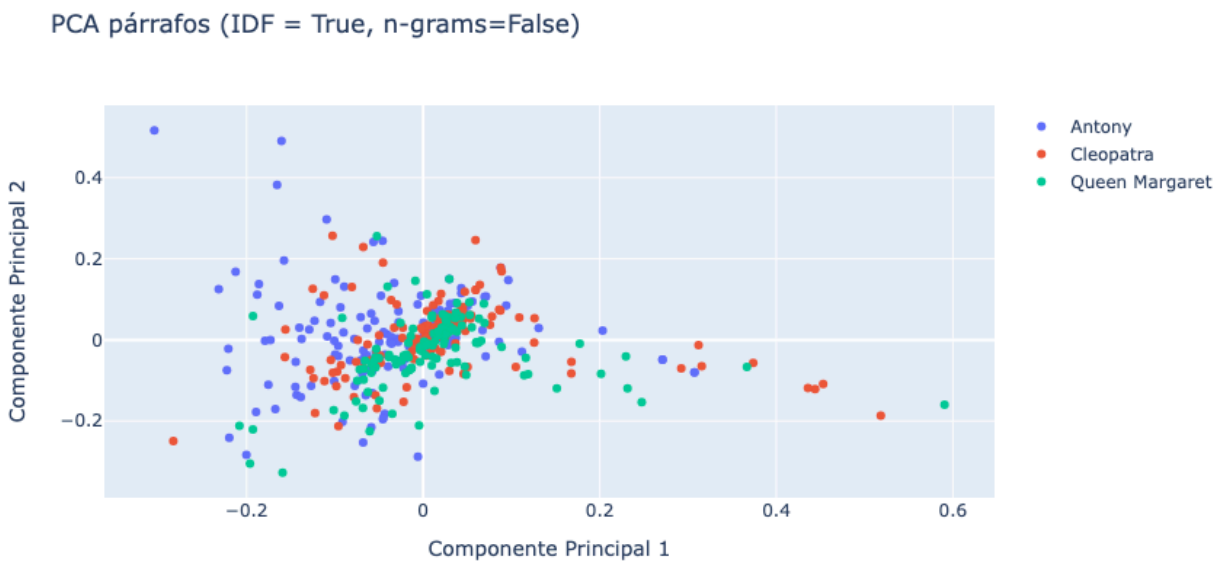

Figura 5: contiene los vectores TF-IDF, con n=1 para n gramas

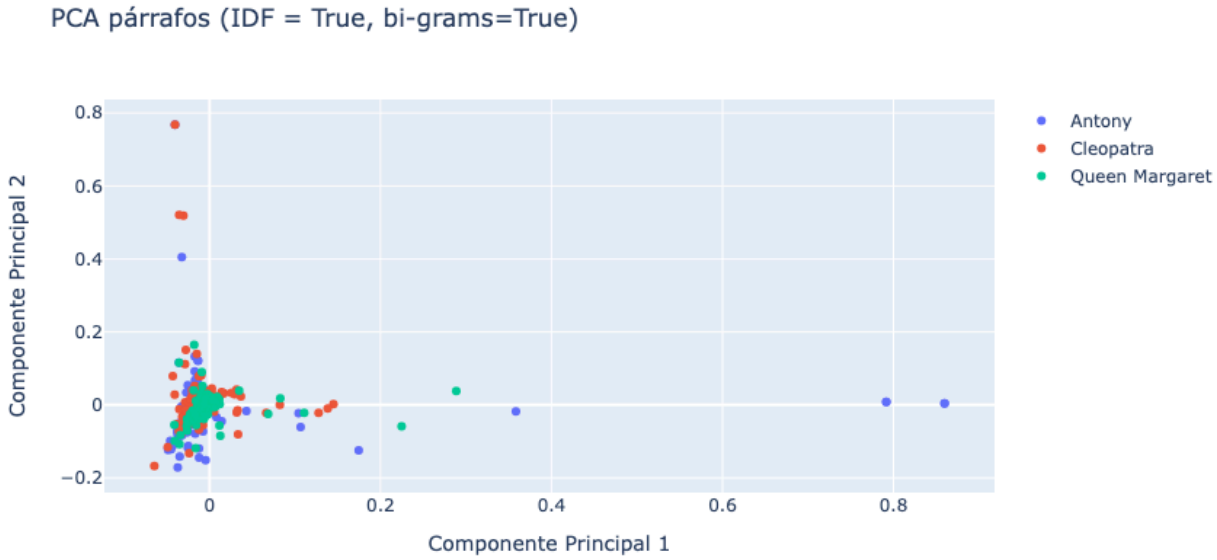

Figura 6: contiene los vectores TF-IDF, con n=2 para n gramas

En ninguna de las cuatro combinaciones es posible separar los personajes de una forma clara. Contra nuestra intuición, la combinación IDF=True y utilizando bigramas, parece ser la representación que parece dividir de peor forma los párrafos de los personajes.

PCA en 3 Dimensiones

Se intentó graficar en 3 dimensiones para analizar si los datos podían ser mejor divididos ahí pero tampoco fue posible, eso puede observarse en la gráfica a continuación

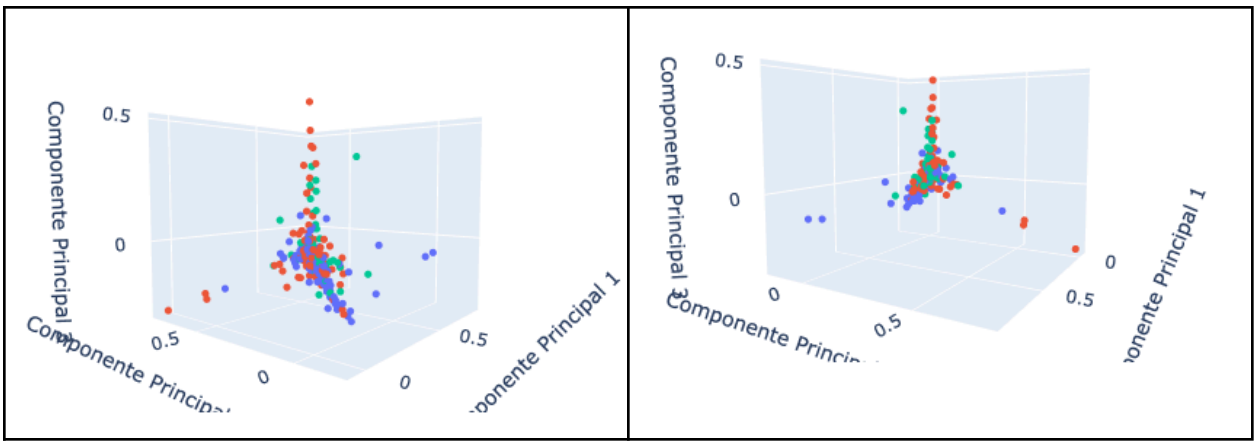

Figura 7: contiene los vectores TF-IDF, con n=2 para n gramas, en 3 dimensiones

Varianza explicada

La varianza explicada es la cantidad de varianza que una componente puede explicar: entre más grande sea más información contiene.
Se corre el PCA sobre el conjunto de datos y se grafica la varianza explicada para cada componente extra que se suma y se obtiene el siguiente gráfico:

