Universidad de la República

Facultad de Ingeniería

Maestría en Ciencia de Datos y Aprendizaje Automático

Introducción a la Ciencia de Datos 2023

Tarea 2. Análisis de Datos de Texto y Modelos de Clasificación en la Obra de William Shakespeare.

Grupo 1:
Ignacio Corrales

Julio de 2023

Contenido

1. Introducción

Este informe tiene por objeto la presentación de los datos relevantes obtenidos a través de técnicas de análisis de datos y clasificación aplicadas sobre una base de datos que contiene, de manera estructurada, la obra del renombrado autor inglés William Shakespeare.

El trabajo realizado, se enmarca en el curso de Introducción a Ciencia de Datos, dictado por la Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay, y corresponde a la segunda tarea obligatoria de la versión 2023 de su dictado.

A su vez, este trabajo, se apoya en lo realizado en el marco de la tarea 1, propuesta en el curso, al cual se puede acceder mediante este link.

De forma sucinta, se puede decir que el trabajo consistió en representar párrafos de texto, contenidos en la obra, en un espacio de vectores normalizados. Mediante los mismos, se entrenaron algunos modelos de clasificación supervisados y se analizaron métricas de predicción para un conjunto de párrafos objetivo.

2. Entorno de trabajo

Para la realización del análisis se desarrollo un Python Notebook en Google Collab y se utilizaron las librerías jupyter, pandas, sqlalchemy, pymysql, seaborn, pillow, scikit-learn, y nltk.

El todo el código Python y sus salidas, pueden verse accediendo al notebook, alojado en esta ubicación.

3. Ejercicio 1

3.1. Ejercicio 1.1

Como punto de partida del análisis, se replicó lo hecho en la referida Tarea 1, y se copió el código de la función que "limpia" el texto de los párrafos. La misma elimina los signos de puntuación y también todo el texto que se encuentre dentro de paréntesis rectos, ya que dicho texto refiere a direcciones de puesta en escena y no a palabras efectivamente pronunciadas por los distintos personajes de la obra.

```python
# Limpieza del texto
def clean_text(df, column_name):
    # Elimino las indicaciones de dirección entre paréntesis rectos
    result = df[column_name].str.replace('\[.*\]', '', regex=True)
    # Quitar signos de puntuación y cambiarlos por espacios (" ")
    for punc in ["[","]","&", "\n", ",", ".", "?", "!", ":", ";", "-",
                 "\'", "\"", "(", ")"]:
        result = result.str.replace(punc, " ")
    return result
```

Código 1: función de limpieza de texto

Luego de depurado el texto, se filtraron los párrafos de manera de almacenar únicamente aquellos pronunciados por los siguientes personajes:

- Antony
- Cleopatra
- Queen Margaret

Este conjunto de párrafos que pasaron el filtro mencionado, se convirtió en nuestro cuerpo de análisis y generamos dos listas a partir del mismo:

- **X**: la lista de todos los párrafos de nuestro cuerpo.
- **y**: el nombre del personaje al que pertenece cada párrafo de X.

Deseamos analizar la factibilidad de poder entrenar un modelo para predecir, a partir de un párrafo cualquiera, a qué personaje pertenece.

Con dicho objetivo en mente, se procedió a generar conjuntos de entrenamiento y de test. Los conjuntos generados fueron los siguientes:

- **X_train:** un conjunto de párrafos para entrenar modelos, que contiene el 70% de los párrafos del cuerpo.
- **y_train:** los nombres de los personajes a los que pertenecen los párrafos contenidos en X_train.
- **X_test:** el 30% restante de los párrafos del cuerpo, reservados para probar los modelos entrenados.
- **y_test:** Los nombres de los personajes que pronunciaron los párrafos de X_test, resrevados para comparar contra la salida predicha por los modelos.

```
# Partir train/test 30% estratificados
# -> Definir X_train, X_test, y_train, y_test
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size = 0.3,
                                                      stratify=y,
                                                      random_state=42)
print(f"Tamaños de Train/Test: {len(X_train)}/{len(X_test)}")
```

```
Tamaños de Train/Test: 438/188
```

Código 2: Definición de conjuntos de entrenamiento y test.

Como se ve, se utilizaron los parámetros `test_size=0.3`, para indicar que el conjunto de test tendrá el 30% de los sujetos del cuerpo, `stratify=y` para indicar que la proporción de las etiquetas de cada conjunto sean igual a las proporciones que existen en todo el cuerpo y `random_state=42` para obtener una separación aleatoria de los conjuntos, pero repetible en el futuro.

3.2. Ejercicio 1.2

Si graficamos las proporciones de las etiquetas (personajes) en cada uno de los conjuntos (train y test), vemos que la estratificación realizada fue correcta:

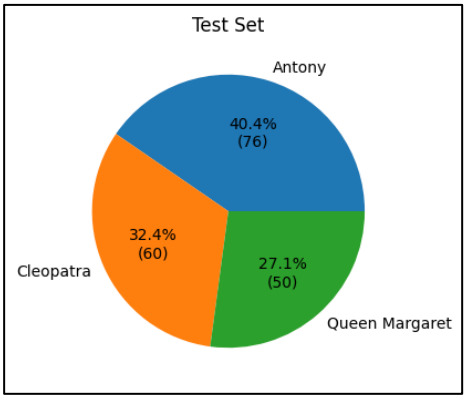


Ilustración 2: Proporción en test.

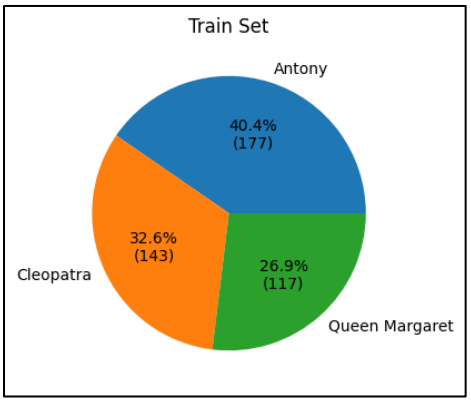


Ilustración 1: Proporción en train.

3.3. Ejercicio 1.3

El siguiente paso consistió en transformar los párrafos de entrenamiento (X_train) en vectores numérico de conteo de palabras. La técnica consiste en generar un espacio de tantas dimensiones como palabras distintas haya en el cuerpo de texto. Luego, a cada párrafo se lo representa con un vector donde, para cada componente (que representa a una palabra existente del cuerpo) se le asigna un valor igual a la cantidad de veces que aparece dicha palabra en el párrafo.

De esta manera, cada párrafo será un vector de una dimensión muy alta (tantas como palabras distintas existan en el cuerpo), y que en la mayoría de sus componentes tendrá el valor 0, ya que es esperable que en la obra de Shakespeare existan muchas palabras diferentes y que cada párrafo incluya una pequeñísima porción de las mismas.

En nuestro caso concreto, la salida de este procesamiento, fue una matriz de dimensión 438 x 2.784.

```
count_vect = CountVectorizer(stop_words=None, ngram_range=(1,1))
X_train_counts = count_vect.fit_transform(X_train)
X_train_counts

<438x2782 sparse matrix of type '<class 'numpy.int64'>'
        with 10713 stored elements in Compressed Sparse Row format>
```

Código 3: Bag of Words

El resultado obtenido nos indica que tenemos 438 párrafos en nuestro conjunto de entrenamiento (resultado verificable en la “Ilustración 2” y el “Código 2”) y que el total de palabras distintas existentes la suma de todos esos párrafos, es de 2.784.

