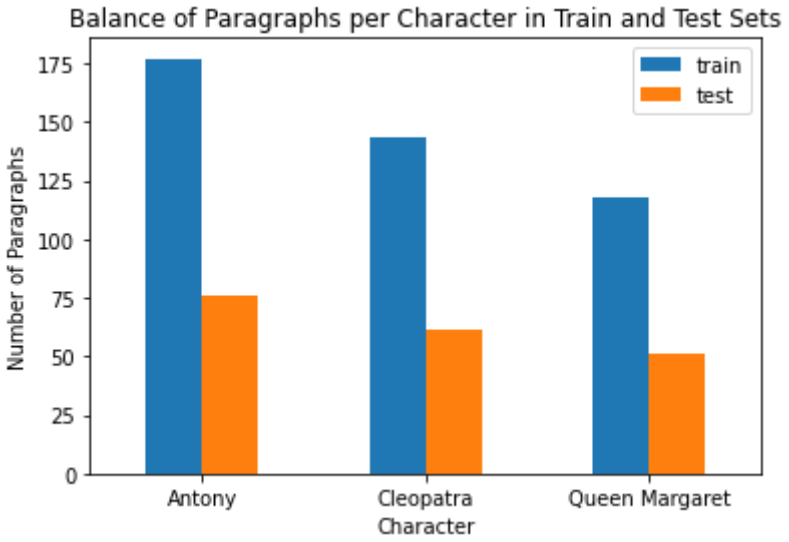
TAREA 2 - ICD - 2024

Gastón Acosta - Martín Sacco García

PARTE 1:

Apartados 1 y 2:

Como resultado de la sección 1 y 2, debajo se presenta la visualización obtenida que permite verificar el balance entre los conjuntos Train y Test entre los personajes, pudiendo confirmar que no se encuentran balanceados dado que de estarlo, el tamaño de los conjuntos (Train y Test) debería tener alturas similares entre los personajes:



Dentro de los resultados que se podrían considerar como relevantes para el apartado de la Sección 1, los mismos estarían asociados a los tamaños del conjunto de entrenamiento y del conjunto de testeo, siendo los mismos los detallados a continuación y estando en coherencia con el tamaño del conjunto de testeo definido:

- Tamaño del conjunto de entrenamiento: 438
- Tamaño del conjunto de testeo: 188

Apartado 3:

La técnica de "Bag of Words" convierte un conjunto de documentos de texto en una matriz de características numéricas. En esta representación, cada documento es convertido en un vector de frecuencia de palabras. A continuación se presentan los pasos principales asociados a la técnica y una breve explicación de cada uno de ellos:

1. **Tokenización**: Se divide el texto en palabras individuales (tokens), generando un vector donde cada palabra toma una ubicación en el mismo (con una fila y tantas columnas como palabras contenía el texto).
2. **Construcción del vocabulario con una Matriz de Documentos-Términos:** Se crea un vocabulario de todas las palabras únicas en el conjunto de párrafos a través de la generación de una matriz, donde las filas corresponden a los párrafos y las columnas corresponden a una palabra única que se identifica en el conjunto de datos.
3. **Conteo de ocurrencias o de frecuencias:** Se cuentan la cantidad de veces que cada palabra del vocabulario aparece en cada documento, generando ahora una matriz donde cada columna existente representa un recuento de una palabra específica de acuerdo a la construcción del vocabulario del paso anterior y la cantidad de filas representa el total de textos o documentos analizados (en nuestro caso de párrafos).

Como generalización, de la matriz resultante de los pasos previos será posible observar que el valor en la celda (i, j) de la matriz es el número de veces que la palabra j aparece en el documento i.

La matriz resultante tendrá dimensiones (n_samples, n_features), donde:

- n_samples: es el número de documentos, donde para nuestro caso los documentos corresponden a cada párrafo del conjunto de entrenamiento a transformar.
- n_features es el tamaño del vocabulario (el número de palabras únicas en todos los párrafos).

La matriz resultante es una "sparse matrix" (matriz dispersa) porque la mayoría de los valores en la matriz son ceros. En grandes conjuntos de datos, cada documento (o párrafo en nuestro caso) suele contener sólo una pequeña fracción de todas las palabras posibles en el vocabulario construido, lo que resulta en una matriz con muchos ceros. Cabe destacar que esta condición genera mayor eficiencia en términos de almacenamiento y uso de memoria RAM, dado que solo almacenan las ubicaciones y los valores de los elementos distintos de cero, que en conjuntos de datos no tan reducidos (a diferencia de los utilizados para el ejercicio) esto pasa a ser algo más relevante para el almacenamiento y computación.

A modo de ejemplo se detalla el paso a paso de la técnica teniendo en cuenta los siguientes dos párrafos:

- Párrafo 1: “Vamos la celeste del alma.”
- Párrafo 2: “Uruguay campeón de la Copa América 2024”

Resultado de ejecución de Tokenización:

- Párrafo 1: ["Vamos", "la", "celeste", "del", "alma"]
- Párrafo 2: ["Uruguay", "campeón", "de", "la", "Copa", "América", "2024"]

Resultado de la Construcción del Vocabulario:

- Palabras únicas: ["Vamos", "la", "celeste", "del", "alma", "Uruguay", "campeón", "de", "Copa", "América", "2024"]

Resultado de la Matriz de Conteo:

Párrafo	**Vamos**	**la**	**celeste**	**del**	**alma**	**Uruguay**	**campeón**	**de**	**Copa**	**América**	**2024**
Párrafo 1	1	1	1	1	1	0	0	0	0	0	0
Párrafo 2	0	1	0	0	0	1	1	1	1	1	1

Las dimensiones se corresponden a 2 filas (1 por cada párrafo) y 11 columnas (tantas como palabras únicas fueron identificadas durante la construcción del vocabulario).

Corriendo la técnica en el conjunto de entrenamiento resultante de la sección 1 y 2:

- Cantidad de filas = 438, coincidiendo con el número de párrafos del tamaño de entrenamiento.
- Cantidad de columnas = 2807, dando el indicio de que se identificaron un total de 2807 palabras únicas durante la etapa de construcción de vocabulario para el conjunto de entrenamiento seleccionado (teniendo en cuenta también que se seleccionó un n-gram (1,1)).
- Cantidad de elementos no nulos en la matriz = 10831, confirmando que efectivamente la cantidad de elementos nulos (438*2807=1.229.466

elementos) supera ampliamente este número, dando coherencia al concepto de que se está ante una sparse matrix.

Apartado 4:

Un n-grama en el contexto del procesamiento de lenguaje natural es una secuencia contigua de n elementos de una cadena de texto. Los elementos pueden ser caracteres, palabras o incluso símbolos más grandes. Los n-gramas se utilizan para capturar la estructura local del texto y son útiles en tareas como modelado de lenguaje, traducción automática y extracción de información. Los n-gramas son capaces de capturar el contexto local y las relaciones entre las palabras en un texto, siendo esto especialmente valioso en tareas de NLP como la predicción de palabras, la clasificación de textos, el análisis de sentimientos, y más. Los modelos de lenguaje basados en n-gramas consideran las probabilidades de ocurrencia de secuencias de palabras para predecir o analizar el texto.

A continuación se detallan algunos ejemplos de tipos de n-gramas:

- **Unigramas (1-gramas)**: Secuencias de 1 elemento. Por ejemplo, en la oración "La casa es linda", los unigramas son: "La", "casa", "es", "linda".
- **Bigramas (2-gramas)**: Secuencias de 2 elementos. Por ejemplo, en la oración "La casa es linda", los bigramas son: "La casa", "casa es", "es linda".
- **Trigramas (3-gramas)**: Secuencias de 3 elementos. Por ejemplo, en la oración "La casa es linda", los trigramas son: "La casa es", "casa es linda".

TF-IDF de sus siglas en inglés "Term Frequency-Inverse Document Frequency" es una técnica utilizada en el procesamiento de texto y minería de datos para evaluar la importancia de una palabra o término de un documento, dentro de un conjunto de documentos. TF-IDF es una técnica que combina dos conceptos:

1. **TF (Term Frequency)**: Es la frecuencia de un término (palabra) en un documento. Se calcula como el número de veces que el término aparece en el documento dividido por el número total de términos en el documento. La idea es que las palabras que aparecen con más frecuencia en un documento sean más importantes.

$$TF(t,d) = \frac{Número\ de\ veces\ que\ el\ término\ t\ aparece\ en\ el\ documento\ d}{Número\ total\ de\ términos\ en\ el\ documento\ d}$$

2. **IDF (Inverse Document Frequency)**: Es una medida de la importancia de un término en el conjunto de documentos (corpus). Se calcula como el logaritmo del número total de documentos dividido por el número de documentos que

contienen el término. La idea es que las palabras que son comunes en muchos documentos sean menos importantes.

$$IDF(t) = log(\frac{N}{1 + Número\ de\ documentos\ que\ contienen\ el\ término\ t})$$

Siendo:

$N = número\ total\ de\ documentos\ en\ el\ llamado\ corpus$

Se suma 1 en el denominador para evitar divisiones por cero en caso de que un término no aparezca en ningún documento.

La fórmula de TF-IDF combina ambas medidas para valorar la importancia de un término en un documento dentro de un corpus. Se calcula multiplicando TF e IDF de la siguiente manera:

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

La matriz TF-IDF resultante tendrá dimensiones (n_samples, n_features), donde:

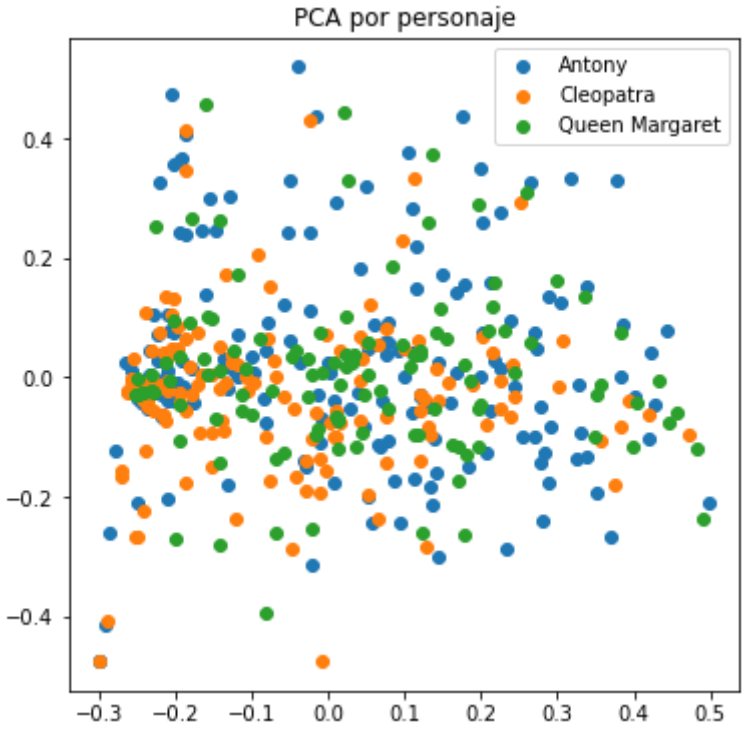
- n_samples es el número de documentos.
- n_features es el tamaño del vocabulario (el número de palabras únicas en todos los documentos).