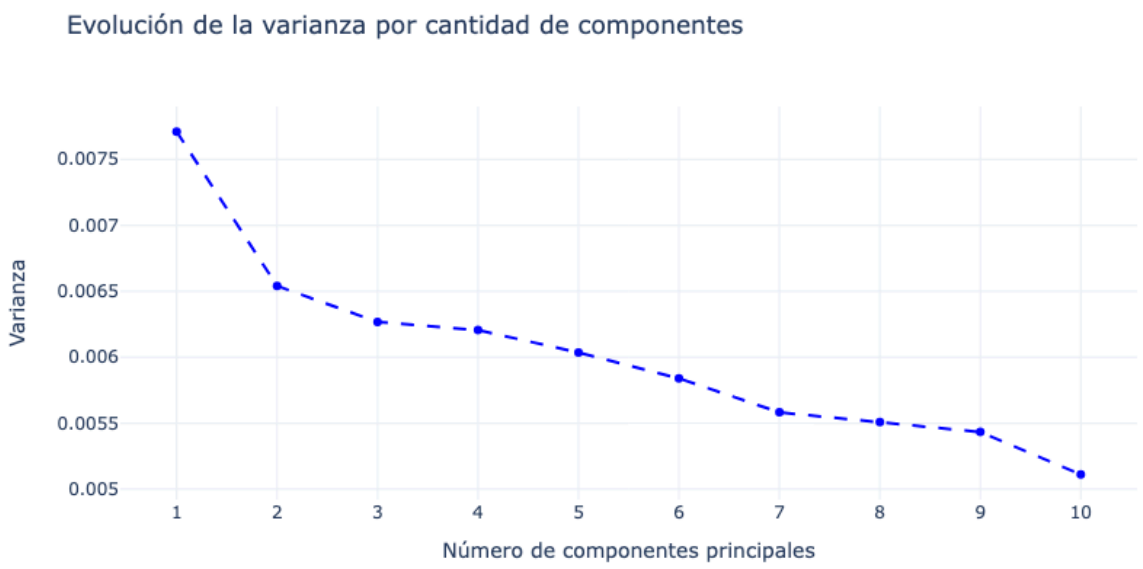

Figura 8: contiene como evoluciona la varianza al agregar componentes extras en PCA

Como primera conclusión del gráfico, se observa que la varianza explicada es casi nula desde el primer componente y tiende a cero cuando aumenta el número de componentes. Si se suma el valor para los primeros 10 componentes, apenas se alcanza un 3% de varianza explicada. Esto quiere decir que por más que aumentemos el número de componentes, es muy difícil explicar la varianza de los datos utilizando este método.

Varianza explicada acumulativa

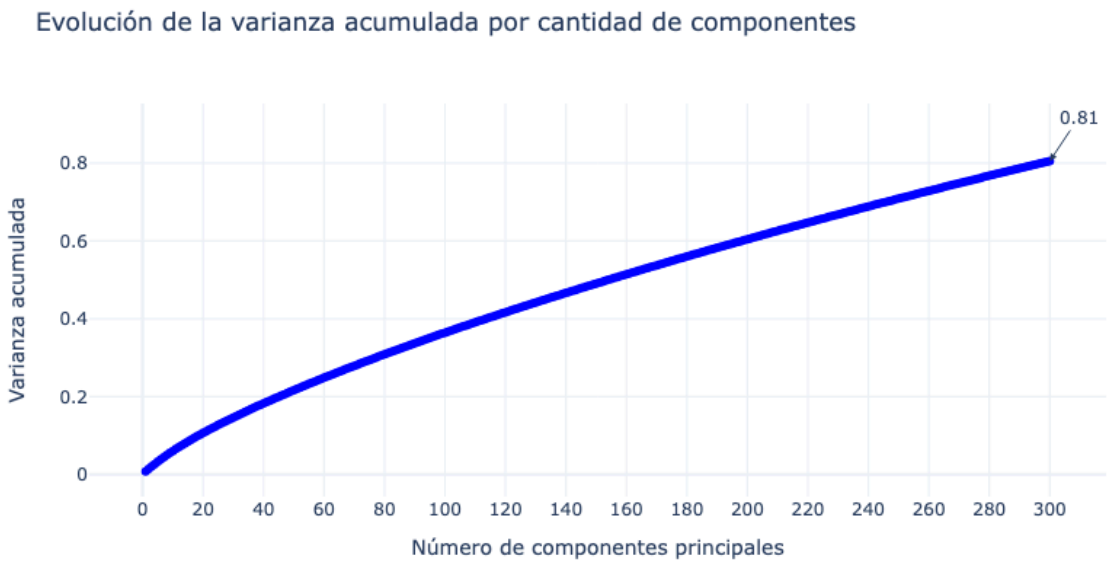

Figura 9: contiene como evoluciona la varianza acumulativa al agregar componentes extras en PCA

Del gráfico de varianza explicada acumulativa, podemos ver que recién utilizando 300 componentes logramos obtener un 81% de la varianza del conjunto original explicada por las componentes. Esto habla de lo complejo o ruidosos que pueden ser los datos.

T-SNE

Se utilizó TSNE para comparar contra PCA y comprender si los datos podían ser divididos de mejor forma, pero tampoco se obtuvieron buenos resultados.
T-SNE (t-distributed Stochastic Neighbor Embedding) es otra técnica de reducción de dimensionalidad. A diferencia de PCA, se basa en preservar la estructura local de los datos, es decir, mantiene cerca en el espacio reducido los puntos que son vecinos en el espacio original.