Luego, cada párrafo se representa como un vector de dimensión 2.784 donde para cada valor

tendrá un número indicando cuantas veces aparece esa palabra en el párrafo. Una matriz con esas dimensiones (438 x 2.782) podría llegar a tener hasta 1.218.516 entradas distintas (en el caso de cada párrafo contuviera al menos una vez cada palabra).

Sin embargo, es lógico deducir que la mayoría de las entradas serán 0, ya que cada párrafo contendrá una pequeña porción de las palabras. Es así que, en la Ilustración anterior, vemos que la cantidad de elementos distintos a 0, son sólo 10.713 (menos de un 1% de las entradas). Es esta la razón que justifica su almacenamiento como una "sparce matrix" comprimida y así ahorrar espacio en memoria. Esto puede ser crucial a la hora de procesar textos más extensos que esos 400 párrafos, donde la dimensión puede ser aún mayor (más cantidad de palabras diferentes) y una mayor cantidad de vectores.

3.4. Ejercicio 1.4

Antes de continuar con el análisis, expliquemos brevemente que es un n-grama. Un n-grama es un conjunto de n símbolos consecutivos. En el contexto del lenguaje natural, se refiere a una cadena de texto que contiene n símbolos (palabras o signos de puntuación) consecutivos.

Como ejemplo, si usamos bigramas (n = 2), se puede representar el espacio del texto como un vector con tantas dimensiones como combinación de 2 palabras consecutivas existan en el texto. Luego, en vez de contar ocurrencia de palabras (bag of words), se contará la ocurrencia de cada bigrama. Resulta intuitivo imaginar que la representación usando n-gramas, será de mayor dimensión que la de palabras, ya que habrá más combinaciones entre palabras que palabras y a su vez, la probabilidad de que esa combinación aparezca en un párrafo dado será menor que la probabilidad que exista una palabra, por lo cual será una matriz más dispersa.

Volviendo a nuestro análisis, el siguiente paso fue obtener la representación de nuestro conjunto de entrenamiento como vectores numéricos normalizados usando la estrategia TF-IDF.

Para entender estas dos siglas, imaginemos que luego de entrenar nuestro modelo con 438 párrafos de longitud promedio de 200 palabras, queremos clasificar un nuevo párrafo, pero que contiene 1500 palabras. Aun hablando del mismo tema que alguno de los párrafos usados para entrenar, es probable que, al ser más extenso, las palabras más importantes del contenido, tengan ocurrencias más altas que en textos más cortos.

Para salvar este problema, se puede usar TF (term frequency) en vez del conteo absoluto de ocurrencias de un término (palabra o n-grama). La frecuencia sería dividir la cantidad de ocurrencias de ese término, sobre el total de términos que existe en el texto que deseo representar, de esta manera estamos calculando la importancia relativa de ese término en todo el párrafo.

Una segunda mejora a la técnica, es la aplicación de "Inverse Document Frequency" (IDF), que consiste en quitarle peso relativo a aquellos términos cuya frecuencia es alta en todo el cuerpo de texto analizado. De esta manera, se les quita peso a aquellos términos que, en definitiva, brindan menos información y por contraposición, pesarán más aquellos términos que aparecen en porciones más pequeñas de texto.

3.5. Ejercicio 1.5

Para continuar trabajando con nuestros datos, realizamos un Análisis de Componentes Principales (PCA por sus siglas en inglés) y utilizamos los 2 componentes principales. De esta manera, nos quedamos con las 2 dimensiones (de las 2.782 originales) que brindan mayor información.

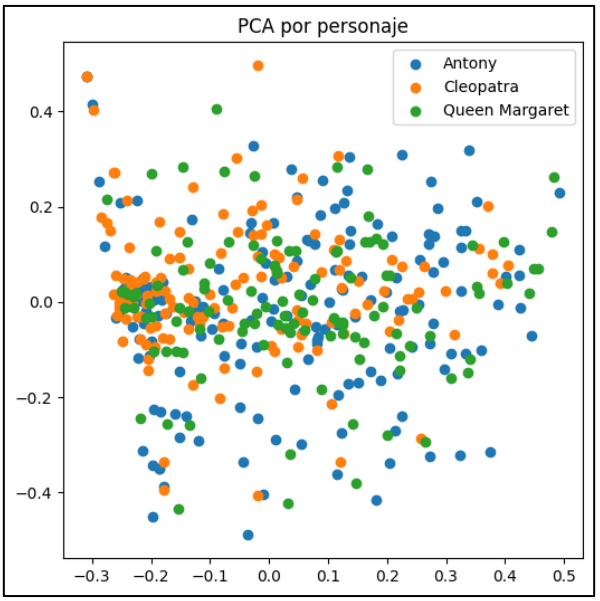


Ilustración 3: Párrafos en el espacio de dimensión 2, TF, palabras

Como vemos, en el espacio de dos dimensiones, resulta difícil detectar límites evidentes entre las clases de los párrafos. Se puede decir que los párrafos de Cleopatra (amarillos) no suelen ocupar la parte inferior del plano con la frecuencia que sí lo hacen los de Antony (azul), pero dicha deducción visual, no parece ser suficiente para obtener una clasificación eficiente.

Para proseguir, se decidió realizar una visualización semejante, pero utilizando la representación de bigramas y adicionar la técnica IDF a la TF ya utilizada en la ilustración anterior. Además, se usó una *stop words list*, de manera que se filtraron las palabras que carecen de significado y que únicamente se usan para dar cohesión y conjunción a las oraciones. Estas palabras son de alta frecuencia en todas las oraciones y por ende brindan poca información.

```
# Bag of n-gramas con n-gramas (1,2) y filtrando stopwords
count_vect2 = CountVectorizer(stop_words='english', ngram_range=(1,2))
X_train_counts2 = count_vect2.fit_transform(X_train)
X_train_counts2


tf_idf = TfidfTransformer(use_idf=True)
X_train_tf_idf = tf_idf.fit_transform(X_train_counts2)
X_train_tf_idf
```

```
<438x8379 sparse matrix of type '<class 'numpy.float64'>'
        with 11920 stored elements in Compressed Sparse Row format>
```

Código 4: Generamos representación TF-IDF de bigramas filtrando stop words

Lo primero que observamos es que la dimensión de nuestra matriz resultante, es de 438 x 8.379.

Si bien la cantidad de vectores permanece constante, lo cual es lógico porque los párrafos de entrenamiento siguen siendo 438, la dimensión de cada uno creció al triple (2.782 dimensiones en

el conteo de palabras). Esto era esperable como detallamos anteriormente ya que las combinaciones de a 2 palabras son más que las palabras existentes en el texto.

A su vez, de las 3.670.002 entradas de esa matriz, sólo 11.920 tienen valores distintos a 0 (0,3%), así que la matriz es más dispersa, tal cual lo vaticinábamos intuitivamente.