Al igual que con la técnica de BoW, la matriz TF-IDF es una sparse matrix porque la mayoría de los valores en la matriz son ceros. Esto es debido a que en grandes conjuntos de datos, cada documento suele contener sólo una pequeña fracción de todas las palabras posibles en el vocabulario, lo que resulta en una matriz con muchos ceros. Como ya se mencionó en el apartado 3, las matrices dispersas son más eficientes en términos de almacenamiento y computación porque solo almacenan las ubicaciones y los valores de los elementos no cero.

Corriendo la técnica en la matriz resultante de la sección 3, dado que no se utiliza el término IDF (tf_idf = TfidfTransformer(use_idf=**False**)), solo se genera la transformación de la matriz a los correspondientes valores TF de cada elemento. En función de esto, las dimensiones resultantes son las mismas a la sección 3 (438 filas, 2807 columnas y 10837 elementos no nulos), ahora con los valores TF para cada elemento.

Apartado 5:

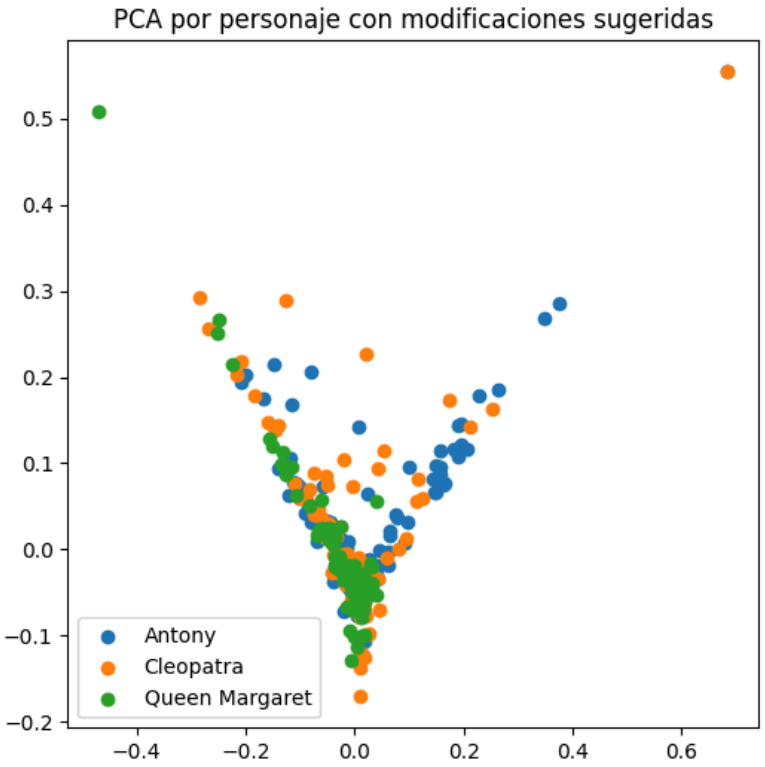
Debajo se detalla la visualización resultante de utilizar PCA sobre los vectores de TF-IDF para el conjunto de entrenamiento:



Surge de inmediato, dado el resultado, que no se está logrando separar a los personajes a través del PCA. Esto sugiere que las características derivadas de los vectores TF-IDF para el conjunto de entrenamiento, tal como se definieron, no capturan de manera efectiva las variaciones únicas asociadas con cada personaje.

En el Análisis de Componentes Principales (PCA, por sus siglas en inglés) se busca reducir la dimensionalidad del set de datos de manera poder visualizarlo en 2D o 3D capturando la mayor variabilidad posible de los datos originales. Claramente la variabilidad capturada con las primeras dos componentes principales no permite separar los párrafos (vectores TF-IDF) correspondientes a cada personaje de modo que veamos en el gráfico de PCA tres “nubes” de puntos separadas correspondientes a cada personaje.

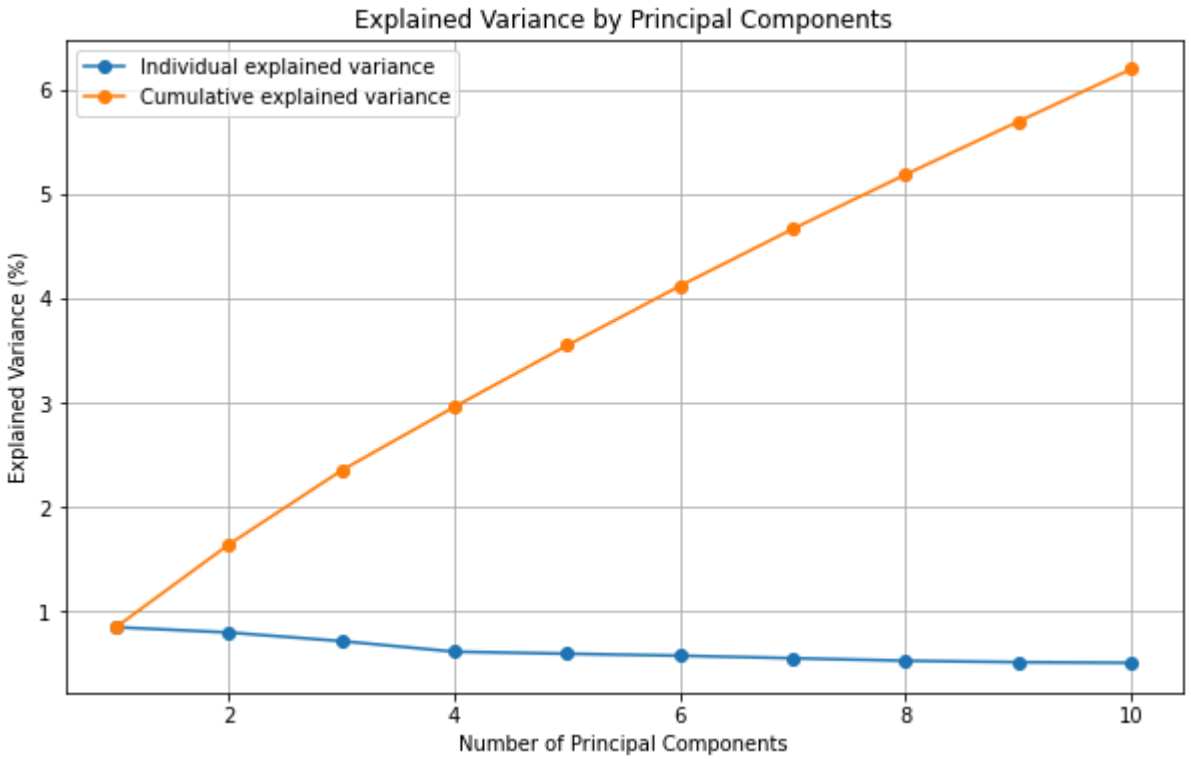
Realizando los cambios sugeridos, a continuación se presenta la visualización resultante con los mismos aplicados:



En este caso, las modificaciones sugeridas tienen un efecto visual positivo en lo que respecta a la disminución de la dispersión de los datos, logrando verse zonas más definidas. Sin embargo, a pesar de esto, se ve que los personajes no terminan de diferenciarse entre ellos de manera contundente.

En base a estos resultados, es posible concluir que no sería posible separar los personajes utilizando solo 2 componentes principales.

A continuación se presenta una visualización de cómo varía la varianza explicada a medida que se agregan componentes (hasta 10 componentes en este caso):



En base a los resultados, es posible confirmar que la variación de la varianza explicada a medida que se agregan componentes es mínima, y esto se puede explicar dado el número de datos manejados para el modelo.

Se esperaría observar una meseta en el aumento de la varianza acumulativa, en la medida que se alcance un número óptimo de componentes principales. Sin embargo, para esta cantidad de componentes, no se estaría alcanzando, y la explicación se podría dar por la gran cantidad de datos, como se mencionó previamente.

PARTE 2:

Apartado 1:

Antes de desplegar los resultados de la ejecución del entrenamiento, más los valores correspondientes, se pasará a generar una explicación teórica de cada uno de los conceptos manejados, para posteriormente aplicarlos al caso de estudio.

Matriz de Confusión:

La matriz de confusión es una herramienta que permite evaluar el rendimiento de un modelo de clasificación mostrando los conteos de predicciones correctas e incorrectas desglosadas por clase. A partir de la matriz de confusión, se pueden calcular varias métricas de rendimiento, incluyendo precisión (precision) y recall. Para profundizar en el cálculo de ambas métricas, se presenta a continuación un ejemplo básico de resultados y su interpretación conceptual:

		PREDICTED	
		Positive	**Negative**
ACTUAL	**Positive**	TP	FN
	Negative	FP	TN

- **TP (True Positive)**: Predicciones correctas para la clase positiva.
- **FP (False Positive)**: Predicciones incorrectas para la clase negativa donde se predice la clase positiva pero la clase real es negativa.
- **FN (False Negative)**: Predicciones incorrectas para la clase positiva donde se predice la clase negativa pero la clase real es positiva.
- **TN (True Negative)**: Predicciones correctas para la clase negativa.

Métrica de Precisión:

- **Precision (Precisión)**:
    - Mide la exactitud de las predicciones positivas.
    - Fórmula: Precision=TP/(TP+FP)

- Una alta precisión indica que hay pocos falsos positivos.

Métrica de Recall:

- **Recall (Sensibilidad o Tasa de Verdaderos Positivos)**:
  - Mide la capacidad del modelo para capturar todas las instancias positivas.
  - Fórmula: Recall=TP/(TP+FN)
  - Un alto recall indica que hay pocos falsos negativos.

Para una clasificación multiclase, la matriz de confusión se expande para manejar múltiples clases (no solo como positivos y negativos según se desarrolló en el ejemplo básico), y las fórmulas para precisión y recall se aplican de manera similar para cada clase individualmente.