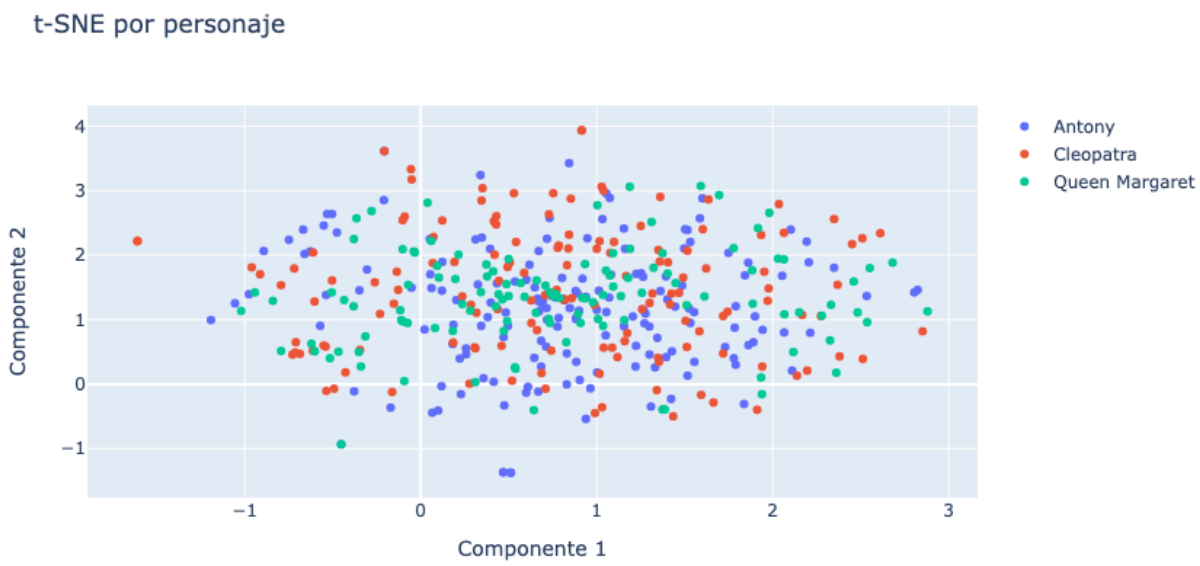

Figura 10: contiene la reducción a 2 componentes con t-SNE para los vectores TF-IDF

Entrenamiento y evaluación de Modelos

Se entrena un clasificador Multinomial Naive Bayes con los parámetros por defecto, luego se prosigue a calcular precisión sobre los conjuntos de train y test, obteniendo **97,9%** de precisión para train y **52,1%** de precisión para test.

Al observar estos valores se podría concluir que simplemente está sucediendo *overfitting*, pero dado que este es un problema multiclase, la precisión no es representativo de qué tan bien funciona el modelo. A continuación se prosigue a incluir *precision* y *recall* de cada clase.

	precision	*recall*
Anthony	0.47	0.95
Cleopatra	0.58	0.23
Queen Margaret	1	0.24

Tabla 2: contiene las métricas de precision y recall para cada clase al clasificar con el algoritmo Multinomial Naive Bayes

La *precision* nos dice qué tan “preciso” es el modelo al clasificar una clase, o sea si se equivoca categorizando algo como positivo, cuando no lo es.

- Se observa como para *Anthony*, el valor de *precision* es menor que 50%, esto indica que más de la mitad de las veces predice algo como *Anthony* que no es.
- Para *Cleopatra* este valor indica que al menos alrededor del 40% de las veces predice esta categoría cuando no lo es.
- Sin embargo, para *Queen Margaret*, no hay predicciones incorrectas.

El *recall* nos dice que tan certero es el modelo al momento de identificar todas las instancias de una clase.

- Para *Anthony* el modelo identifica correctamente la mayoría del total de ejemplos del conjunto de datos.
- Para *Cleopatra* este valor es menor que un tercio.
- Por otro lado, para *Queen Margaret*, se acerca al 25% de los ejemplos totales.

Otra opción es observar la matriz de confusión, que puede ayudar a dar una noción de si las predicciones se centran alrededor de una clase, o si hay dos clases en particular que el modelo confunde.

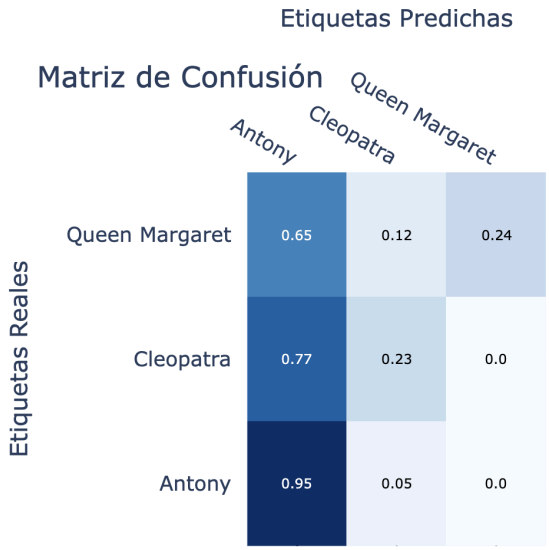

Figura 11: contiene la matriz de confusión para la predicción del conjunto de test utilizando el algoritmo Multinomial Naive Bayes

Dados los valores visibles en la primera columna, se concluye que el modelo está prediciendo la categoría *Anthony* en la mayoría de ejemplos. Esto sucede en parte debido al desbalance inicial del conjunto de datos, el modelo tiende a aprender a predecir esa categoría para minimizar el *loss* o error.

Este es un típico problema que se origina en un desbalance del conjunto de datos, a pesar de que el desbalance no es tan grande, igualmente condiciona al modelo a predecir la clase que más se repite para minimizar el error durante el entrenamiento. Esto es especialmente agravante si el problema de clasificación es de carácter crítico, como por ejemplo la detección de un tumor, donde el 99% de los casos predice negativo pero el 1% debería predecir positivo, si es especialmente importante que el recall sea alto.

Para corregir los problemas de desbalance, existen principalmente dos técnicas relacionadas al conjunto que podrían ser utilizadas: undersampling y oversampling. Oversampling es una técnica que replica ejemplos de las clases minoritarias para que tengan más peso en el entrenamiento del modelo y undersampling, por el contrario, consiste en tomar una muestra aleatoria de un porcentaje del conjunto original de una clase para balancear las clases. Ambas

técnicas pueden utilizarse en conjunto para no eliminar tantos datos con undersampling o para evitar generar muchos datos que podrían no aportar información nueva en el oversampling.

Por último, es posible configurar la *loss* de un modelo para que no todas las clases aporten lo mismo a esta métrica, pudiendo penalizar más a los errores de las clases minoritarias.

Cross validation

La técnica de validación cruzada o cross validation, es una técnica que se utiliza para evaluar la performance de un modelo sobre un conjunto de datos. Se divide el conjunto de datos en N particiones, luego, en N iteraciones, se entrena el modelo con un subconjunto y se evalúa con el subconjunto restante de datos. Luego se realiza un promedio sobre las métricas de validación para obtener un estimativo robusto de la performance.

Se realiza la técnica de validación cruzada con 4 splits y para cada iteración se consideran las siguientes 16 combinaciones de hiperparámetros (que no son todas las posibles):

```
  {"stop_words": None, "ngram": (1,2), "idf": True},
  {"stop_words": None, "ngram": (1,1), "idf": False},
  # Incluir stop words
  {"stop_words": 'english', "ngram": (1,2), "idf": True},
  {"stop_words": 'english', "ngram": (1,1), "idf": False},
  # Excluir stop words custom
  {"stop_words": stop_words, "ngram": (1,2), "idf": True},
  {"stop_words": stop_words, "ngram": (1,3), "idf": True},
  {"stop_words": shakespeare_stop_words, "ngram": (1,2), "idf": True},
  {"stop_words": shakespeare_stop_words, "ngram": (1,3), "idf": True},
  # Testear diferentes N para los n gramas
  {"stop_words": 'english', "ngram": (1,3), "idf": True},
  {"stop_words": 'english', "ngram": (1,4), "idf": True},
  # Probar eliminando el parámetro smooth idf
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "smooth_idf": False},
  # Testear diferentes alphas para Naive Bayes
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.5},
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.1},
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.05},
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.01},
  {"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.001},
```

