UNIVERSIDAD DE LA REPÚBLICA ORIENTAL DEL URUGUAY
FACULTAD DE INGENIERÍA
INSTITUTO DE COMPUTACIÓN

INTRODUCCIÓN A LA CIENCIA DE DATOS
Junio de 2025

Entrega 2

GRUPO 11

Karina Cardozo – CI.: 4.135.872-7
Karen García - CI.: 4.945.279-5

INDICE

1 Parte 1: Dataset y representación numérica de texto

1.1 PARTE A

En el notebook de la tarea, se utilizará un conjunto de datos reducido de los tres candidatos con más discursos. Se espera que utilice su propia versión de la función clean_text() de la Tarea 1. Particione los datos para generar un conjunto de test del 30% del total, utilizando muestreo estratificado. Sugerencia: utilice el parámetro stratify de la función train_test_split de scikit-learn y fije también el valor de random_state para obtener resultados reproducibles.

1.1.1 Marco Teórico

Conjuntos Train - Test

Para entrenar modelos de aprendizaje automático sobre texto se separa el conjunto completo de datos en:

- Conjunto de entrenamiento (train): datos usados exclusivamente para ajustar parámetros del modelo.
- Conjunto de prueba (test): datos reservados para evaluar el rendimiento final en ejemplos que el modelo no ha visto.

Esta práctica busca estimar de forma confiable la capacidad de generalización del modelo.
Se destina alrededor del 30 % de los datos a test porque si se usa muy poco (menos del 20 %), las métricas pueden ser poco confiables. Y si se usa demasiado (más del 40 %), el modelo no tiene suficientes ejemplos para aprender bien, sobre todo si hay varias clases. Por eso, usar un 30 % para test es lo más recomendable y funciona bien en la mayoría de los casos, incluso cuando no hay tantos datos.

Muestreo Estratificado: Objetivos y Beneficios

Cuando nuestro dataset contiene varias clases (en este caso, discursos de distintos candidatos), es crucial que el conjunto de prueba (test) mantenga la misma proporción de cada clase que existe en el total.
Sin estratificar, el proceso aleatorio podría asignar muy pocos discursos de un candidato (por ejemplo, “Bernie Sanders”) al test, o incluso dejar alguna clase ausente. Eso distorsionaría las métricas finales: el accuracy, precision o recall en test no reflejarían el rendimiento real sobre ejemplos nuevos, pues la muestra de prueba no sería representativa.
Al usar `stratify=y` (donde `y` es la variable “candidato”), garantizamos que cada clase preserve aproximadamente el mismo reparto 70 % train / 30 % test:
Para “Joe Biden”, si hay 100 discursos en total, 70 irán a train y 30 a test.
Para “Donald Trump”, si hay 50 discursos, 35 irán a train y 15 a test.
Lo mismo para “Bernie Sanders” u otras clases presentes.
Con esto logramos:

- **Evaluación confiable:** Las métricas calculadas en test reflejan fielmente la capacidad del modelo para distinguir discursos de cada candidato.
- **Minimizar el sesgo de muestreo:** Sin estratificación, podríamos tener muy pocos ejemplos de un candidato en test, haciendo sus métricas poco fiables o imposibles de medir. Con estratificación, ese riesgo se elimina.

Reproducibilidad

Al fijar `random_state`, cada ejecución de `train_test_split` con los mismos datos produce

exactamente las mismas particiones train/test. Esto permite comparar ajustes y modelos de forma consistente, porque todos usan los mismos conjuntos.

1.1.2 Metodología

Se parte de un DataFrame, logrado en la tarea 1, que ya contiene los datos limpios. Cada fila representa un discurso completo, y además de las columnas originales, se incorporaron cinco columnas adicionales: una por cada uno de los cinco candidatos con más discursos. En estas columnas, donde el nombre coincide con el orador normalizado, se incluye únicamente el texto correspondiente a su intervención, ya normalizado y con los encabezados extraídos según lo definido en la Tarea 1.