En busca de complejizar el ejemplo y además aportar resultados numéricos, se detalla un modelo de 3 clases (más parecido al caso de estudio) para profundizar en los conceptos, suponiendo la siguiente matriz con sus resultados:

		PREDICTED		
		A	**B**	**C**
ACTUAL	**A**	50	2	3
	B	5	45	5
	C	3	2	50

Se calcula la precisión y el recall para cada una de las clases según lo detallado previamente:

- **Precision (clase A)**: TP**A**/(TP**A**+FP**A**) = 50/(50+5+3) = 50/58 = 0.86
- **Recall (clase A)**: TP**A**/(TP**A**+FN**A**) = 50/(50+2+3) = 50/55 = 0.90
- **Precision (clase B)**: TP**B**/(TP**B**+FP**B**) = 45/(45+2+2) = 45/49 = 0.92

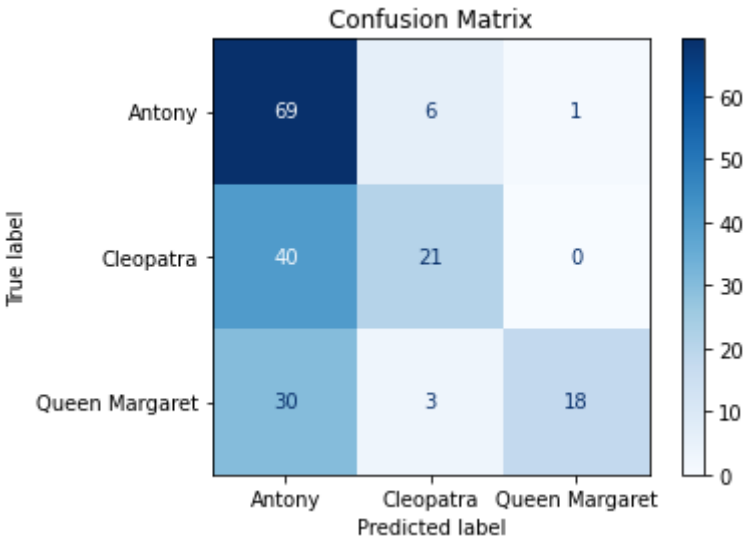
- **Recall (clase B)**: TP**B**/(TP**B**+FN**B**) = 45/(45+5+5) = 45/55 = 0.81
- **Precision (clase C)**: TP**C**/(TP**C**+FP**C**) = 50/(50+5+3) = 50/58 = 0.86
- **Recall (clase C)**: TP**A**/(TP**A**+FN**A**) = 50/(50+2+3) = 50/55 = 0.90

Generalizar este proceso para cada clase es posible obtener una tabla resumen con precisión, recall y F1-score (si es necesario) para todas las clases pudiendo de esa manera evaluar el modelo en general (que para el ejemplo en cuestión, el modelo es muy bueno para predecir todas las clases teniendo en cuenta los resultados de las métricas mencionadas).

Pasando al caso de estudio, habiendo explicado conceptualmente tanto el aporte de una matriz de confusión como las métricas a calcular para evaluar el modelo, se entrenó un modelo de Multinomial Naive Bayes utilizando la representación TD-IDF de los datos de entrenamiento.

Las predicciones sobre el conjunto de entrenamiento (train) mostraron un accuracy de 0.96, mientras que las predicciones sobre el conjunto de prueba (test) un valor de 0.57.

La matriz de confusión para el conjunto de prueba es la siguiente:



Y los valores de Precision, Recall, F1 Score y Support obtenidos para cada personaje (clase) son:

Character	Precision	Recall	F1 Score	Support
Antony	0,50	0,91	0,64	76

Cleopatra	0,70	0,34	0,46	61
Queen Margaret	0,95	0,35	0,51	51

En base a estos resultados y teniendo en cuenta los conceptos mencionados, es posible aventurar que el modelo es bueno en términos de Recall respecto a la predicción de Antony, dado que no genera una cantidad importante de Falsos Negativos (6+1). Sin embargo, en términos de Precisión, la misma baja considerablemente, dado que hay una gran cantidad de Falsos Positivos (40+30). Sucede algo que se podría considerar opuesto para Cleopatra y Queen Margaret, pudiendo aventurar un modelado preciso en función de la baja cantidad de Falsos Positivos (9 y 1 respectivamente), y un Recall bajo debido a la cantidad alta de Falsos Negativos (40 y 33 respectivamente). Se puede aventurar este resultado como consecuencia del desbalance inicial entre los conjuntos de entrenamiento y testeo para cada personaje.

Teniendo en cuenta la matriz de confusión resultante del conjunto de testeo, es posible identificar los factores que generan el valor de Precisión detallado más arriba resultando del siguiente cociente:

Precisión (test): (69+21+18)/(69+21+18+40+30+6+3+1+0)=108/188=**0.57**

Cabe destacar que el denominador corresponde a un total de 188 que es coherente con el tamaño del conjunto de testeo definido en el caso de estudio.

La precisión general (accuracy) puede ser engañosa en casos de desbalance de datos. Si una clase es mucho más frecuente que otras, un modelo puede tener alta precisión simplemente por predecir siempre la clase más frecuente.

En un escenario de mayor desbalance de datos, la métrica de precisión podría no reflejar correctamente el rendimiento del modelo en clases menos representadas.

Se incluyen también las métricas F1 Score y Support como adicionales a la Precisión y el Recall, siendo el F1 Score fruto del siguiente cociente:

$$F1\ Score\ =\ 2\ *\ \frac{Precisión+Recall}{Precisión*Recall}$$

Y el Support es fruto de la suma de la cantidad de instancias efectivamente verdaderas para cada clase de conjunto de datos.

El **F1 score** es una métrica de evaluación utilizada en problemas de clasificación para medir el rendimiento del modelo. Es particularmente útil en casos donde los datos están desbalanceados, es decir, cuando una clase es mucho más frecuente que otra y proporciona un balance entre precisión y recall.

El **Support** es utilizado para evaluar la importancia de cada clase al calcular otras métricas.

Apartado 2:

La validación cruzada, o cross-validation, es una técnica estadística utilizada para evaluar el rendimiento de un modelo de aprendizaje automático y asegurar que los resultados obtenidos son generalizables a datos no vistos. Se utiliza para evitar el sobreajuste (overfitting) y proporcionar una mejor estimación del desempeño del modelo. La validación cruzada es especialmente útil cuando se dispone de un conjunto de datos limitado, similar al presente en el caso de estudio.

Funcionamiento de la Validación Cruzada

El procedimiento más común de validación cruzada es la **k-fold cross-validation**, y a continuación se detalle el paso a paso del funcionamiento del método:

1. **División de los Datos**:
    - El conjunto de datos se divide en k subconjuntos (o "folds") aproximadamente del mismo tamaño.
    - Por lo general, k se elige entre 5 y 10, aunque pueden usarse otros valores.
2. **Entrenamiento y Evaluación**:
    - El modelo se entrena k veces, cada vez con un conjunto de entrenamiento diferente y un conjunto de prueba diferente.
    - En cada iteración, se utiliza k−1 folds para entrenar el modelo y 1 fold para evaluarlo.
    - Este proceso se repite k veces, cambiando el fold de prueba cada vez, hasta que cada fold haya sido usado como conjunto de prueba exactamente una vez.
3. **Cálculo de Métricas**:
    - Se calculan las métricas de rendimiento (como precisión, recall, F1-score, etc.) en cada iteración.
    - Al final, se promedian las métricas obtenidas en las k iteraciones para obtener una estimación más robusta del rendimiento del modelo.

Variantes de Cross-Validation

Como se mencionó en el apartado de funcionamiento, k-fold cross-validation es el procedimiento más común utilizado de validación cruzada. Sin embargo, a continuación, se detallan otras opciones existentes:

- **Leave-One-Out Cross-Validation (LOOCV)**: Cada observación se utiliza como conjunto de prueba una vez, mientras que el resto se utiliza como conjunto de entrenamiento. Esto es equivalente a k-fold cross-validation con k igual al número de observaciones.
- **Stratified k-Fold Cross-Validation**: Similar a k-fold, pero se asegura de que cada fold tenga la misma proporción de clases que el conjunto de datos original. Es útil para problemas de clasificación con clases desbalanceadas.
- **Time Series Cross-Validation**: Se utiliza para datos secuenciales o series temporales, donde el orden de los datos importa. Aquí, los folds se crean respetando el orden temporal.

Beneficios de la Validación Cruzada

1. **Mejor Estimación del Rendimiento del Modelo**: Proporciona una estimación más fiable del desempeño del modelo en comparación con la simple división entrenamiento/prueba.
2. **Reducción del Sobreajuste**: Ayuda a verificar que el modelo generaliza bien a datos no vistos, reduciendo el riesgo de sobreajuste.
3. **Optimización de Hiperparámetros**: Facilita la selección y ajuste de hiperparámetros del modelo de manera más robusta.

En resúmen, la validación cruzada es una técnica esencial en el flujo de trabajo de aprendizaje automático para asegurar que los modelos son robustos y generalizan bien a nuevos datos.

A continuación se detalla la interpretación previa e inicial del código proporcionado por la letra del caso en estudio y se continúa completando el mismo para la búsqueda de los hiper-parámetros correspondientes.

Interpretación inicial del código proporcionado:

Objetivo del Código:

El objetivo es evaluar cómo diferentes configuraciones de preprocesamiento de texto afectan el rendimiento de un modelo de Naive Bayes, utilizando validación cruzada para obtener una evaluación más robusta del rendimiento del modelo.

Visión general del proceso a ejecutar en el código:

1. Se definen varias combinaciones de parámetros que incluyen el uso de stop words, el rango de n-gramas y si se aplica o no el término IDF.

2. Para realizar la validación cruzada, se usa StratifiedKFold con 4 folds, asegurando que cada fold tenga la misma proporción de clases.
3. Los datos de entrenamiento se preparan para ser utilizados en la validación cruzada a través de las combinaciones de parámetros definidas en el paso 1.
4. Para cada combinación de parámetros se generan las iteraciones:
    - Se configuran los transformadores de texto (CountVectorizer y TfidfTransformer) según los parámetros actuales (“stop_words”, “ngram” e “idf” respectivamente).
    - Se divide el dataset de desarrollo en conjuntos de entrenamiento y validación para cada fold.
    - Los datos de entrenamiento se transforman utilizando los transformadores configurados.
    - Se entrena un modelo de Naive Bayes con los datos transformados.
    - Los datos de validación se transforman y el modelo hace predicciones sobre ellos.
    - Se evalúa la precisión de las predicciones y se imprime junto con los parámetros utilizados.