Y finalmente se obtiene que el mejor modelo (con una precisión de 0.67) es el que utiliza los siguientes hiperparámetros:

```
{"stop_words": 'english', "ngram": (1,3), "idf": True, "alpha": 0.05}
```

Comparación de métricas de modelos

Mediante una gráfica de violines se comparan las métricas de los distintos modelos resultantes de combinar los diferentes hiperparámetros:

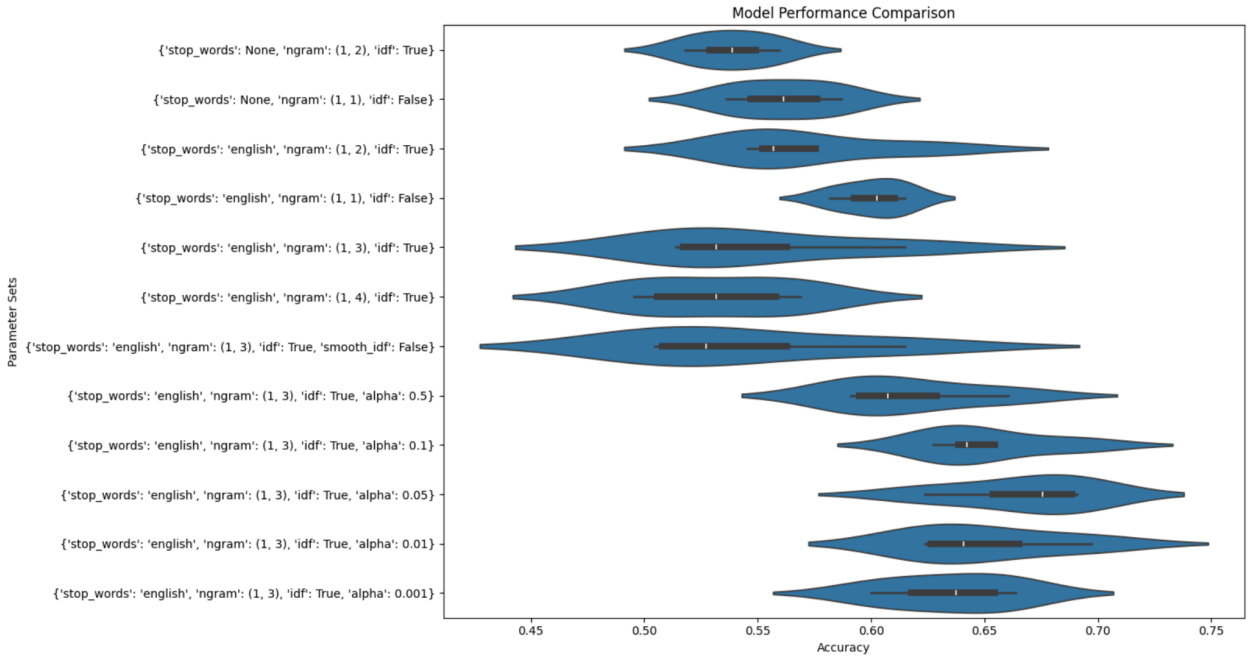

Figura 12: contiene la comparación de las gráfica de violines para cada una de las combinaciones de parámetros en Naive Bayes

Como dato anecdótico, de la gráfica anterior puede observarse que la penúltima combinación de hiperparámetros uno de los 4 splits mejora la performance del modelo ganador (alrededor de 0.74), pero el promedio de las precisiones termina siendo peor.

Análisis del modelo ganador

A continuación se entrenó un modelo con los hiperparámetros ganadores y se obtiene uno con precisión **98,2%** en el conjunto de train y **58,5%** en el conjunto de test, lo que representa una suba de **6,5** puntos porcentuales frente al modelo que utiliza los parámetros por defecto. Aunque mejore la métrica, también puede afirmarse que existe un gran *overfitting* por la diferencia de precisión en entrenamiento y test.
La siguiente matriz compara el nuevo modelo (*new*) con el anterior (*old*):

	precision_new	*recall_new*	*precision_old*	*recall_old*
Anthony	**0.55**	**0.70**	0.46	0.91
Cleopatra	**0.65**	**0.39**	0.61	0.33
Queen Margaret	**0.61**	**0.65**	0.86	0.37

Tabla 3: contiene la comparación de las métricas entre mejores parámetros o parámetros por defecto

y a continuación se dibujan las matrices de confusión:

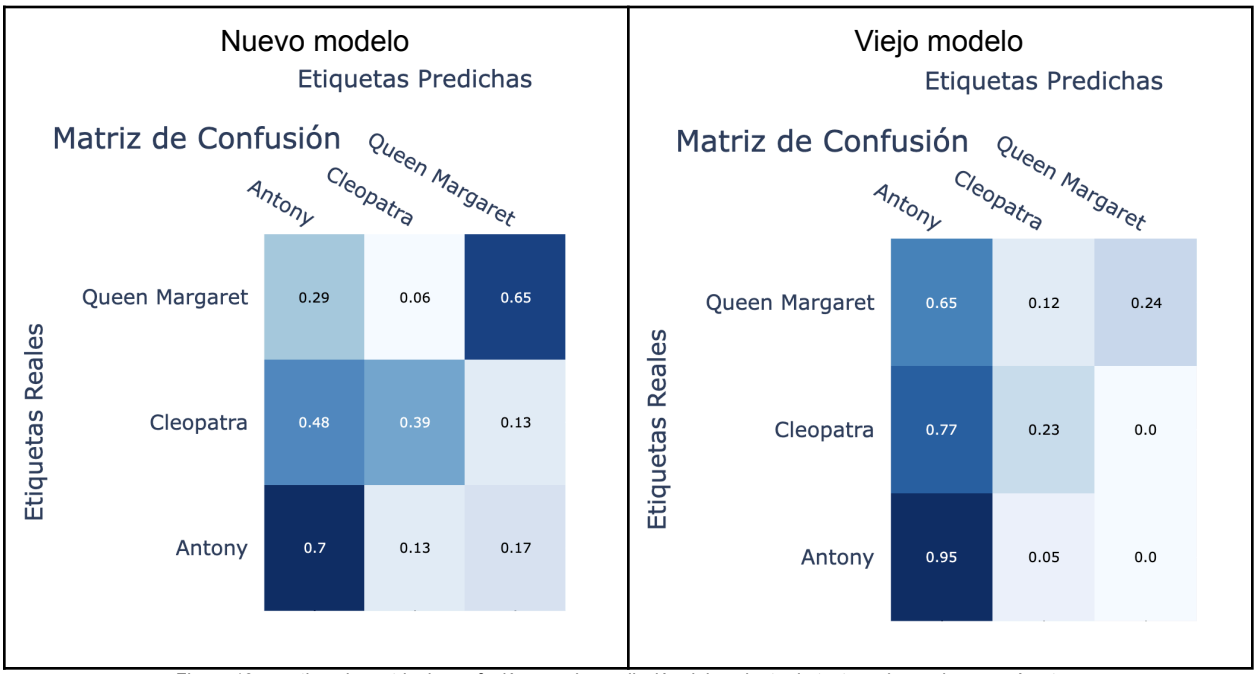

Figura 13: contiene la matriz de confusión para la predicción del conjunto de test con los mejores parámetros
Figura 14: contiene la matriz de confusión para la predicción del conjunto de test con los parámetros por defecto

En general se observan mejores resultados “sacrificando” recall de Antony. El modelo anterior casi que predecía todas las clases como Antony a diferencia del actual que sigue teniendo una tendencia a predecir a dicho personaje pero acierta más a los demás en general.