Al quedarnos con los dos componentes más importantes, obtenemos la siguiente distribución de los párrafos en el plano:

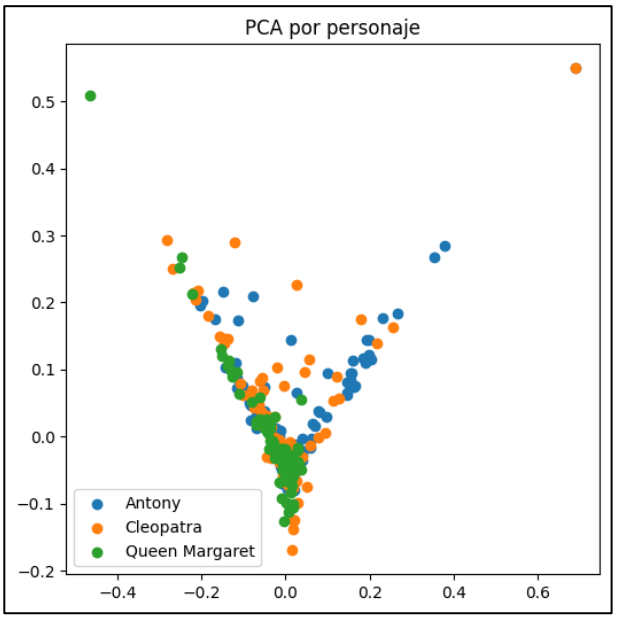


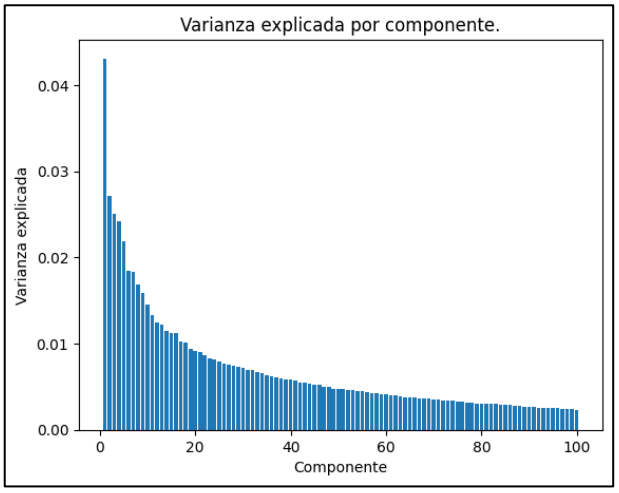
Ilustración 4: Párrafos en el plano, bigramas, TF-IDF

A simple vista, podemos decir que los vectores no están tan dispersos en el plano como en el caso anterior y que hay partes del plano que los párrafos de Queen Margaret no ocupan, y que sí son frecuentemente ocupados por Cleopatra y en mayor medida por Antony. Sin embargo, seguimos percibiendo como dificultosa, la clasificación de clases con sólo 2 componentes.

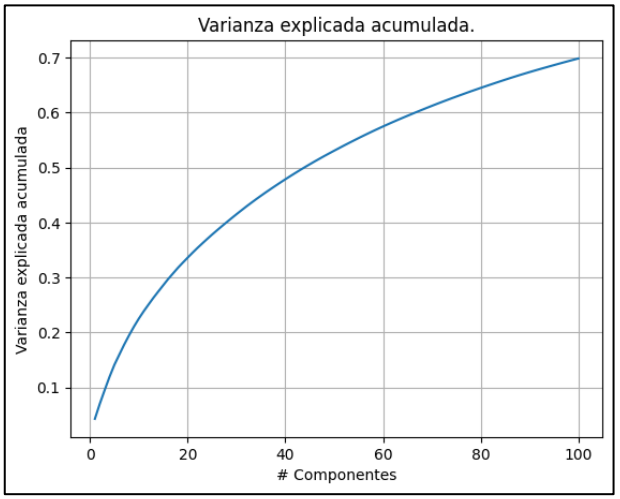
El siguiente paso es analizar cómo evoluciona la varianza explicada a medida que considero más componentes de los vectores. Para ello elegimos los 100 componentes más influyentes.

```
reductor100 = PCA(n_components=100)
X_train_red_100 = reductor100.fit_transform(X_train_tf.toarray())

varianza_exp = reductor100.explained_variance_ratio_
accum_var = np.cumsum(varianza_exp)
```

Código 5: obtención de varianza explicada y acumulada



Código 7: varianza explicada por cada componente



Código 6: varianza explicada acumulada

Al graficar lo obtenido, observamos que el primer componente, explica únicamente el 4% de la varianza de los datos. El segundo componente más importante, explica por debajo de 3%.

Viendo el acumulado, con 20 componentes, podremos explicar el 30% de la varianza y para explicar el 70% precisaremos 100 componentes.

Esto nos confirma que reducir la dimensión de nuestros vectores no es una opción válida para un clasificador que tenga una precisión aceptable.

4. Ejercicio 2

4.1. Ejercicio 2.1

Se procede a entrenar un modelo *Multinomial Naive Bayes* para tratar de clasificar nuestros párrafos.

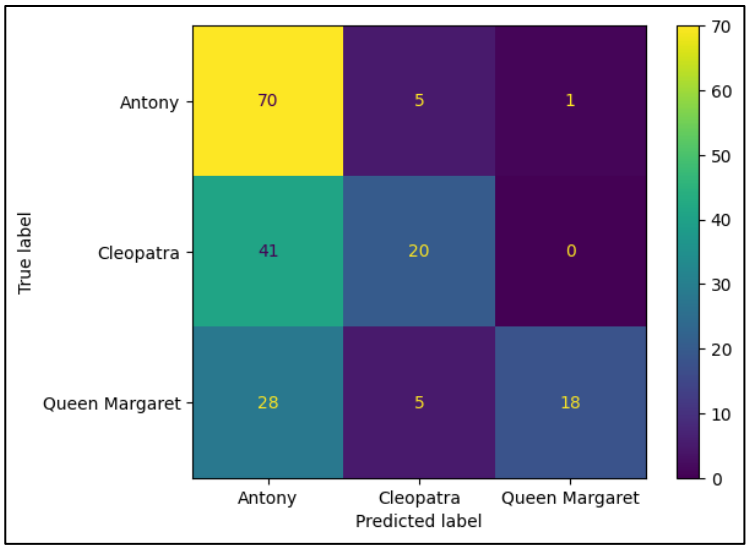
```
# Entrenamiento del modelo
bayes_clf_idf = MultinomialNB().fit(X_train_tf_idf, y_train)

#transformar el conjunto de test al espacio de vectores.
X_test_counts = count_vect2.transform(X_test)
X_test_tfidf = tf_idf.transform(X_test_counts)

# Predecir para test
y_test_pred = bayes_clf_idf.predict(X_test_tfidf)


# df de predicciones
df_pred = pd.DataFrame({'parrafo': X_test, 'valor_real': y_test,
                        'prediccion': y_test_pred})
df_pred
```

	parrafo	valor_real	prediccion
0	For the most part too they are foolish that ...	Cleopatra	Antony
1	Nay I ll help too What s this for	Cleopatra	Cleopatra
2	But I can give the loser leave to chide	Queen Margaret	Antony
3	Wither goes Vaux so fast what news I prithee	Queen Margaret	Cleopatra
4	As sweet as balm as soft as air as gentle ...	Cleopatra	Antony
...	...	...	...
183	This was the noblest Roman of them all All th...	Antony	Antony
184	No Caesar we will answer on their charge Ma...	Antony	Antony
185	I ll not believe but they ascend the sky And ...	Queen Margaret	Antony
186	Ay Lepidus	Antony	Antony
187	His legs bestrid the ocean his rear d arm Cre...	Cleopatra	Antony

188 rows × 3 columns

Código 8: Entrenamiendo del modelo y predicción en test

A simple vista, las predicciones del modelo no parecen buenas. El valor de *accuracy* obtenido fue de 0,574.