A continuación se muestran las partes del código que fueron completadas:

Se agregaron las variantes de parámetros

```
# TODO: Agregar más variantes de parámetros que les parezcan relevantes
param_sets = [{"stop_words": None, "ngram": (1,2), "idf": True},
              {"stop_words": None, "ngram": (1,2), "idf": False},
             {"stop_words": None, "ngram": (1,1), "idf": False},
             {"stop_words": None, "ngram": (1,1), "idf": True},
             {"stop_words": 'english', "ngram": (1,2), "idf": True},
             {"stop_words": 'english', "ngram": (1,1), "idf": False},
             {"stop_words": 'english', "ngram": (1,1), "idf": True},
             {"stop_words": 'english', "ngram": (1,2), "idf": False}]
```

Se generaron repositorios de resultados para cada iteración con las variantes de parámetros definidas:

```
# Almacenar los resultados
results = []

for params in param_sets:

    accuracies = []
    precisions = []
    recalls = []
    f1_scores = []
```

Se entrena, transforma, predice y evalúa la transformación:

```
# Entrenamos con Train
bayes_clf = MultinomialNB().fit(X_train_tf, Y_train_)

# Transformamos Validation
X_val_counts = count_vect.transform(X_val)
X_val_tfidf = tf_idf.transform(X_val_counts)

# Predecimos y evaluamos en Validation
Y_pred_val = bayes_clf.predict(X_val_tfidf)
acc = get_accuracy(Y_val, Y_pred_val)
accuracies.append(acc)
precisions.append(precision_score(Y_val, Y_pred_val, average='weighted'))
recalls.append(recall_score(Y_val, Y_pred_val, average='weighted'))
f1_scores.append(f1_score(Y_val, Y_pred_val, average='weighted'))
```

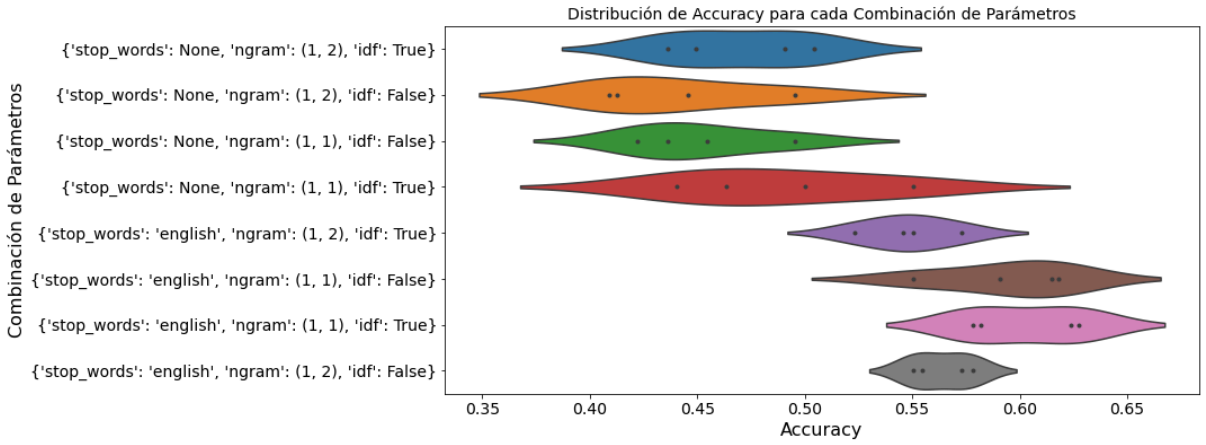
Guardando los resultados de las iteraciones según el repositorio inicial mencionado y lo siguiente detallado:

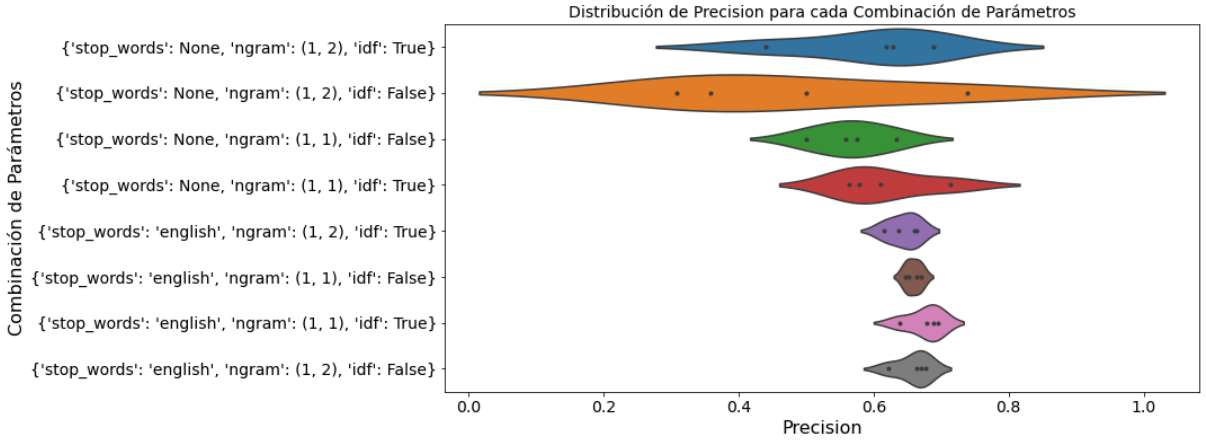
```
# Guardar resultados
results.append({
    'params': params,
    'accuracies': accuracies,
    'precisions': precisions,
    'recalls': recalls,
    'f1_scores': f1_scores,
    'mean_accuracy': np.mean(accuracies),
    'std_accuracy': np.std(accuracies),
    'mean_precision': np.mean(precisions),
    'std_precision': np.std(precisions),
    'mean_recall': np.mean(recalls),
    'std_recall': np.std(recalls),
    'mean_f1_score': np.mean(f1_scores),
    'std_f1_score': np.std(f1_scores)})
```

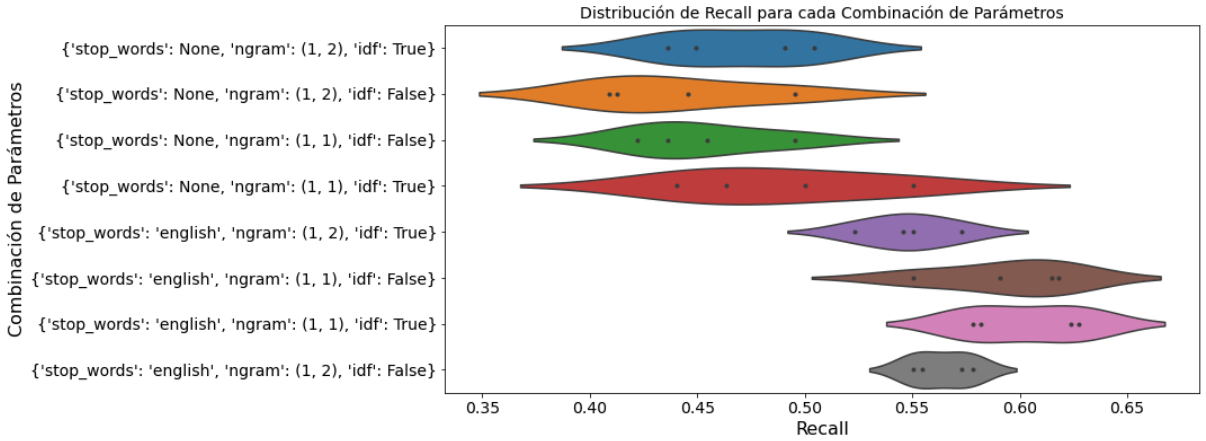
Y finalmente generar las visualizaciones correspondientes, seleccionando la recomendad en la propuesta, como gráficos de violín:

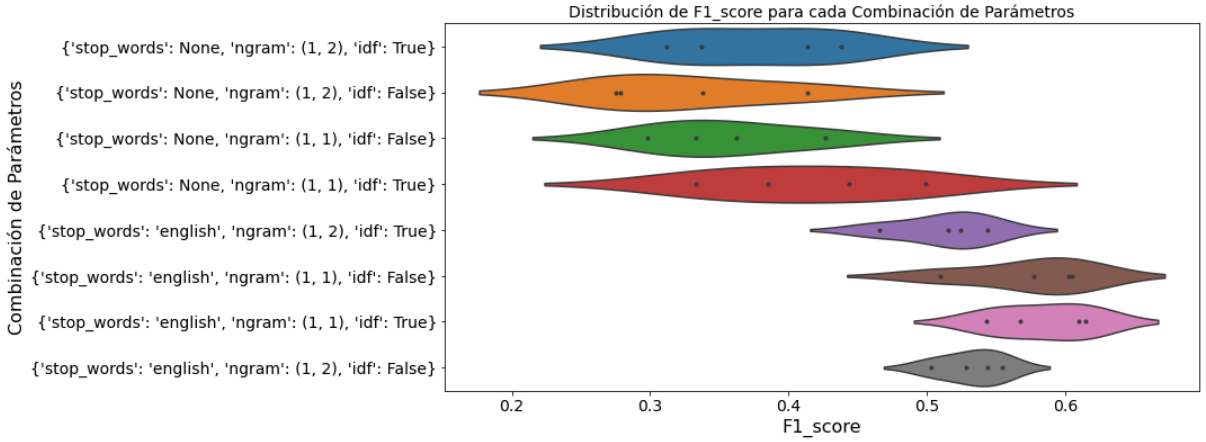
```
# Crear los gráficos de violín
metrics = ['accuracy', 'precision', 'recall', 'f1_score']
for metric in metrics:
    plt.figure(figsize=(12, 6))
    sns.violinplot(x=metric, y='params', data=df, inner='point',density_norm='width')
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.title(f'Distribución de {metric.capitalize()} para cada Combinación de Parámetros', fontsize=14)
    plt.xlabel(metric.capitalize(), fontsize=16)
    plt.ylabel('Combinación de Parámetros', fontsize=16)
    plt.show()
```

A continuación se disponen las gráficas resultantes para cada una de las métricas en análisis y los resultados para cada una de las combinaciones de parámetros:









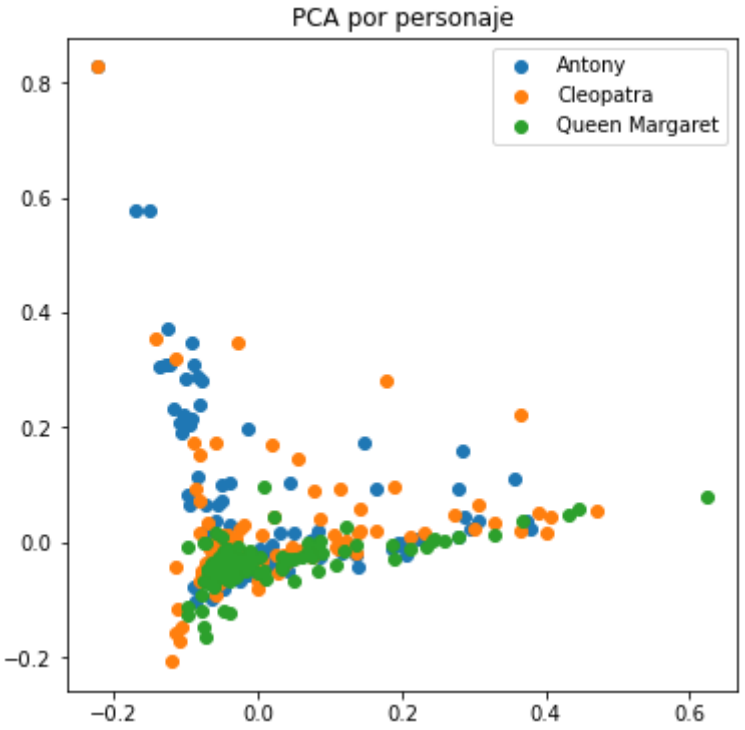
En función de los resultados de las visualizaciones, la mejor combinación de parámetros sería la siguiente (color rosado):