Limitaciones de los modelos utilizados

Los modelos utilizados son básicos y se han utilizado durante mucho tiempo pero cuentan con grandes limitaciones que se detallan:

- **Ignoran contexto, semántica y orden:**
  Tanto TF-IDF como Bag of Words no consideran el significado de las palabras ni el contexto en el que se utilizan. Ambos métodos tratan las palabras de manera aislada, sin tener en cuenta las relaciones semánticas o gramaticales entre ellas o el orden. Esto puede llevar a representaciones de texto que no capturen el significado de las frases. Por ejemplo, las frases "Hamlet mata a Claudio" y "Claudio mata a Hamlet" serían tratadas de manera similar, a pesar de tener significados opuestos.
- **Dimensionalidad:**
  Como se analizó previamente, a medida que el tamaño del vocabulario crece, las representaciones de texto basadas en TF-IDF y Bag of Words pueden volverse extremadamente grandes y dispersas. Esto resulta en matrices de alta dimensionalidad que pueden ser difíciles de manejar computacionalmente.
- **Poca flexibilidad frente a cambios en el corpus:**
  Como las matrices se calculan inicialmente con el corpus dado, tanto las ocurrencias como las frecuencias son para un corpus en particular. Si se agregaran nuevos documentos (y con ellos palabras) deberían recalcularse todas las matrices nuevamente.

KNN

KNN o K Nearest Neighbor es un método de aprendizaje supervisado utilizado tanto para problemas de clasificación como de regresión. Su funcionamiento se basa en encontrar los K ejemplos más cercanos en el espacio de características y hacer predicciones basadas en ellos. Por ejemplo para clasificar se puede tomar la clase que más aparezca entre estas K, mientras que para regresión se puede tomar un promedio de los valores.

Se prueban los K entre 1 y 30, obteniendo que el mejor K es 29 con un valor de **0.56** para accuracy. Esto es apenas peor que el modelo ganador obtenido anteriormente con Naive Bayes.

Nuevo set de personajes

A continuación se seleccionan 3 personajes de la obra Hamlet que tienen un desbalance considerable entre ellos:

- *Hamlet*: El príncipe de Dinamarca, conocido por sus mensajes filosóficos y su lenguaje introspectivo. Su discurso a menudo contiene metáforas complejas y reflexiones profundas sobre la existencia y la moralidad.

- *Polonius*: El consejero del rey, conocido por su verborragia y consejos prácticos. Polonius tiende a hablar en prosa y su estilo es más directo y prolijo, con un tono a menudo moralizante.
- *Laertes*: El hijo de Polonius, quien tiene un estilo más apasionado y directo. Sus diálogos suelen ser menos introspectivos que los de Hamlet y más enfocados en la acción y la justicia, especialmente en el contexto de la venganza por la muerte de su padre.

Se visualiza el desbalance de antemano, que resulta ser bastante mayor que el de la tripla anterior de personajes.

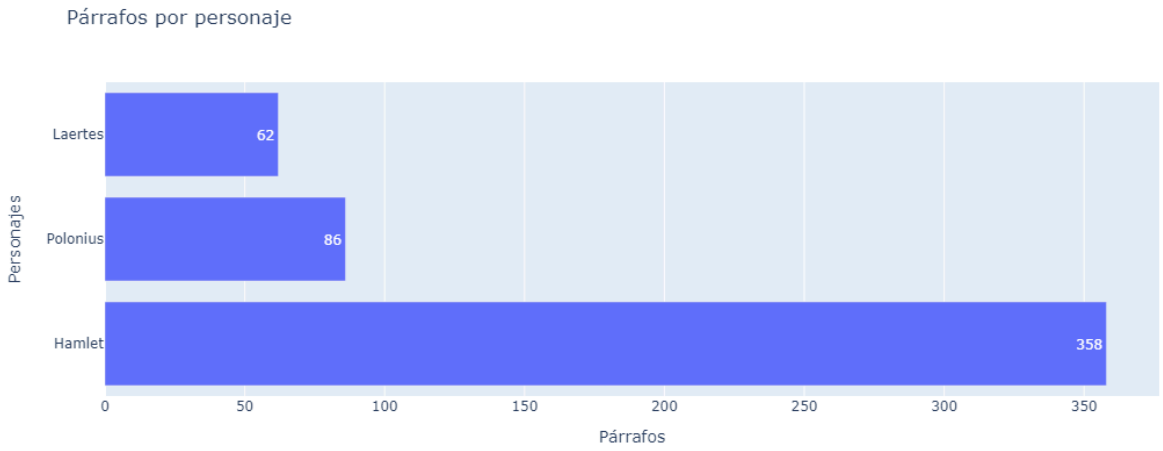

Figura 15: contiene la visualización de la proporción de diálogo de los personajes

Se prueba nuevamente con KNN y siguiendo con las líneas de conclusión anterior, el “mejor” clasificador es aquel que aprende a predecir una única clase. Pues esto minimiza el error.
A continuación se visualiza esta conclusión en la matriz de confusión.

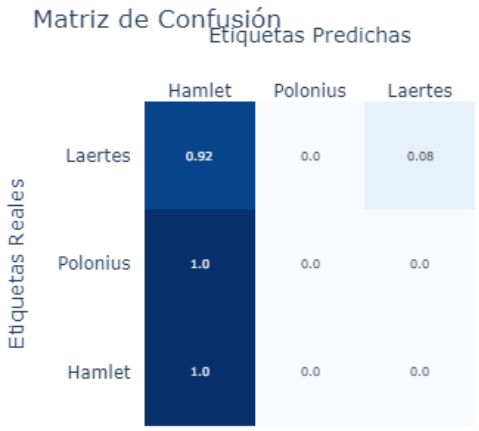

Figura 16: contiene la matriz de confusión para el clasificador KNN de la nueva tripla de personajes

Para solucionar el problema de la mala clasificación debido al desbalance en los datos, existen técnicas de sobremuestreo y submuestreo que pueden ser utilizadas para aumentar o reducir los ejemplos de cierta categoría obteniendo .