Código 9: Matriz de confusión

La matriz de confusión se puede leer de la siguiente forma:

- En aquellas ocasiones en que el modelo predijo que el párrafo pertenecía a Antony, 70 veces le acertó, 41 veces pertenecían a Cleopatra y 28 a Queen Margaret.

- De los párrafos que el modelo clasificó como pertenecientes a Cleopatra, 20 fueron aciertos, mientras que 5 pertenecían en realidad a Antony y otros 5 a Queen Margaret.

- De los párrafos que el modelo clasificó como pertenecientes a Queen Margaret, 18 fueron aciertos, mientras que 1 era en realidad de Antony.

Otra lectura posible es la siguiente:

- De los 76 párrafos de Antony que había en mi conjunto de test, el modelo clasificó 70 de manera correcta, 5 los clasificó pertenecientes a Cleopatra y 1 a Queen Margaret.

- De los 64 párrafos de Cleopatra, el modelo clasificó 20 correctamente y 41 los clasificó con la clase Antony.

- De los 51 párrafos de Margaret, 18 fueron bien clasificados, 28 se clasificaron como Antony y 5 como Cleopatra.

Estas distintas miradas, son las definiciones de ***precision*** y ***recall*** en ese orden respectivamente.

Veamos nuevas matrices de confusión, pero esta vez normalizadas para ver estos dos conceptos en valores entre 0 y 1.

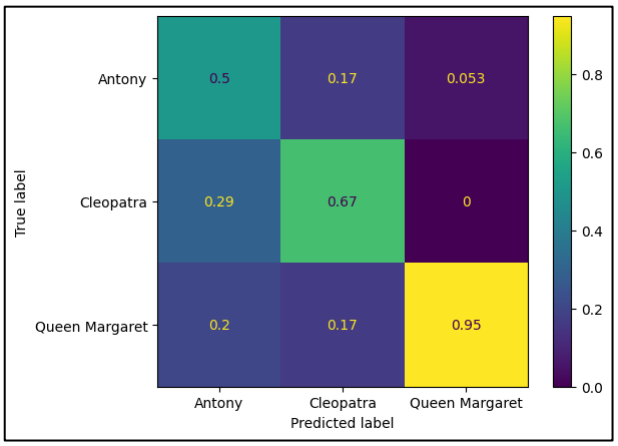


Ilustración 5: Matriz de precision

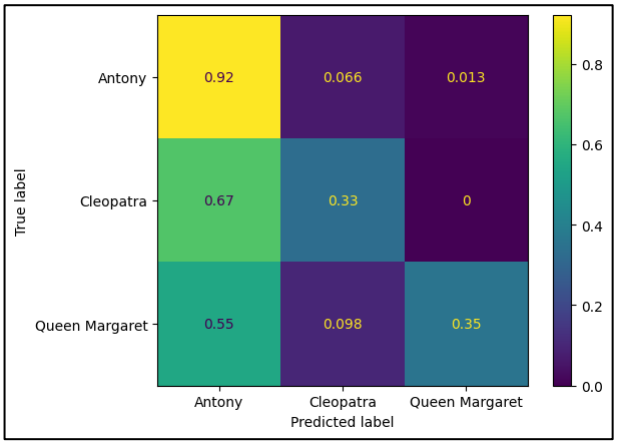


Ilustración 6: Matriz de recall

	precision	recall	f1-score	support
Antony	0.503597	0.921053	0.651163	76.0
Cleopatra	0.666667	0.327869	0.439560	61.0
Queen Margaret	0.947368	0.352941	0.514286	51.0

accuracy	0.574468
macro avg	0.705877
weighted avg	0.676893

Tabla 1: Métricas del modelo

Por su parte, la accuracy, se mide como la cantidad de aciertos en la predicción sobre la cantidad total de predicciones hechas. Se puede usar como medida de calidad de un modelo, siempre y cuando se entienda bien su alcance y sobre todo sus limitaciones.

Por ejemplo, en un conjunto de sujetos, donde el 70% de ellos pertenecen a la misma clase, un clasificador que siempre tenga esa misma clase por respuesta, sin importar la entrada, tendrá a priori, un accuracy del 70%. Es por esto que resulta inteligente analizar varias métricas y combinaciones razonables entre ellas para dimensionar la calidad de un modelo.

4.2. Ejercicio 2.2

La validación cruzada es una técnica que se basa en dividir el conjunto de entrenamiento en n subconjuntos. En la primera iteración, quita el primero de los n subconjuntos del conjunto de entrenamiento y ajusta un modelo con los n-1 conjuntos restante. Luego de entrenado, valida qué tan bien ajusta ese modelo al subconjunto que no se usó en el entrenamiento.

En la segunda iteración, agrega nuevamente el primer subconjunto a los datos de training, pero quita al segundo. Una vez más ajusta un modelo con esos datos y usa el subconjunto que dejó fuera para validar. Así iterará n veces, dejando cada vez a un subconjunto afuera del entrenamiento y usándolo para validar el modelo entrenado.

Finalmente se podrá emitir juicio en cuanto a la performance de cada modelo ajustado y elegir el que mejor ajuste logró.

Para seleccionar mejor los parámetros para nuestro clasificador, decidimos usar el método de cross-validation llamado Stratified K Fold, que hace lo explicado anteriormente, dividiendo el conjunto de entrenamiento en K subconjuntos (en nuestro caso serán 4), pero cuidando que todos

tengan las mismas proporciones en las clases de sus sujetos; de ahí el “stratified” en el nombre.

```
# Agregar más variantes de parámetros que les parezcan relevantes
param_stop_words = [['mi lista',sw], ['english','english'], ['None', None]]
param_ngram = [(1,1), (1,2), (2,2), (1,3)]
param_idf = [True, False]
param_sets = []
for p_idf in param_idf:
  for p_ngram in param_ngram:
    for p_stopwords in param_stop_words:
      param_sets.append(
          {"stop_words": p_stopwords, "ngram": p_ngram, "idf": p_idf})
```

Código 10: generación de conjunto de parámetros

El código que se ilustra, genera un arreglo con todos las combinaciones de parámateros que nos interesa comparar mediante el método de cross-validation.

Para generarlo se creó una lista por cada parámetro, conteniendo los valores posibles y luego con iteraciones anidadas, se generaron todas las combinaciones posibles entre estos valores:

- **Stop-words:** Los valores posibles son None, o sea no usar ninguna, ‘english’ que refiere a la lista por defecto de palabras en inglés, ya incorporada en la librería sklearn o ‘mi lista’ que es una lista definida por nosotros, que contiene las palabras en inglés de la librería *nltk*, sumadas a las palabras del inglés antiguo, cargadas desde esta lista.