A partir de ese DataFrame inicial, se seleccionaron únicamente las tres columnas correspondientes a los candidatos con mayor cantidad de discursos. Se aplicó la función `melt()` para convertir el DataFrame de formato ancho (una columna por candidato) a formato largo, en el que cada fila representa un bloque de discurso asociado a su respectivo orador.

Se aplicó una función personalizada `clean_text()` desarrollada en etapas anteriores para normalizar el contenido textual: conversión a minúsculas, remoción de puntuación y tildes, eliminación de encabezados irrelevantes, reducción de espacios múltiples.
Se creó el DataFrame `df_clf`, que contiene únicamente dos columnas:

- `clean_text`: texto limpio del discurso
- `target`: nombre del orador (etiqueta)

Esta estructura es más adecuada para tareas de clasificación supervisada.

Se dividieron los datos en dos subconjuntos:

- **70 %** para entrenamiento/desarrollo (`X_dev`, `y_dev`)
- **30 %** para prueba (`X_test`, `y_test`)

`X_dev`: Contiene los **textos ya preprocesados (clean_text)** que se usarán para entrenar y ajustar el modelo. Es un subconjunto del total, con el 70 % de los discursos.
`y_dev`: Contiene las **etiquetas** (oradores) correspondientes a cada texto en `X_dev`. Es decir, es la columna `target` para ese 70 % de textos.
`X_test`: Contiene el 30 % restante de los textos, que **no se usarán para entrenar**, sino solo para evaluar qué tan bien funciona el modelo en datos nuevos.
`y_test`: Contiene las etiquetas reales (oradores) de los textos en `X_test`, que se usarán para comparar con las predicciones del modelo.

Esta partición se realizó con muestreo **estratificado** (usando `stratify=y`) y además se fijó `random_state=42` para asegurar la **reproducibilidad** del experimento.

1.2 PARTE B

Genere una visualización que permita verificar que el balance de discursos de cada candidato es similar en train y test.

1.2.1 Metodología

Se calcularon los conteos de discursos por candidato en cada subconjunto (`y_dev` y `y_test`) utilizando `value_counts()` para confirmar que se haya respetado la proporción original en la partición, y se generó una visualización.