- **stop_words = “english”**
- **n_gram = (1, 1)**
- **idf = True**

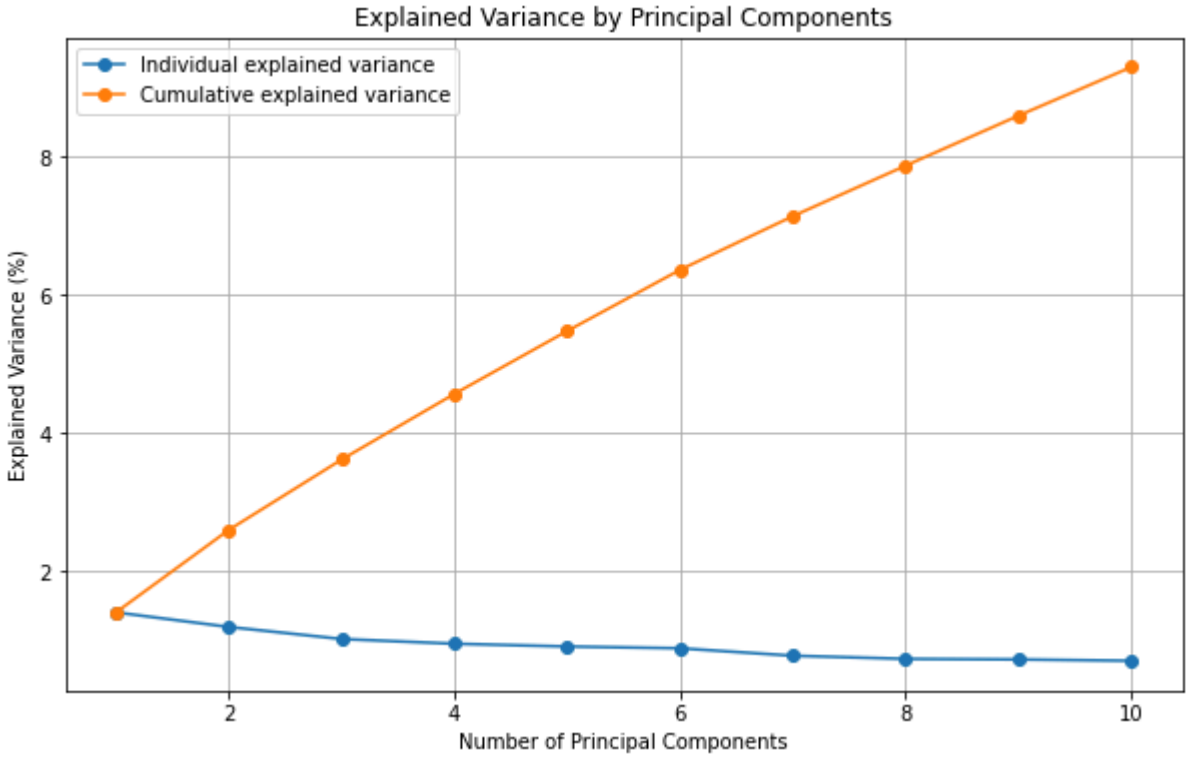
Apartado 3:

Volviendo a entrenar sobre todo el conjunto de entrenamiento con la combinación de parámetros seleccionada (sin quitar datos para validación):

A continuación se detallan las métricas resultantes y la matriz de confusión, así como adicionalmente también un mapeo previo con PCA, más un gráfico visualizando la varianza explicada.



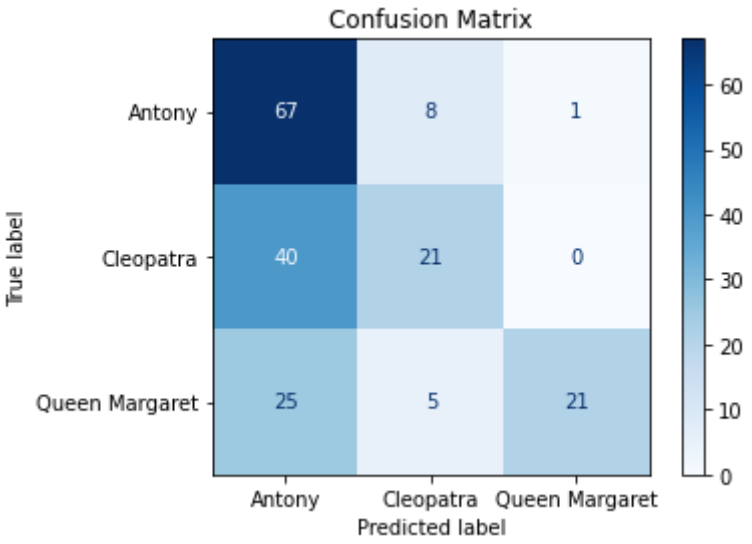
La varianza explicada a medida que se agregan componentes principales:



El valor de accuracy del modelo es de 0,58. El mismo surge del siguiente cociente (ver Apartado 1 - Parte 2 para mayor referencia del cálculo) con los componentes tomados desde el resultado de la matriz de confusión:

$$Accuracy = \frac{67+21+21}{67+21+21+8+1+40+25+5} = \frac{109}{188} = 0.58$$

Reporte de la matriz de confusión:



Reporte del valor final de las métricas:

Character	Precision	Recall	F1 Score	Support
Antony	0,51	0,88	0,64	76
Cleopatra	0,62	0,34	0,44	61
Queen Margaret	0,95	0,41	0,58	51

En vistas de discutir sobre las limitaciones de un modelo basado en BoW o TF-IDF, de acuerdo a los resultados obtenidos, se podrían resaltar las siguientes dos grandes limitantes:

Clasificación de Personajes:

- Al clasificar personajes, se pueden perder matices importantes en la caracterización de los mismos debido a la pérdida de contexto. Frases que describen a un personaje de manera similar pueden ser tratadas de forma diferente si no se captura el contexto adecuadamente.

Varianza en los Datos:

- La variabilidad en las longitudes de los diálogos de los personajes puede afectar la representación en BoW y TF-IDF, sesgando el modelo hacia personajes que hablan más o menos, independientemente de la relevancia de sus palabras.

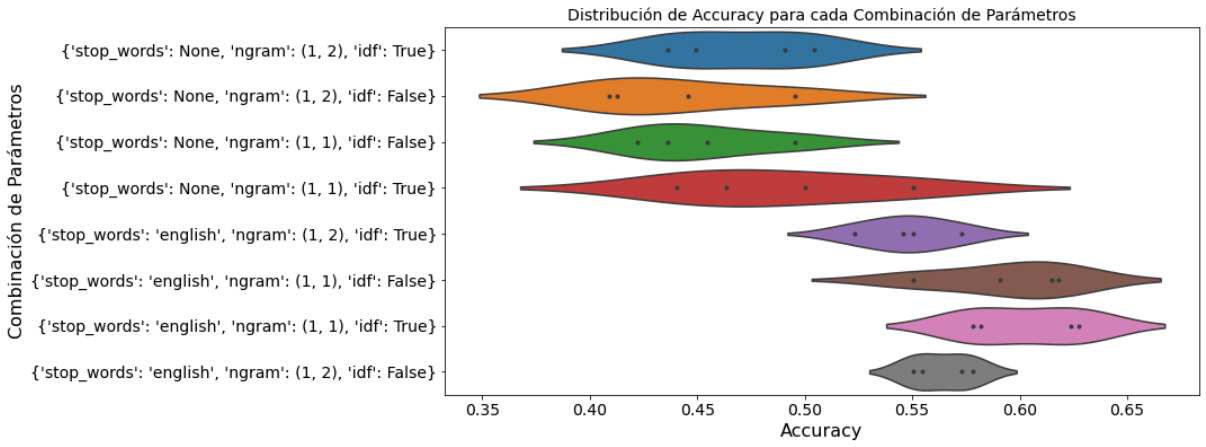
Apartado 4:

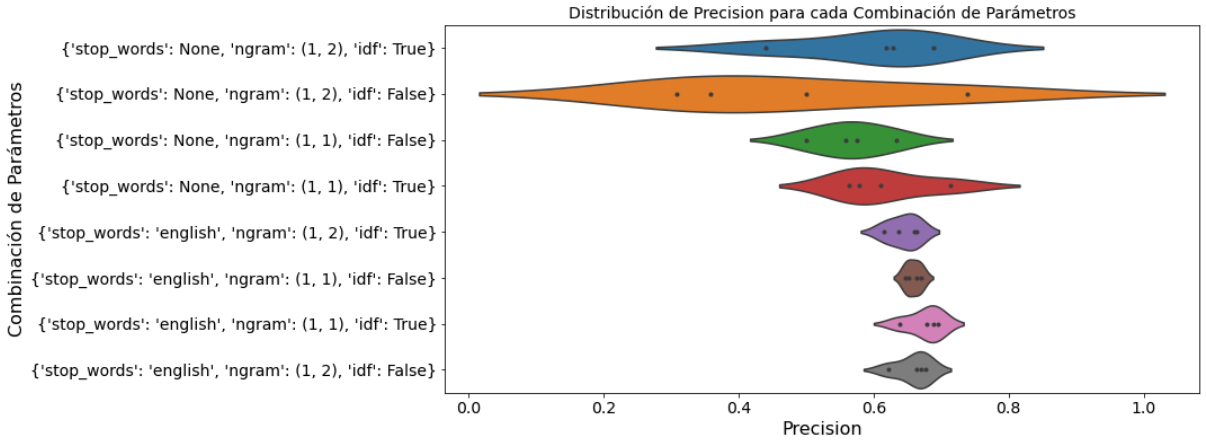
Para evaluar un modelo adicional, vamos a usar un Support Vector Machine (SVM), específicamente con el clasificador LinearSVC de scikit-learn.