- **N-grams:** los valores posibles son (1,1) o sea bag of words, (1,2) palabras y bigramas, (2,2) únicamente bigramas y (1,3) palabras, bigramas y trigramas.

- **IDF:** True o False: usar método de frecuencia inversa o no.

El código para generar nuestra lista de stop-words que incluya palabras del inglés antiguo, es el siguiente:

```
# Configuracón de stopwords
stopwords.words('english')
sw = nltk.corpus.stopwords.words('english')
old_stop_words = []
# Lista online de stop-words de inglés antiguo
old_stopwords_txt = urllib.request.urlopen(
  "http://earlymodernconversions.com/wp-content/uploads/2013/12/stopwords.txt")

for line in old_stopwords_txt:
    old_stop_words.append(line.decode('utf-8').replace('\n',''))
# agrego palabras a la lista
sw.extend(old_stop_words)
```

Código 11: stop words con palabras antiguas

Por cada una de las combinaciones de valores posibles para cada parámetro (24 combinaciones en total), se hizo 4 iteraciones según lo explicado para validación cruzada.

Al final de las 4 iteraciones, registramos el promedio de los 4 valores de *accuracy* obtenidos para esa configuración de parámetros y los promediamos. De esa manera, pudimos elegir, al final del camino, el mejor promedio de *accuracy*.

```
mejor_accuracy = 0.0
mejor_params = None
accuracies =  {}
for params in param_sets:
  vis =[params["stop_words"][0], params["ngram"], params["idf"]]
  accuracies[str(vis)] = []
  # Transormaciones a aplicar (featurizers)
  count_vect = CountVectorizer(stop_words=params["stop_words"][1],
                               ngram_range=params["ngram"])
  tf_idf = TfidfTransformer(use_idf=params["idf"])
  sum_acc = 0.0

  for train_idxs, val_idxs in skf.split(X_dev, y_dev):
    # Train y validation para el split actual
    X_train_ = X_dev[train_idxs]
    y_train_ = y_dev[train_idxs]
    X_val = X_dev[val_idxs]
    y_val = y_dev[val_idxs]

    # Ajustamos y transformamos Train
    X_train_counts = count_vect.fit_transform(X_train_)
    X_train_tf = tf_idf.fit_transform(X_train_counts)

    # 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)
    vis =[params["stop_words"][0], params["ngram"], params["idf"]]
    print(f"{acc=:.4f} {vis=}")
    accuracies[str(vis)].append(acc)
    sum_acc += acc
  if (sum_acc/4.0) > mejor_accuracy:
    mejor_accuracy = (sum_acc/4.0)
    mejor_params = params
print(f"{mejor_accuracy=:.4f} {mejor_params=}")
```

Código 12: Búsqueda de hiperparámetros

Al imprimir en pantalla la mejor accuracy promedio la configuración de parámetros asociada a dicho resultado, obtuvimos lo siguiente:

```
mejor_accuracy=0.5844 mejor_params={'stop_words': ['english', 'english'], 'ngram': (1, 1), 'idf': True}
```

Sin embargo, al hacer un gráfico de violín, que no sólo considera la media del accuracy, sino también la desviación estándar de dicha métrica entre los 4 folds, observamos que la configuración

```
{ stop_words = 'mi lista', ngram = (1, 1), 'idf' = True }
```

parece mejor desde el punto de vista de la accuracy.

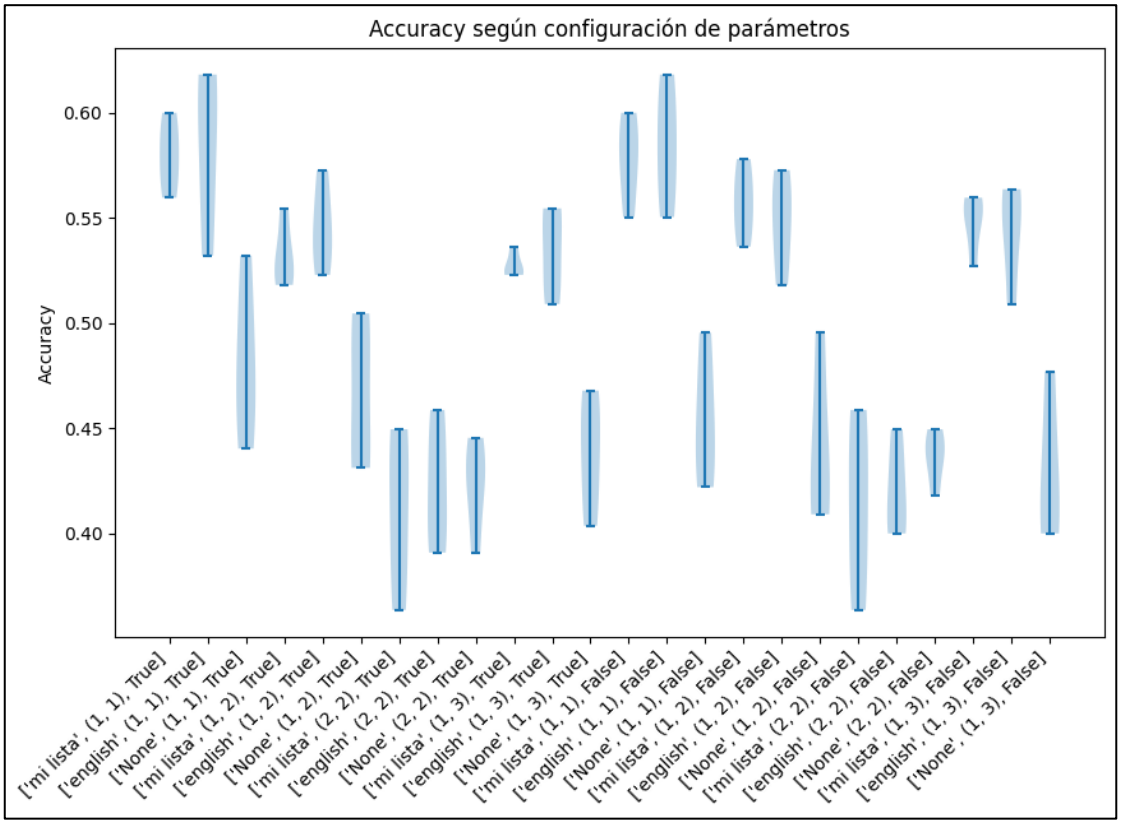


Ilustración 7: Accuracies de las configuraciones de parámetros

Esto se sustenta en que la mediana es apenas inferior (0.5799 versus 0.5844), pero tiene mucho menos variabilidad entre sus 4 componentes, lo cual es un aspecto deseable en un modelo.

Esta observación se puede ver gráficamente al observar los dos primeros violines, de izquierda a derecha, donde el segundo representa a la configuración de mejor media y el primero representa los que seleccionamos como mejor configuración.

4.3. Ejercicio 2.3

Elegido los mejores parámetros, volvemos a entrenar el modelo con todo el conjunto de entrenamiento y realizamos predicciones sobre el conjunto de test.