- *Oversampling (o sobremuestreo):* Consiste en aumentar la cantidad de muestras de la o las clases minoritarias para equilibrar el conjunto de datos. Para esto se pueden usar técnicas como replicación de datos, o de creación de datos sintéticos.
- *Undersampling (o submuestreo):* De forma similar, se realiza una reducción de la cantidad de muestras de la o las clases mayoritarias. Esto se puede lograr eliminando aleatoriamente muestras de la clase mayoritaria hasta alcanzar un balance con la clase minoritaria.

Estas técnicas pueden ayudar a mejorar la capacidad de generalización del clasificador al proporcionar un conjunto de datos más equilibrado. Aunque aplicarlas puede traer riesgos. Realizar un sobremuestreo sobre clases minoritarias, puede llevar a que el clasificador aprenda o se sesgue sobre ciertos atributos y no generalice bien sobre ejemplos no vistos. También la fabricación de datos puede introducir variabilidad que no existe en la realidad.

De forma similar, para el submuestreo, al eliminar ejemplos de la clase mayoritaria puede llevar a una pérdida de información y que el muestreo restante no represente correctamente a la clase o incluso que el clasificador se incline hacia la/las clases minoritarias. Esto empeoraría la calidad del modelo y sus predicciones .

Word Embeddings

Los Word Embeddings son una representación vectorial de palabras en un espacio de alta dimensión, aunque mucho menor que los vectores utilizados en esta tarea. Estos vectores tienen varias diferencias con los de bag of words. Por ejemplo, los embeddings son vectores densos (no contienen casi 0s), mientras que los de *bag of words* son *sparse* (muchos de sus elementos son 0).

Otra diferencia clave es que los Word Embeddings capturan el contexto en el cual las palabras aparecen. Esto termina teniendo propiedades semánticas dentro del espacio vectorial, por ejemplo que palabras con significados similares se mapean a vectores cercanos en el espacio vectorial. Esto también permite que los modelos puedan generalizar sobre datos no vistos mucho mejor que al utilizar *bag of words* pues aunque no hayan aprendido la palabra pueden hacer uso del contexto.

Los modelos que producen Word Embeddings se entrenan utilizando grandes corpus de texto, donde haya gran cantidad de ejemplos o extractos de lenguaje. Entre los algoritmos más famosos se encuentra Word2Vec o FastText, que es una extensión del anterior. Extiende el algoritmo anterior considerando las distintas componentes de cada palabra al crear el vector, mientras que el anterior toma en cuenta la palabra entera. Esto le permite a FastText capturar mejor la morfología de las palabras y comprender palabras nuevas ayudándose de sus componentes.

Fasttext

FastText no solo se utiliza para la generación de embeddings de palabras, sino que también es posible utilizarlo para la clasificación de texto mediante el algoritmo de regresión logística múltiple. Cada extracto de texto se representa como una suma de los vectores de palabras y sub-palabras que lo componen. El modelo se entrena utilizando un corpus etiquetado, donde cada extracto de texto tiene una etiqueta asociada.

Los datos se deben formatear de la siguiente manera:
__label__<etiqueta> <texto>
A continuación se presentan ejemplos de cómo quedaría el conjunto de datos:
__label__Antony let fellow take reward say god quit familiar playfellow hand kingli seal plighter high heart upon hill
__label__Queen_Margaret leav stay dog shalt hear heaven grievou plagu store exceed wish upon let keep till thi sin
__label__Cleopatra prais antoni disprais caesar

Una vez entrenado, el modelo puede asignar etiquetas a nuevos textos basándose en las representaciones vectoriales aprendidas. La accuracy general para el conjunto de test da un valor de **61%**. Esto es casi **2%** más alto que el mejor modelo de Naive Bayes, lo cual no parece ser una gran mejoría. A continuación se incluye la matriz de confusión para una mejor comprensión de las métricas.

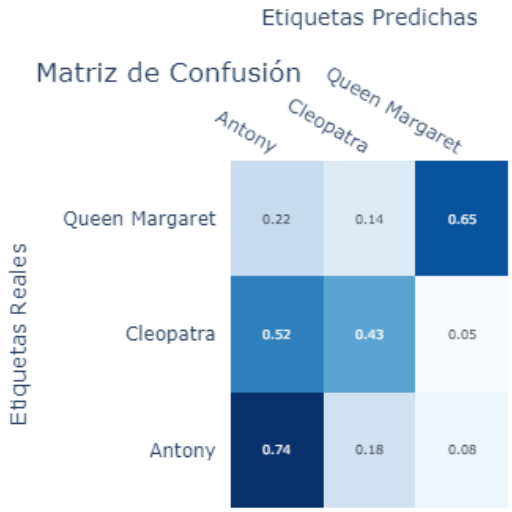

Figura 17: contiene la matriz de confusión para Fasttext con la tripla original de personajes

La mejoría es pequeña, se puede notar que la cantidad de veces que predice correctamente para las clases Anthony y Cleopatra incrementa, mientras que para Queen Margaret se mantiene. Este modelo tiende a confundir un poco más los diálogos de Cleopatra como si fueran de Anthony. Para la tabla de métricas:

	precision_FT	*recall_FT*	*precision_NB*	*recall_NB*
Anthony	**0.57**	**0.74**	0.55	0.70
Cleopatra	**0.55**	**0.43**	0.65	0.39
Queen Margaret	**0.79**	**0.65**	0.61	0.65

Tabla 4: contiene la comparación de las métricas de test entre los algoritmos FastText y Naive Bayes

Aquí es posible notar como con la excepción de la precisión para Cleopatra, el resto de las métricas en general mejora para todos los personajes. Esto se relaciona con lo mencionado de que este clasificador tiende a confundir los diálogos de Cleopatra como si fueran de Anthony, reduciendo la precisión sobre esa clase al predecir.

Como ventajas el algoritmo de FastText se destaca en varios puntos.
Primero, el manejo de palabras nuevas y entendimiento de la morfología del texto, debido a que descompone palabras en sus componentes, puede entender palabras que no aparecieron previamente en el corpus de entrenamiento, a diferencia de utilizar *bag-of-words* por ejemplo, que no tiene manera de manejar palabras no vistas previamente.
Luego se debe destacar la simpleza de su implementación y eficiencia computacional al compararlo con modelos más complejos de embeddings que requieren mucho más poder computacional para poder entrenar. Es posible entrenar FastText rápidamente y de forma simple.

Por otro lado, se debe mencionar que la calidad de los embeddings y predicciones dependen de la calidad del conjunto de datos. Esto es una acotación general para cualquier situación y cualquier modelo, la diferencia es que al estar trabajando con texto, lograr conseguir un corpus completo y de calidad puede ser muy dificultoso, pues el lenguaje puede ser muy extenso. También puede llegar a ser demandante en términos de almacenamiento o memoria, debido al manejo de sub-palabras.

Por último FastText no tiene en cuenta el orden de las palabras en el contexto de una oración. Se enfoca en representar palabras y sub-palabras, pero no en cómo las palabras se organizan secuencialmente. Modelos como GPT y BERT utilizan arquitecturas de *transformers* donde consideran el orden de las palabras, poniendo atención basada en el contexto, esto permite identificar relaciones sintácticas y una mejor comprensión del texto.