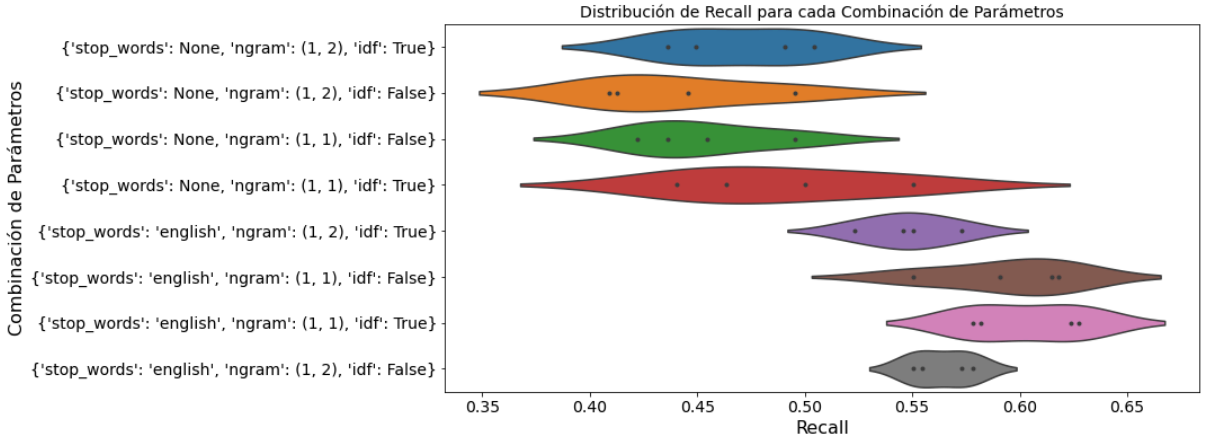
¿Cómo funciona un SVM?

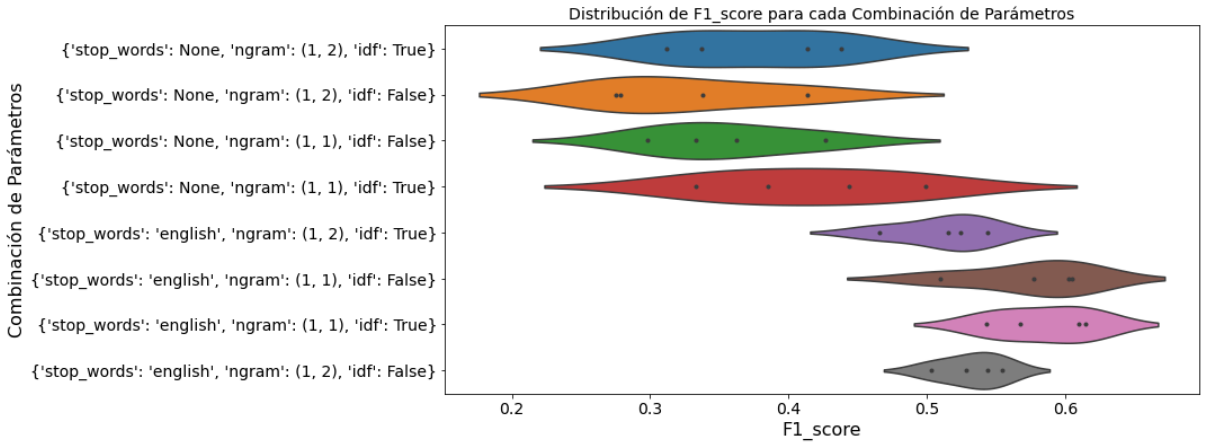
El **Support Vector Machine (SVM)** es un algoritmo de clasificación que encuentra el hiperplano óptimo que separa los datos en diferentes clases. En el caso de LinearSVC, se utiliza un hiperplano lineal para realizar la clasificación. Los SVMs son efectivos en espacios de alta dimensión y en casos donde el número de dimensiones es mayor que el número de muestras.

Realizamos cross-validation con el nuevo modelo SVM, presentándose los gráficos de violín para cada una de las métricas:









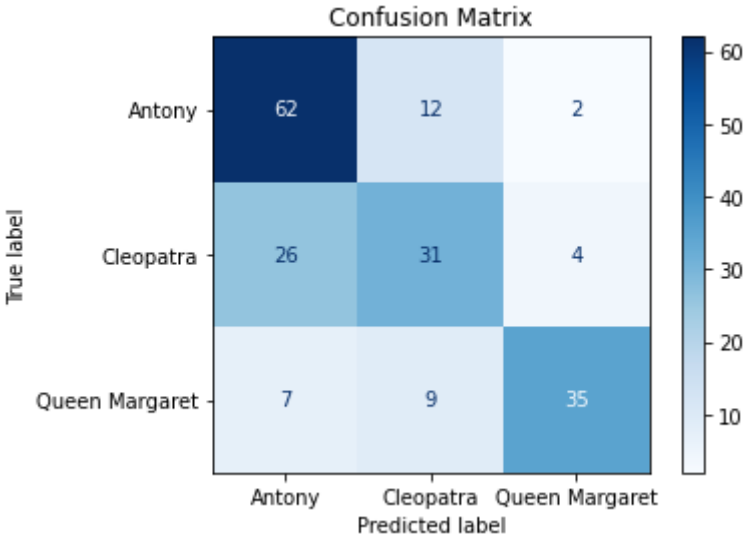
En función de los resultados de las visualizaciones, la mejor combinación de parámetros sería la siguiente (color rosado):

- **stop_words = “english”**
- **n_gram = (1, 1)**
- **idf = True**

Bajo estos parámetros seleccionados, el valor de accuracy resultante para el modelo, teniendo en cuenta los resultados expuestos de la matriz de confusión

presentada debajo es 0,68 (ver nuevamente Apartado 1 - Parte 2 para mayor detalle del cálculo).

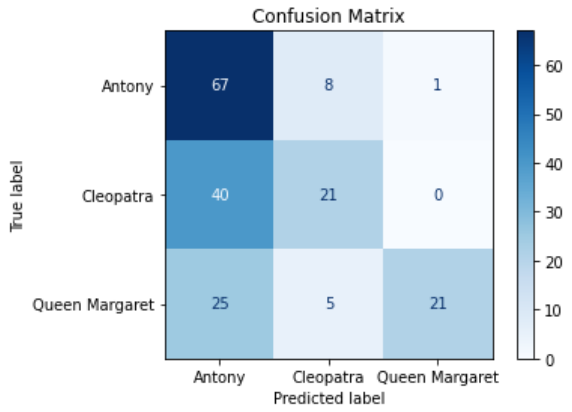
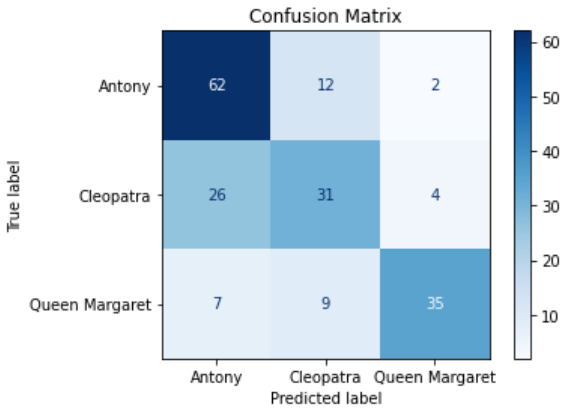
Reporte de la Matriz de confusión aplicando el modelo SVM:



Métricas resultantes aplicando el modelo SVM a los datos:

Character	Precision	Recall	F1 Score	Support
Antony	0,65	0,82	0,73	76
Cleopatra	0,60	0,51	0,55	61
Queen Margaret	0,85	0,69	0,76	51

Para realizar la comparativa se presentan los resultados relevantes de ambos modelos en las siguientes tablas resumen:

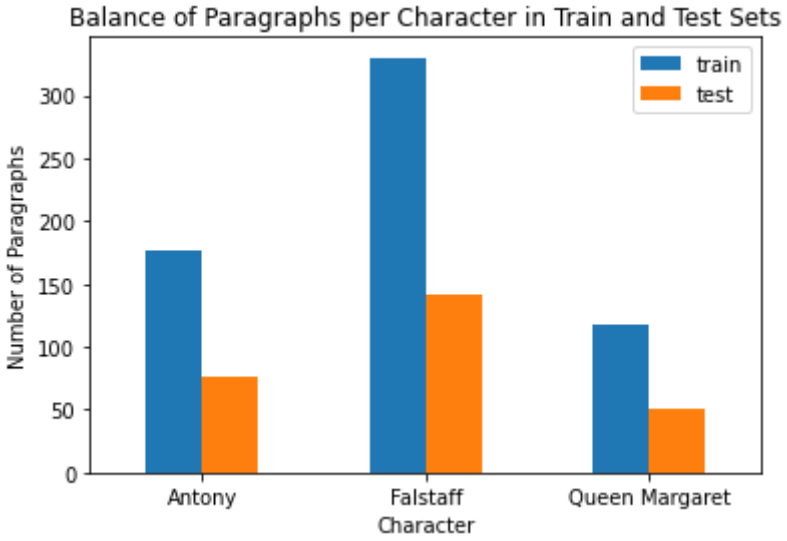
Resultado de Accuracy y Matriz de Confusión	
Modelo Multinomial Naive Bayes (MNB)	**Modelo SVM**
Accuracy = 0.58	Accuracy = 0.68
	

Character	Precision		Recall		F1 Score		Support	
	MNB	SVM	MNB	SVM	MNB	SVM	MNB	SVM
Antony	0,51	0,65	0,88	0,82	0,64	0,73	76	76
Cleopatra	0,62	0,60	0,34	0,51	0,44	0,55	61	61
Queen Margaret	0,95	0,85	0,41	0,69	0,58	0,76	51	51

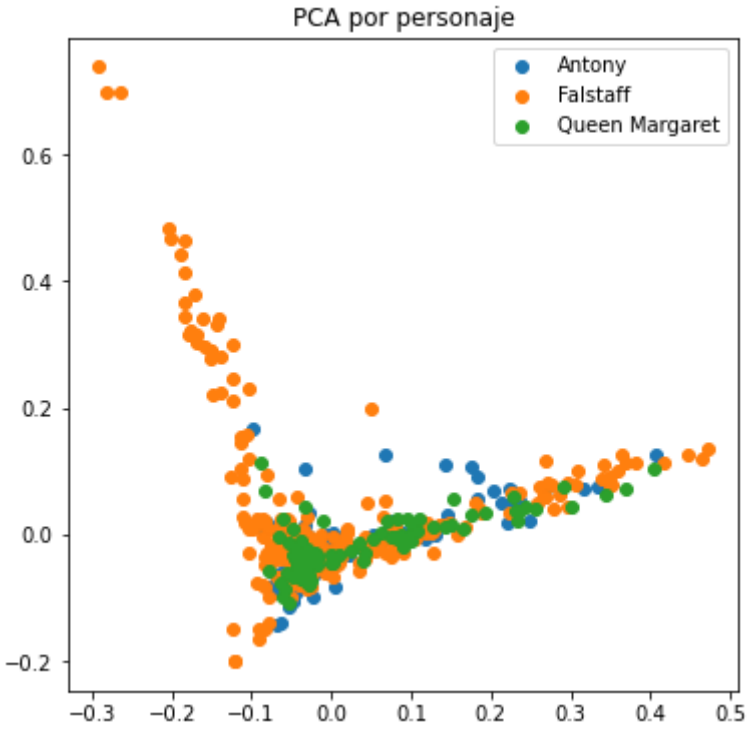
Teniendo en cuenta que según los resultados, se mejora la precisión general del modelo con SVM, así como también se ve que se mejora el F1 Score para todos los personajes, es posible aventurar que los resultados, para este set de datos y el modelo del caso de estudio, SVM se adecúa mejor y genera mejores predicciones, tal cual era esperable dadas la capacidad para manejar alta dimensionalidad y sparsidad, características comunes en los datos de texto.