La matriz de confusión resultante de dichos pasos, es la siguiente:

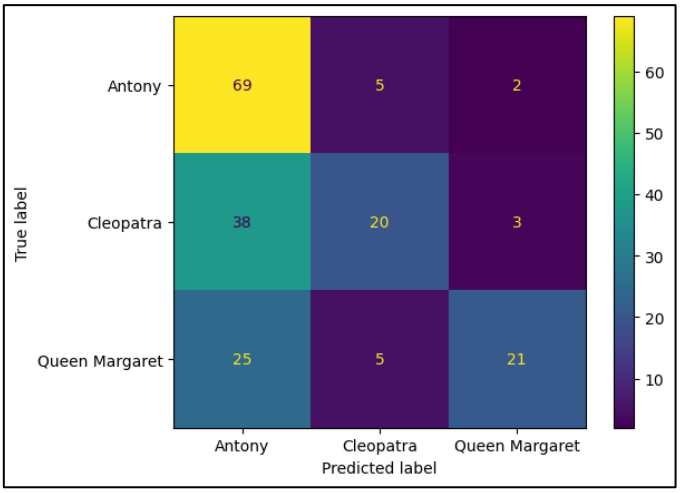
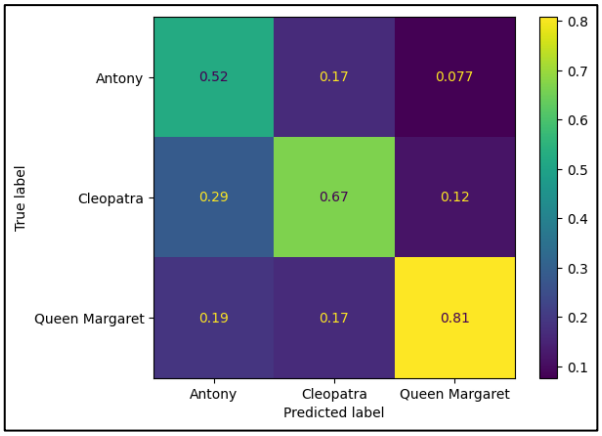
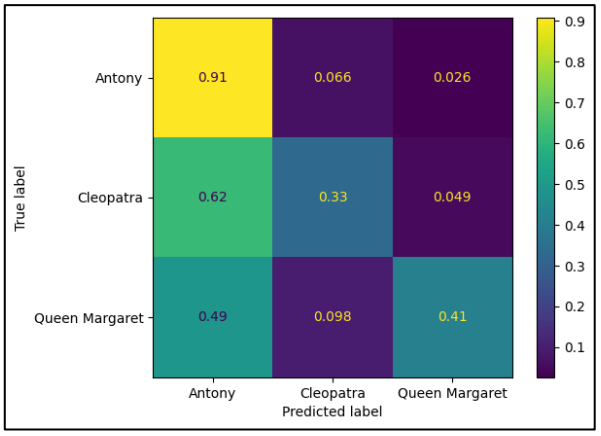


Ilustración 8: Matriz de confusión, parámetros óptimos

A simple vista, en comparación con el modelo que analizamos anteriormente, vemos que la nueva configuración de parámetros clasifica bastante mejor a Queen Margaret, aunque aún con métricas que distan mucho de ser buenas. En el resto de los personajes, la clasificación tiene métricas similares.

También podemos decir que se obtuvo un mejor accuracy (0,585 vs 0,574) y mejores registros de f1-score para las 3 clases. Esto último lo vemos como positivo, ya que lo único que buscaba mejorarse explícitamente era el accuracy, y el f1 score toma en cuenta tanto la precisión como el recall, por ende, parece un modelo mejor, aunque la diferencia no es estadísticamente significativa como para asegurarlo (de hecho, para asegurar algo de ese estilo, los violines de ambos modelos, en el gráfico presentado en la ilustración 7, deberían tener una intersección vacía, cosa que no ocurre).





	precision	recall	f1-score	support
Antony	0.522727	0.907895	0.663462	76.0
Cleopatra	0.666667	0.327869	0.439560	61.0
Queen Margaret	0.807692	0.411765	0.545455	51.0

accuracy	0.585106
macro avg	0.665695
weighted avg	0.646735

Tabla 2: Métricas del modelo

Limitaciones de los modelos BoW y TF-IDF.

Las estrategias vistas para extracción de features a partir de textos tienen algunos problemas conocidos.

Por ejemplo, no consideran la semántica, su orden ni los cambios que introducen en el significado, los signos de puntuación.

Dos oraciones pueden se opuestas y contener exactamente las mismas palabras, pero en distinto orden, o con una coma que articula esta diferencia, sin embargo, ambos métodos asignaran a las dos frases, el mismo vector.

“Juan es alto y José bajo”

“Juan es bajo y José alto”

Claramente son dos oraciones que refieren a distintos Juanes y Joseses, sin embargo, tendrán idéntica representación tanto en conteo de palabras como en TF-IDF.

Otras limitaciones ya comentadas, tienen que ver con la creciente dimencionalidad del espacio de vectores con un cuerpo de texto creciente, la cantidad de entradas con valores en 0 que tendrá cada vector y la incapacidad de generalizar palabras nuevas que no estaban en el cuerpo de entrenamiento, ya que se limitan a ver la ocurrencia de las palabras aprendidas en el entrenamiento.

Una tercera limitación tiene que ver con la carencia de un aspecto semántico de cada vocablo. Por ejemplo, palabras que son diferentes en su escritura y forma, pueden ser muy cercanas en cuanto a significado, sin embargo, esta distancia semántica no se contempla en estos modelos.

4.4. Ejercicio 2.4

Para profundizar aún más en nuestro análisis y clasificación de los datos, hemos elegido entrenar un modelo de clasificación de SGD (Stochastic Gradient Descent).

El modelo en sí, pertenece a la clase de modelos lineales. El algoritmo de ajuste utilizado por el modelo, encuentra los coeficientes óptimos de los clasificadores lineales, mediante la actualización de los parámetros del modelo, usando el gradiente de la función de pérdida.

En concreto y luego de algunas pruebas artesanales, hemos elegido la función *hinge* como función de pérdida, y *L2* como penalización en la función de costo. Con esta selección de parámetros el clasificador se convierte en un modelo SVM lineal (Support Vector Machine). El SVM intenta encontrar el “hiperplano” que mejor separa los elementos de dos clases diferentes. El mejor, para SVM, es el que maximiza la distancia entre el hiperplano en cuestión y los elementos más cercanos de ambas clases.

Como vimos, en principio, este es un modelo clasificador binario, pero se usa una estrategia de OvA (One versus All o uno contra todos) para obtener los hiperplanos que definen a las distintas clases.

La selección de dicho modelo, se basó en la amplia literatura existente en sitios web de análisis de datos, donde coinciden en las bondades de este modelo para abordar esta clase de problemas.