Alternativas para extraer features

Modelo de transformers (mecanismo de atención)

Los transformers son una arquitectura de redes neuronales usada en procesamiento de lenguaje natural que mejora considerablemente las técnicas previamente descriptas por principalmente dos razones:

1) Tienen el cuenta el orden de las palabras:
   Las arquitecturas de transformers tienen una codificación de posiciones de cada palabra de una oración que también se utiliza para generar el embedding, por lo que el orden de cada palabra es tenida en cuenta para generar el embedding de cada oración.
2) Cuentan con mecanismos de atención:
   Los transformers cuentan con un mecanismo de auto-atención que permite al modelo enfocarse en diferentes partes del texto mientras procesa cada palabra. Calcula cuánto debe "atender" a cada palabra en la secuencia, basándose en la relevancia de las palabras entre sí. Por esto es que los vectores de cada palabra varían según el contexto que la rodea.

Embeddings de ChatGPT

Se utiliza la API de ChatGPT para construir embeddings para cada uno de los párrafos de los personajes inicialmente utilizados (Anthony, Cleopatra y Queen Margaret) para comparar la calidad de estos embeddings contra los generados por TF-IDF. En este caso, el modelo utilizado genera vectores de magnitud 1536, lo que representa un sexto de las dimensiones de los vectores generados por el algoritmo inicial, que tenía (9010). Sin embargo se verá que estos vectores tienen información de mayor calidad en sus coordenadas.

Proyección PCA

Se realiza la proyección PCA en dos componentes para visualizar los vectores:

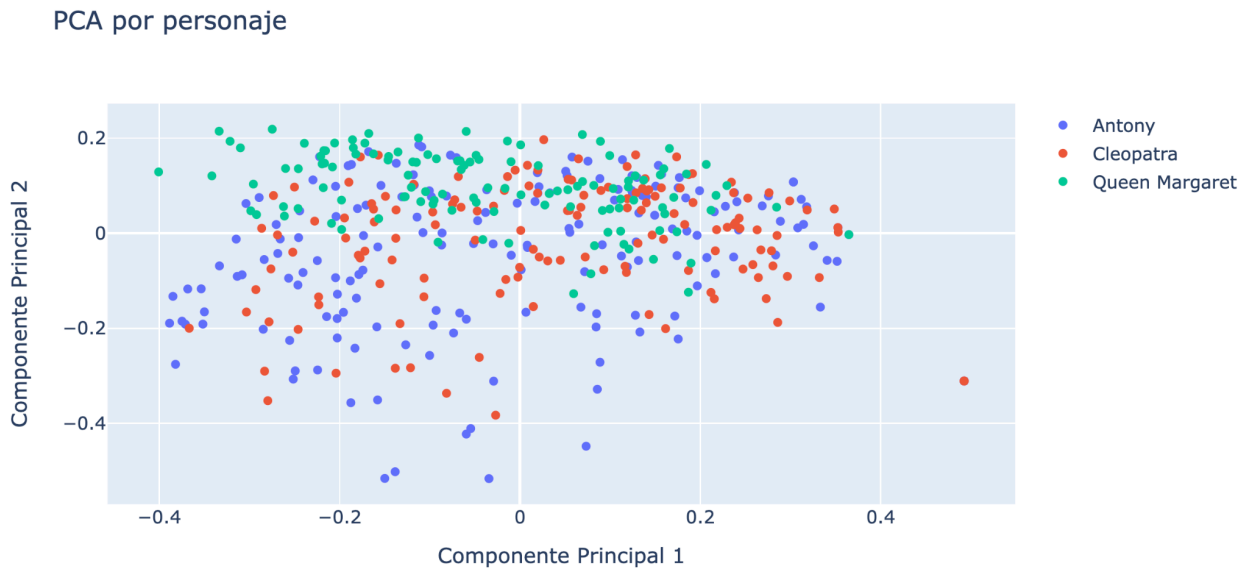

Figura 18: contiene la reducción a 2 componentes conPCA para los embeddings de OpenAI

Puede apreciarse que hay cierta estructura, donde por ejemplo los párrafos de Queen Margaret tienden a estar concentrados en un área aproximada determinadas por las rectas { x=-0,4, x=0.2, y=0.2, y=0 }.

Explicación de la varianza acumulada por cada componente

Se grafica la varianza acumulada para compararla con la de la correspondiente a los vectores generados con TF-IDF, donde se necesitaban 300 componentes para llegar a un 81% de la varianza original.

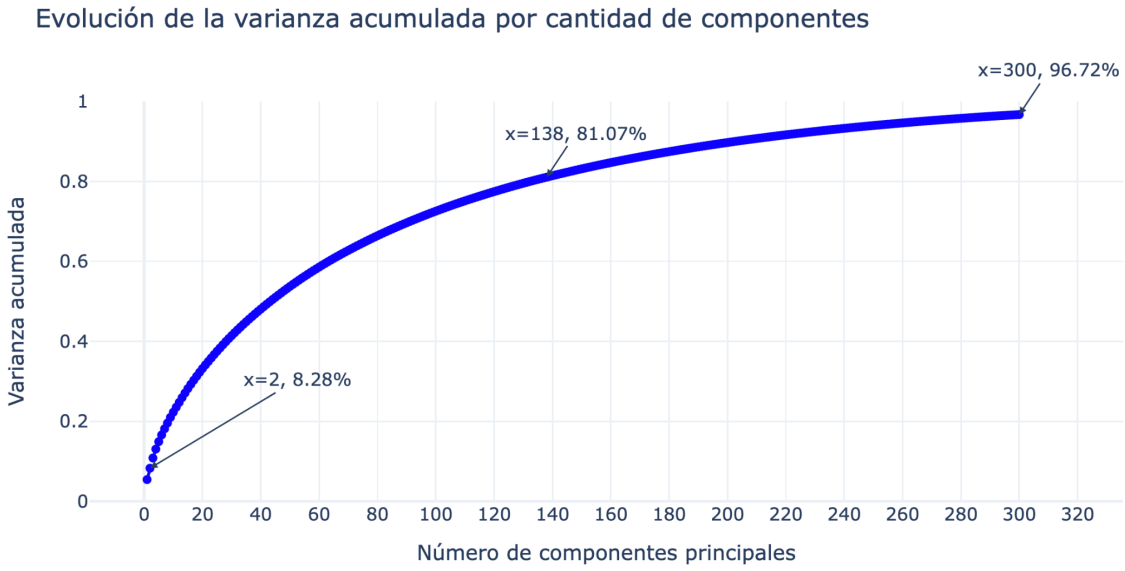

Figura 19: contiene la varianza acumulada por componente al reducir mediante PCA los embeddings de OpenAI

En este caso podemos ver cómo utilizando sólo 2 componentes, podemos explicar más de un 8% de la varianza frente a un 0.65% que se podía ver con las dos componentes proyectadas de TF-IDF. Además, para alcanzar el 81% de la varianza que se explicaba con 300 componentes en TF-IDF, se necesita menos de la mitad (138). Por último, podemos apreciar que con 300 componentes llegamos a un poco más de 96% de la varianza del conjunto de embeddings de chatGPT.