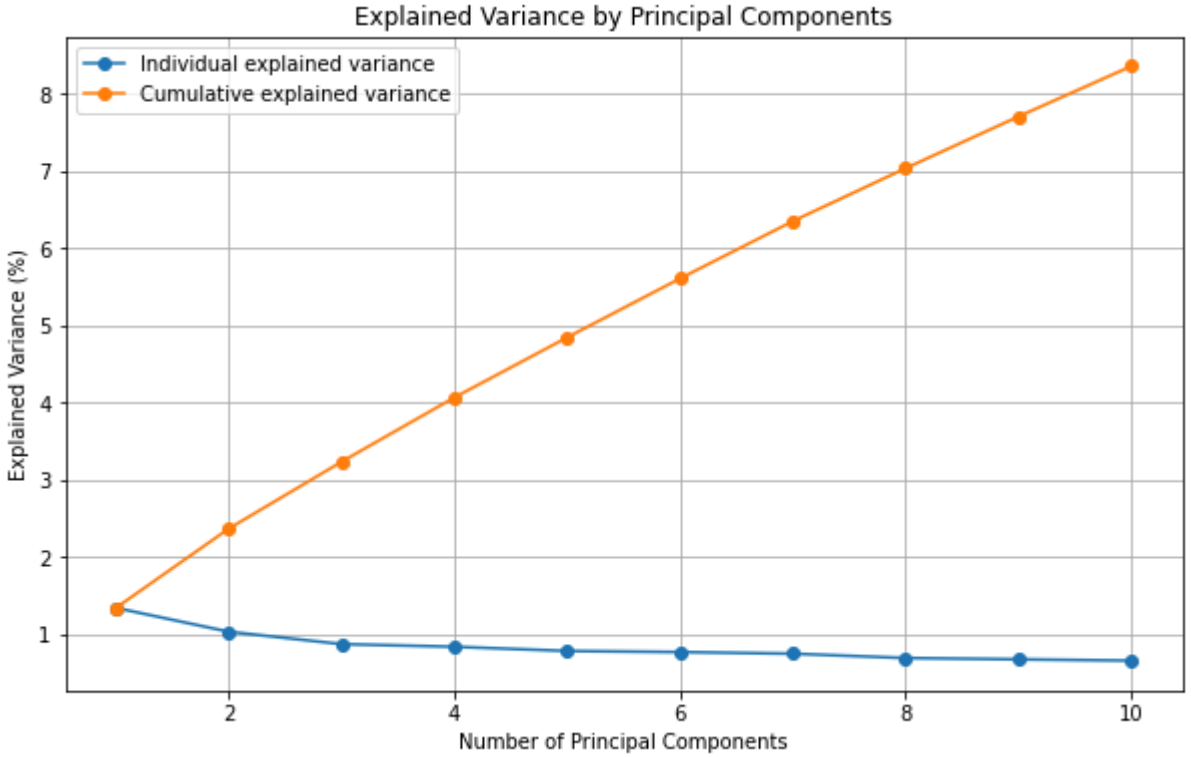
Apartado 5:

Para evaluar el problema cambiando un personaje por otro, se decide cambiar a Cleopatra por Falstaff. Manteniendo la estrategia general del caso en estudio, se utiliza un Dataset de 3 personajes con un conjunto de test del 30% utilizando muestreo estratificado, dando como resultado la visualización a continuación:

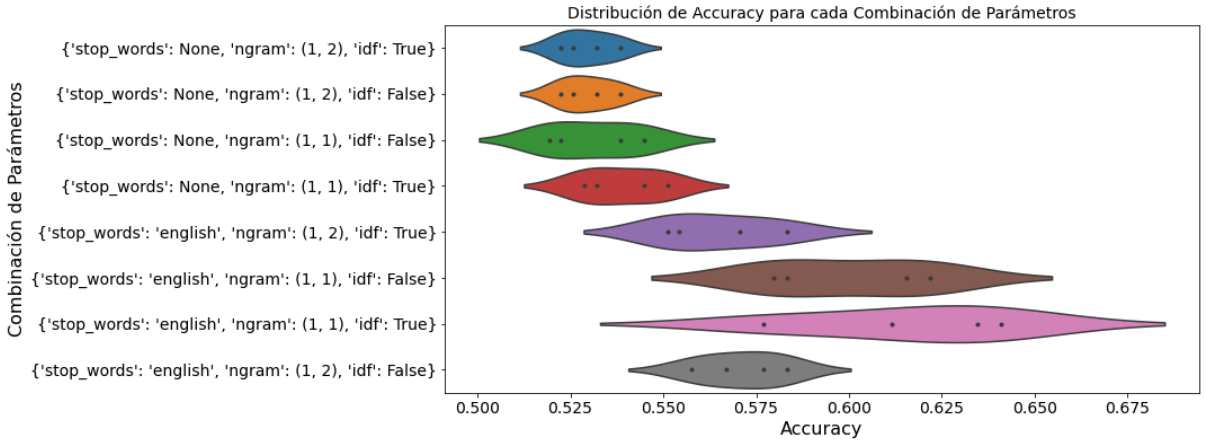


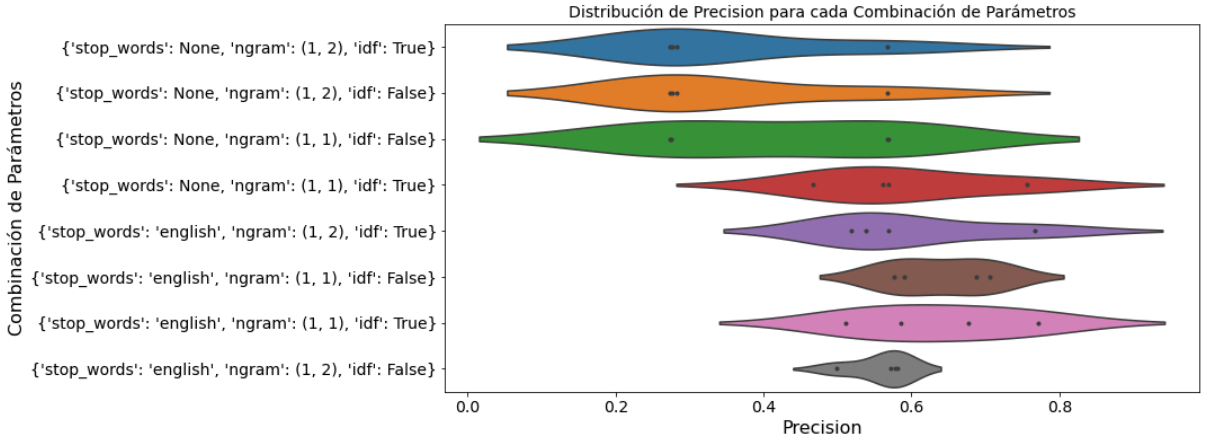
En busca de realizar un mapeo del set de datos, se muestra el siguiente PCA utilizando dos componentes principales, con su varianza explicada (hasta 10 componentes principales), manteniéndose hasta ahora las mismas conclusiones obtenidas que para los apartados previos.

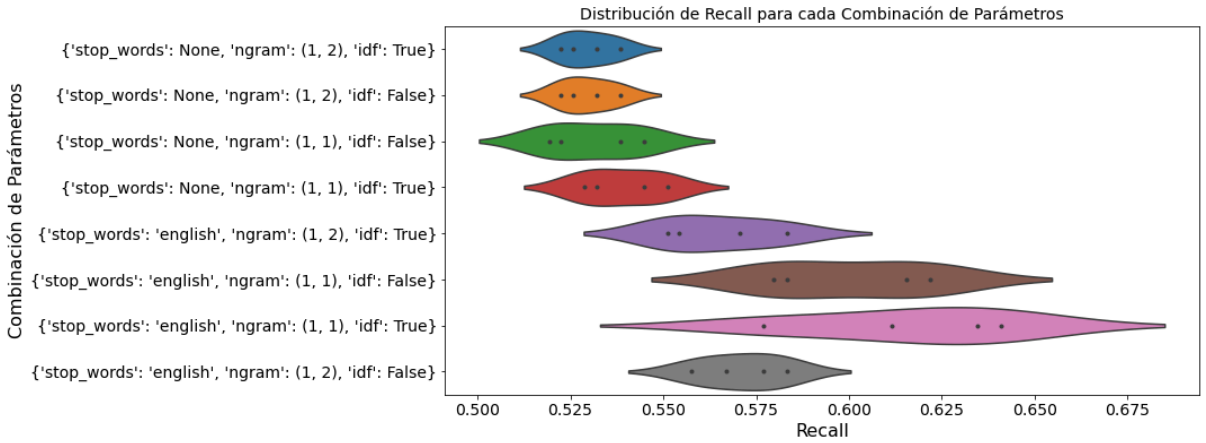




En busca de identificar el mejor set de parámetros, se utiliza la técnica de Cross-Validation nuevamente obteniéndose como resultados los gráficos de violín a continuación:



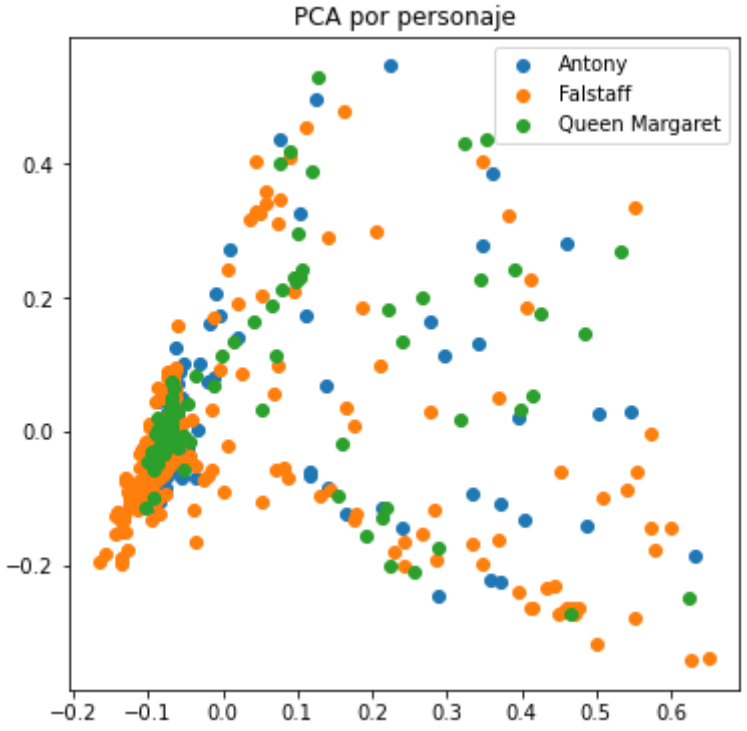




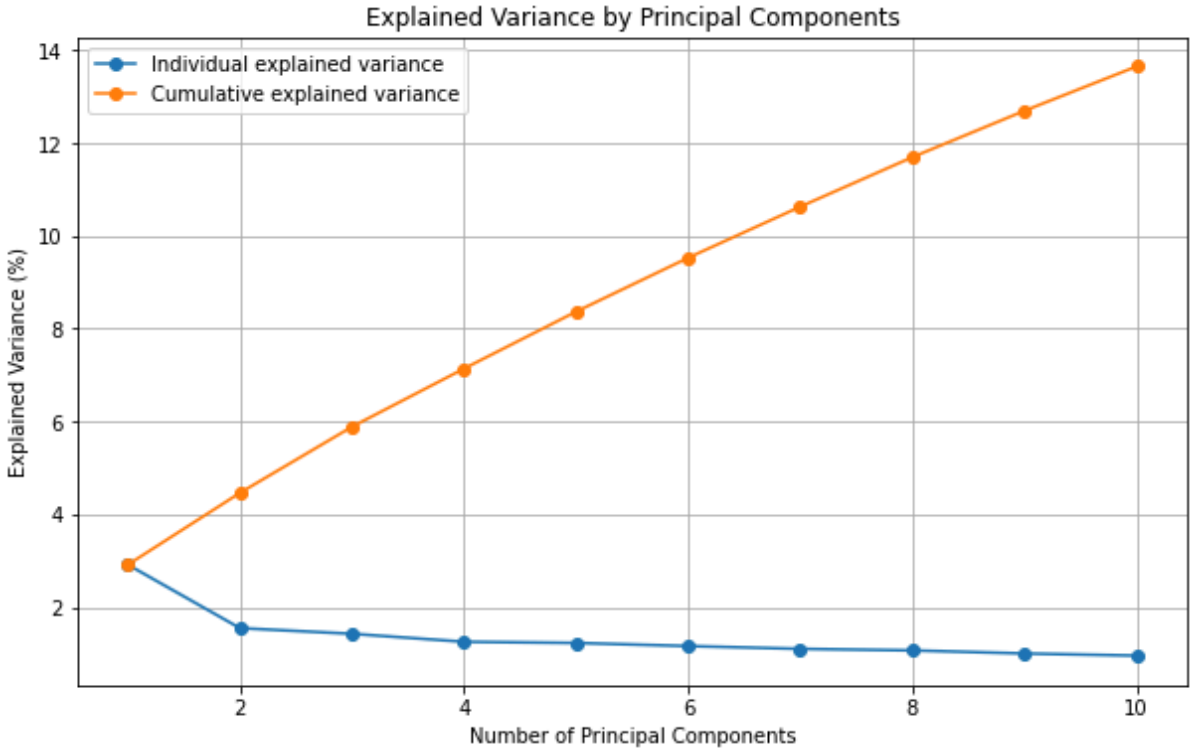
En esta ocasión y para este conjunto de datos, la mejor combinación de parámetros, y, por lo tanto, la seleccionada sería la siguiente (color marrón):

- **stop_words = “english”**
- **n_gram = (1, 1)**
- **idf = False**

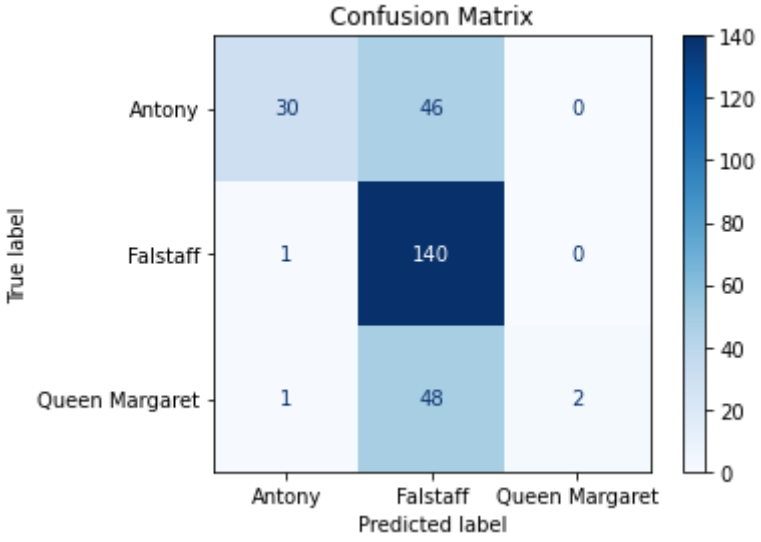
Se visualiza a continuación el gráfico de PCA resultante de esta selección de parámetros, así como también el valor de varianza explicada, valor de accuracy del modelo, su correspondiente matriz de confusión y tabla de métricas resultantes por personaje:



Varianza explicada:



El valor de accuracy es 0,64 y se presenta la matriz de confusión correspondiente:



Las métricas resultantes por cada personaje son las siguientes:

Character	Precision	Recall	F1 Score	Support
Antony	0,94	0,39	0,56	76
Falstaff	0,60	0,99	0,75	141
Queen Margaret	1,00	0,04	0,08	51

En este caso, el modelo da señales de terminarse inclinando a que prácticamente no existen párrafos asociados a Queen Margaret, terminando con un Recall para Falstaff cercano a 1 y para Queen Margaret cercano a 0. Los valores resultantes para Queen Margaret son verdaderamente mejorables y se podría aventurar que esto sería fruto del desbalance entre Falstaff y Queen Margaret que es posible visualizar en el gráfico inicial del presente apartado.

Respecto a temas de sobre-muestreo y submuestreo, el sobre-muestreo puede ayudar a crear más ejemplos de la clase minoritaria y el submuestreo puede equilibrar las clases pero a costa de perder datos de las clases mayoritarias. Teniendo en cuenta el desbalance importante que se ve en el conjunto de datos, cualquiera de las dos formas de muestreo podrían aportarles resultados al modelo para mejorar su performance.

Apartado 6:

Técnica Alternativa: Word Embeddings

Una técnica popular para la extracción de características de texto que va más allá de BoW y TF-IDF es el uso de **word embeddings**. Word embeddings son representaciones vectoriales densas de palabras en un espacio continuo, donde palabras con significados similares tienen representaciones similares. A diferencia de los enfoques basados en frecuencia como BoW y TF-IDF, los word embeddings capturan relaciones semánticas y contextuales entre palabras.

Métodos Comunes de Word Embeddings:

1. **Word2Vec:**
    - Desarrollado por Google, Word2Vec es una técnica que genera embeddings utilizando una red neuronal de dos capas. Hay dos enfoques principales: Continuous Bag of Words (CBOW) y Skip-gram.
    - **CBOW:** Predice la palabra objetivo utilizando el contexto circundante.
    - **Skip-gram:** Predice las palabras de contexto utilizando la palabra objetivo.
    - Word2Vec mapea cada palabra a un vector de tamaño fijo (por ejemplo, 100, 200, 300 dimensiones), y palabras similares en significado tienen vectores cercanos en el espacio vectorial.
2. **GloVe (Global Vectors for Word Representation):**
    - Desarrollado por Stanford, GloVe es un método que combina el conteo global de palabras con las capacidades de aprendizaje de embeddings de Word2Vec.
    - GloVe construye una matriz de coocurrencia de palabras y factoriza esta matriz para obtener los vectores de palabras.
3. **FastText:**
    - Desarrollado por Facebook, FastText es una extensión de Word2Vec que toma en cuenta la morfología de las palabras. Cada palabra es representada como una suma de los embeddings de sus n-gramas de caracteres.
    - Esto permite a FastText manejar palabras fuera del vocabulario y aprender representaciones más finas, especialmente para idiomas morfológicamente ricos.

Diferencias Esperadas en los Resultados:

1. **Captura de Relaciones Semánticas:**
    - Word embeddings captura relaciones semánticas entre palabras, lo que significa que las palabras con significados similares estarán cercanas en el espacio vectorial. Esto mejora la capacidad del modelo para generalizar a textos no vistos y capturar sutilezas del lenguaje.
2. **Contexto:**
    - Mientras que BoW y TF-IDF tratan las palabras de forma independiente, word embeddings puede capturar el contexto en el que las palabras aparecen. Por ejemplo, las palabras "banco" en "sentarse en el banco" e "ir al banco" tendrán diferentes representaciones en embeddings contextuales como los generados por técnicas más avanzadas (e.g., BERT).
3. **Dimensionalidad:**
    - Los vectores de word embeddings suelen ser de menor dimensionalidad (e.g., 100-300 dimensiones) comparados con las

matrices dispersas de BoW o TF-IDF que pueden tener miles de dimensiones. Esto puede resultar en modelos más eficientes computacionalmente.

4. **Rendimiento en Clasificación:**
    - Se espera que los modelos entrenados con word embeddings tengan mejor rendimiento en tareas de clasificación de texto, especialmente cuando se trata de comprender el significado y las relaciones entre palabras. Esto puede llevar a mejoras en métricas como precisión, recall y F1-score.
5. **Manejo de Palabras Nuevas:**
    - Técnicas como FastText pueden manejar palabras fuera del vocabulario mejor que BoW o TF-IDF, ya que representan palabras como composiciones de n-gramas de caracteres, permitiendo que las palabras nuevas se aproximen utilizando componentes conocidos.

El uso de **word embeddings** para la extracción de características de texto representa una mejora significativa sobre los métodos tradicionales como BoW y TF-IDF. Capturan más información contextual y semántica, lo que generalmente lleva a un mejor rendimiento en tareas de procesamiento de lenguaje natural y clasificación de texto. Sin embargo, estos métodos también requieren más recursos computacionales para entrenar y pueden ser más complejos de implementar.

Esta técnica sería un buen complemento o incluso una alternativa a los enfoques actuales del proyecto, potencialmente llevando a mejores resultados y modelos más robustos.