Veamos los resultados obtenidos al entrenar el modelo descrito:

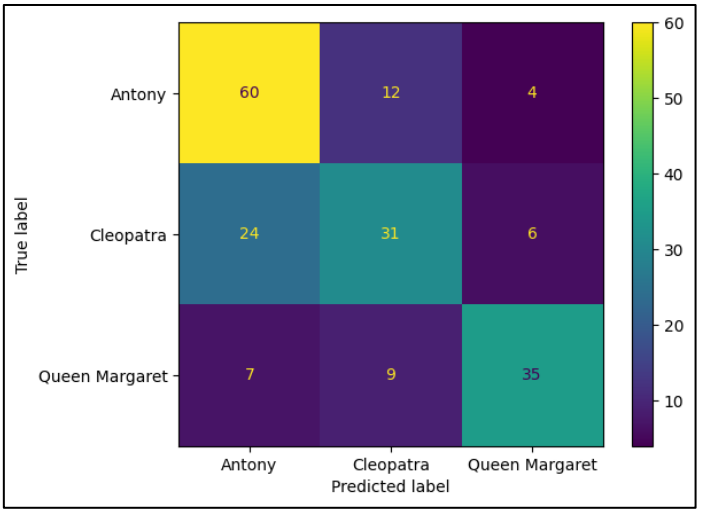


Ilustración 11: matriz de confusión SVM

Vemos claramente, que los números obtenidos en la diagonal principal son mayores a los obtenidos en los modelos analizados con anterioridad, augurando resultados mejores en todas las métricas.

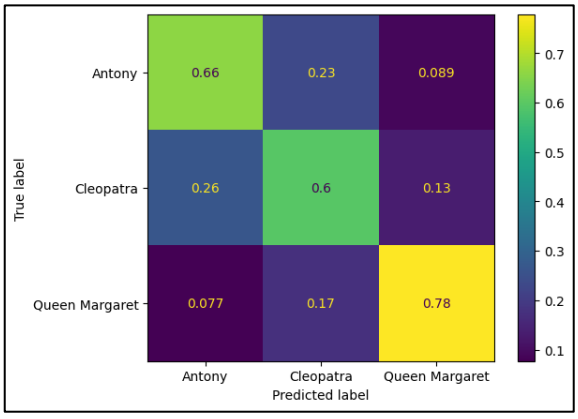


Ilustración 13: matriz de precision

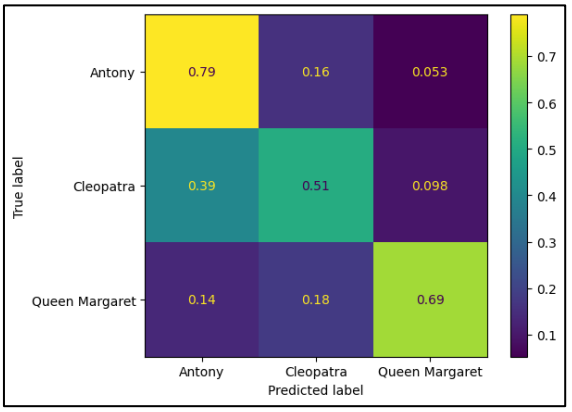


Ilustración 12: Matriz de recall

	precision	recall	f1-score	support
Antony	0.659341	0.789474	0.718563	76.0
Cleopatra	0.596154	0.508197	0.548673	61.0
Queen Margaret	0.777778	0.686275	0.729167	51.0

accuracy	0.670213
macro avg	0.677757
weighted avg	0.670968

Tabla 3: métricas del modelo SVM

El modelo SVM mejoró el accuracy de un 58% a un 67%, a su vez mejoraron varios puntos todas

las métricas de f1-score.

4.5. Ejercicio 2.5

Hemos decidido cambiar al personaje Antony por Poet, dado a que en la Tarea 1, descubrimos que este último es el personaje con mayor cantidad de párrafos y así poder graficar el problema del desbalance de datos.

Llevando a cabo dicho cambio, nuestros nuevos conjuntos de entrenamiento y test lucen así:

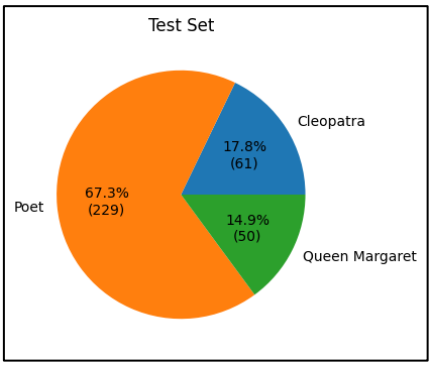


Ilustración 14: Nueva distribución Test

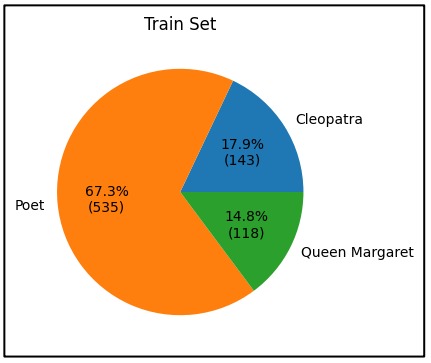


Ilustración 15: Nueva distribución train

Luego de realizar la vectorización de los párrafos y usando TF, reducimos el espacio de vectores a 2 dimensiones mediante PCA.

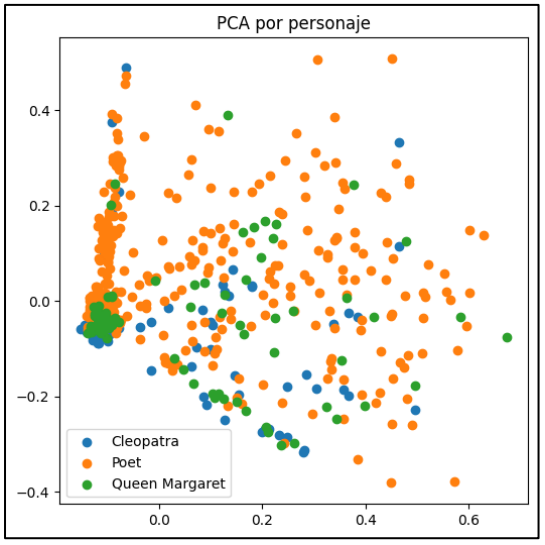


Ilustración 16: Mapeo PCA

Observamos como el color amarillo domina el plano y tiende a hacer más dificultosa la separación de dicha clase del resto.

Analizando luego la varianza explicada a medida que agregamos dimensiones al mapeo PCA, notamos que, a partir del cuarto componente, todos los componentes subsiguientes agregan menos de un 1% a la explicación de la varianza. De hecho, agregando los 100 componentes más importantes, logramos explicar únicamente un 42% de la varianza (contra un 70% explicado cuando aún teníamos un set más balanceado).