Proyección TSNE

En el gráfico que continúa puede verse la proyección en dos dimensiones de los embeddings generados por chatGPT utilizando la técnica de TSNE:

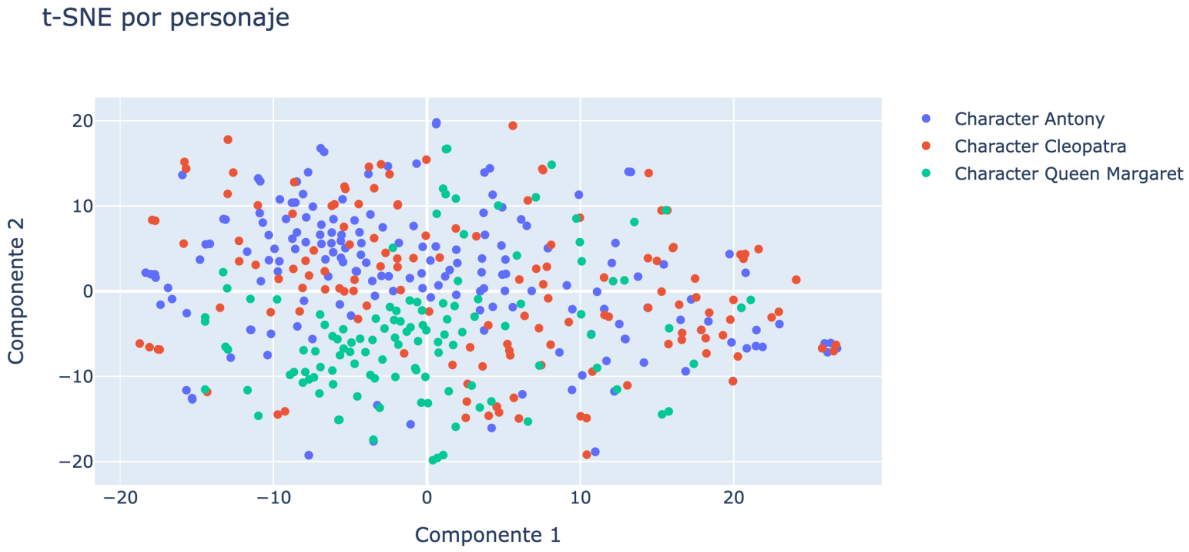

Figura 20: contiene la reducción a 2 componentes con t-SNE para los embeddings de OpenAI

Aquí también pueden verse algunas tendencias a agruparse en determinadas zonas, lo que daría a pensar que con un algoritmo simple como KNN se podría obtener una precisión no despreciable.

Entrenamiento de KNN

Se entrena un modelo KNN con K=3 con los vectores obtenidos de chatGPT y se obtiene una precisión de train del **77,3%** y de test de **64%**, mejorando 5,5 puntos porcentuales con respecto al modelo entrenado con los vectores de TF-IDF y MNB.

Comparaciones de métricas

	Precision MNB con TFIDF	*Recall MNB con TFIDF*	*Precision KNN con chatGPT*	*Recall KNN con chatGPT*
Anthony	0.55	0.70	**0.61**	**0.83**
Cleopatra	0.65	0.39	**0.56**	**0.41**
Queen Margaret	0.61	0.65	**0.85**	**0.65**

Tabla 5: contiene la comparación de las métricas de los vectores TF-IDF + Naive Bayes y los embeddings de OpenAI + KNN

y a continuación se dibujan las matrices de confusión:

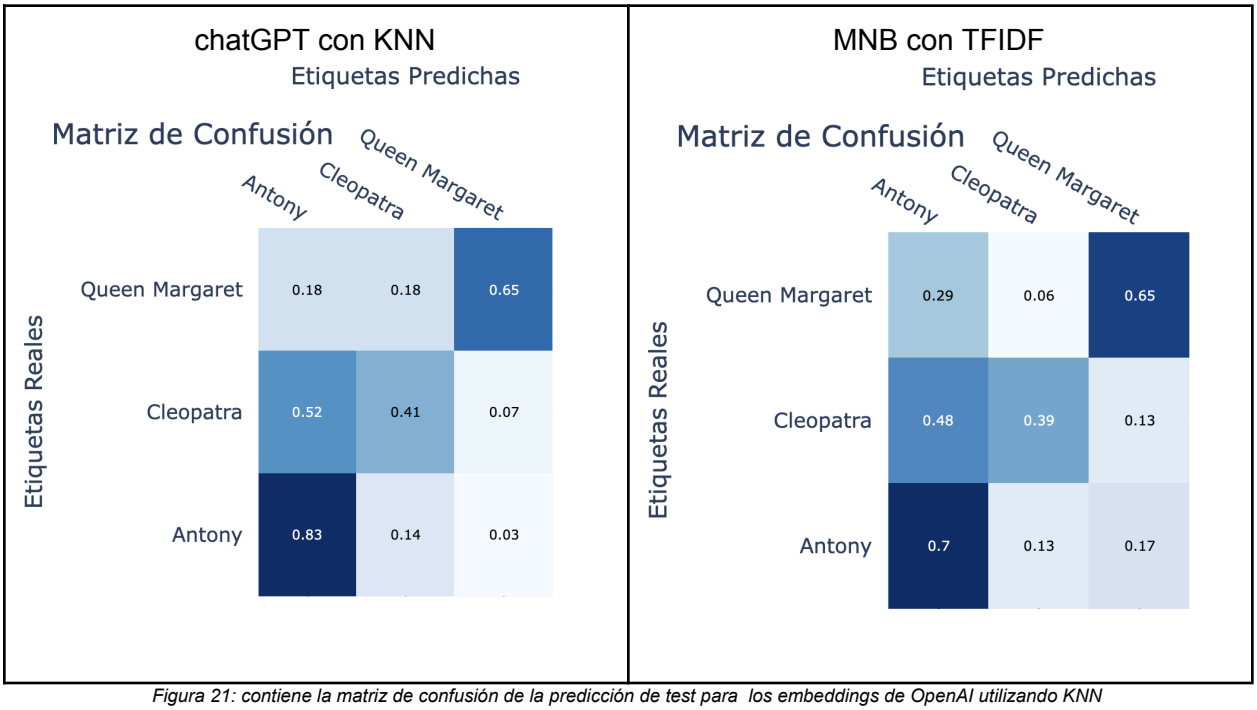

Figura 21: contiene la matriz de confusión de la predicción de test para los embeddings de OpenAI utilizando KNN
Figura 20: contiene la matriz de confusión de la predicción de test para los vectores TF-IDF utilizando Naive Bayes

Salvo la precisión de la clase Cleopatra, todas las métricas incluidos el recall y la precisión general del modelo mejoran. Esto nos habla de cómo los nuevos embeddings capturan los significados de los párrafos con mayor precisión y por lo tanto permiten dividir las clases de forma más certera.