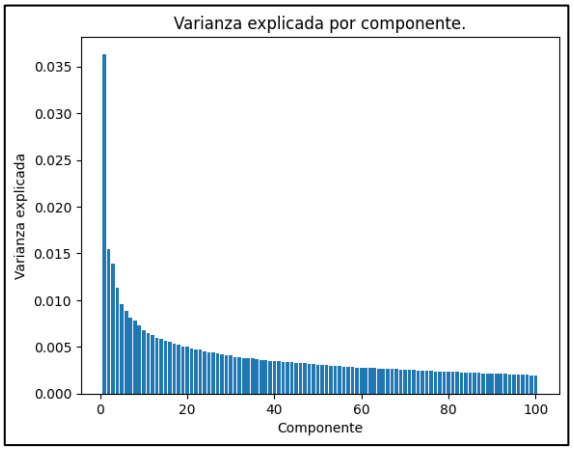


Ilustración 18: Varianza explicada por cada componente

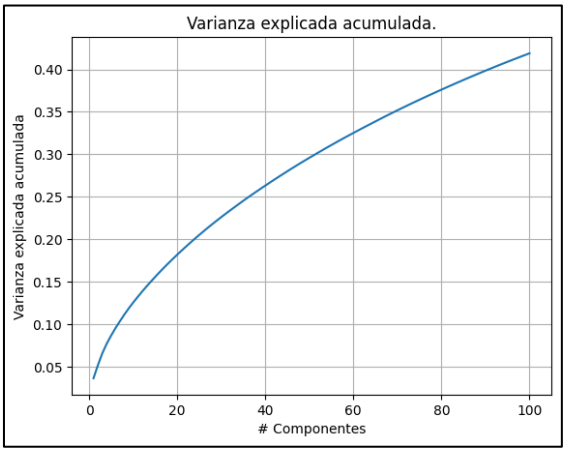


Ilustración 17: Varianza explicada acumulada

Luego entrenamos un modelo *Multinomial Naive Bayes*, con los parámetros elegidos en el ejercicio 1. La matriz de confusión obtenida fue la siguiente:

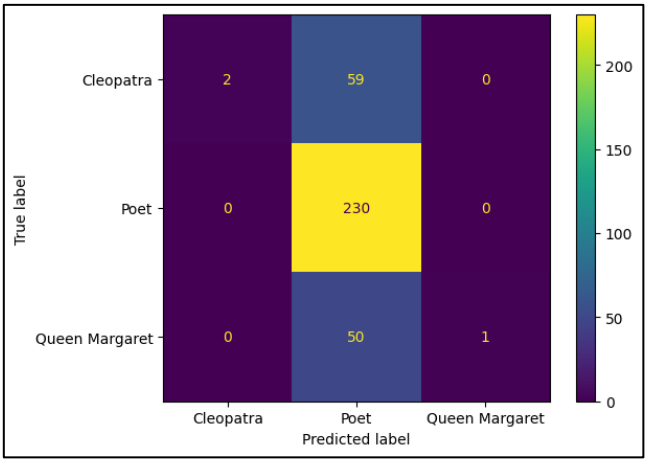


Ilustración 19: matriz de confusión MNB

Vemos que casi sin importar la verdadera etiqueta objetivo, el modelo entrenado clasifica la entrada como de la clase Poet casi en e l 100% de los casos.

	precision	recall	f1-score	support
Cleopatra	1.000000	0.032787	0.063492	61.0
Poet	0.678466	1.000000	0.808436	230.0
Queen Margaret	1.000000	0.019608	0.038462	51.0

accuracy	0.681287
macro avg	0.892822
weighted avg	0.783764

Tabla 4: Métricas MNB

Las métricas arrojadas por el modelo son muy malas para los personajes minoritarios. Sin embargo, son muy buenas (como es lógico) para el personaje Poet y, al tener este último una porción tan mayoritaria del conjunto de test, hace que el accuracy total, sea bueno (en comparación con los modelos vistos anteriormente). Esto reafirma la peligrosidad de usar el accuracy como única métrica de calidad del modelo.

El modelo SVM mejora mucho el resultado obtenido en cuanto a la clasificación obtenida, pero a grandes rasgos, sigue adoleciendo de el sesgo que el conjunto de datos de entrenamiento le impuso.

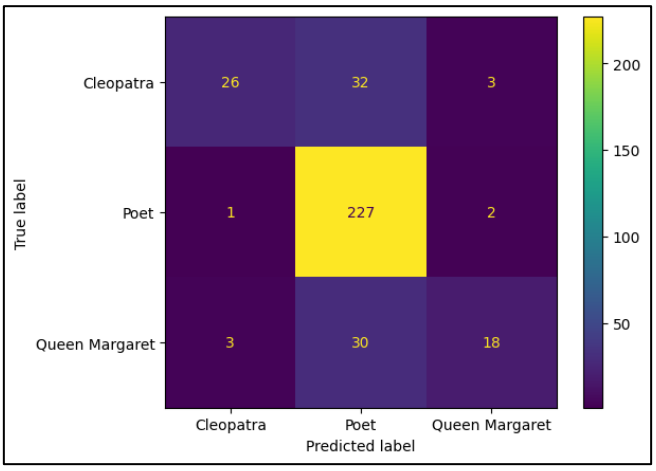


Ilustración 20: Matriz de confusión SVM

Vemos que el modelo SVM ya no clasifica absolutamente todas las muestras como Poet, aunque sigue siendo su salida mayoritaria.

	precision	recall	f1-score	support
Cleopatra	0.866667	0.426230	0.571429	61.0
Poet	0.785467	0.986957	0.874759	230.0
Queen Margaret	0.782609	0.352941	0.486486	51.0

accuracy	0.792398
macro avg	0.811581
weighted avg	0.799524

Tabla 5: Métricas SVM

Como era de esperar, las métricas son sensiblemente mejores que para el modelo MNB, y especialmente buenas para clasificar a los párrafos de Poet, siendo que para clasificar párrafos de los demás personajes, únicamente logra hacerlo correctamente en menos del 40% de los casos.

El problema observado en los clasificadores entrenados con el conjunto de datos desbalanceado, se debe a un sesgo impuesto a la hora del aprendizaje, a favor, justamente, de la clase predominante.

Existen maneras de mitigar el imacto de este sesgo, como ser:

- **Submuestreo**: consiste en eliminar aleatoriamente datos del conjunto de entrenamiento, pertenencientes a la clase mayoritaria, hasta tanto lograr un razonable balance entre las clases.

- **Sobremuestreo**: consiste en agregar datos de clases minoritarias, por ejemplo generando nuevos datos por extrapolación de datos existentes (técnicoa SMOTE) o simplemente replicando sujetos ya conocidos.

- **Híbrido**: ir usando ambos abordajes hasta converger en un balance medio.

4.6. Ejercicio 2.6

Técnica alternativa de extracción de features a partir de texto.

Una de las técnicas para procesamiento de lenguaje natural, se llama word2vec. La misma usa un modelo de redes neuronales (generalmente poco profundas, 2 capas) para aprender asociación entre palabras. Dicho aprendizaje se hace a partir de un extenso cuerpo de texto y una vez entrenado, el modelo puede detectar sinónimos, palabras que usualmente vienen seguidas en las oraciones y otra gran cantidad de nociones de distancia y cercanía entre palabras. Word2vec, representa cada palabra como un vector, los cuales selecciona cuidadosamente para capturar su semántica y cualidades sintácticas. Luego, a través de esa representación, se puede calcular la distancia entre vectores para obtener el grado de similitud semántica entre las palabras.

La técnica está dentro de un grupo de modelos relacionados que se utilizan para producir Word embeddings. Los word embeddings están colocados en el espacio vectorial de forma que las palabras que comparten contextos comunes en el corpus están localizadas cerca unas de otras en el espacio.

Utilizar esta técnica, soluciona varios de los problemas de BoW y TF-IDF descritos en el ejercicio 2.3 y sin la necesidad de tener previamente una ontología de palabras.