1.2.2 Resultado

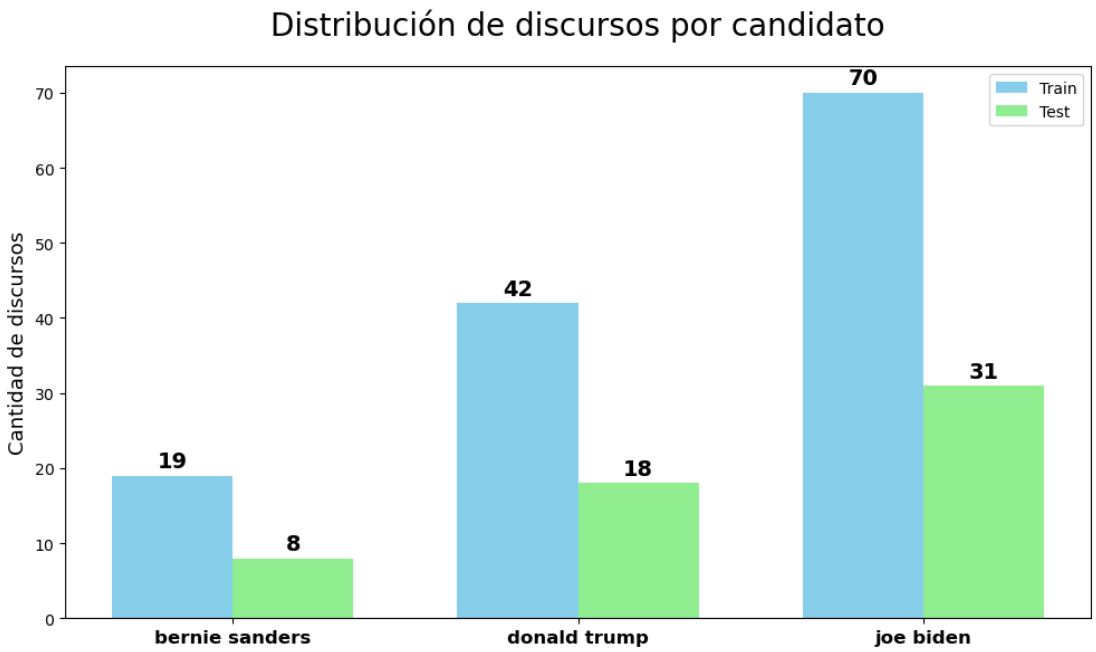


Figura 1: Distribución de discursos por candidato

Se obtuvieron los siguientes valores:

	% train	% test
Joe Biden	69,31%	30,69%
Donald Trump	70,00%	30,00%
Bernie Sanders	70,37%	29,63%

Tabla 1: % de División final de train y test por clase

Los resultados confirman que las **proporciones relativas se mantienen estables** en ambos conjuntos, como era de esperarse al aplicar el muestreo estratificado. Esto garantiza que los modelos entrenados no estarán sesgados por una distribución desigual de clases entre entrenamiento y evaluación.

1.3 PARTE C

Transforme el texto del conjunto de entrenamiento a la representación numérica (features) de conteo de palabras o bag of words. Explique brevemente cómo funciona esta técnica y muestre un ejemplo. En particular, explique el tamaño de la matriz resultante y la razón por la que es una matriz es sparse. Sugerencia: puede ser útil imaginar qué sucedería con la memoria RAM requerida si no estuviéramos trabajando con un conjunto de datos tan reducido.

1.3.1 Marco Teórico

En esta parte de la tarea se pide transformar el texto del conjunto de entrenamiento en una representación numérica conocida como **Bag of Words** (bolsa de palabras).
La técnica **Bag of Words (BoW)** representa cada documento (cada muestra) como un vector numérico en función de la frecuencia con la que aparecen las palabras, ignorando el orden en que estas aparecen. Se parte de un vocabulario formado por todas las palabras distintas en el corpus de entrenamiento, y cada texto se convierte en un vector cuya longitud es igual al tamaño de ese vocabulario.
Por ejemplo:
Si el vocabulario contiene las palabras: ***['freedom', 'hope', 'justice', 'nation']***
y el documento es**:** **"hope and justice for our nation"**
el vector resultante sería**:** **[0 , 1 , 1 , 1],**
ya que aparecen las palabras ***hope***, ***justice*** **y** ***nation*** una vez cada una, y ***freedom*** no aparece.

Este enfoque permite transformar texto en una representación numérica apta para ser utilizada por modelos de machine learning, sin requerir comprensión semántica ni sintáctica del lenguaje.

1.3.2 Metodología

Se utilizó la clase `CountVectorizer` de la biblioteca **scikit-learn**, que permite vectorizar texto mediante conteo de términos.
El procedimiento constó de dos pasos principales:

- **Creación del vectorizador** con `CountVectorizer()`.
- **Ajuste y transformación del texto** mediante `fit_transform(X_dev)`, lo cual construyó automáticamente un vocabulario a partir del conjunto de desarrollo (`X_dev`) y transformó cada discurso en un vector de conteos.

1.3.3 Resultado

El resultado fue una **matriz de conteo** de tamaño (131, 13068), donde:

- **131** corresponde a la cantidad de discursos en el conjunto de entrenamiento.
- **13.068** es la cantidad de palabras distintas (vocabulario total) detectadas.

Esta matriz es de tipo **sparse matrix** (matriz dispersa) porque la mayoría de los elementos son ceros: cada discurso solo utiliza una fracción muy pequeña del vocabulario total. Por esta razón, scikit-learn utiliza una representación comprimida para optimizar el uso de memoria.

Cuando se transforma texto a números mediante la técnica **Bag of Words**, se construye una **matriz** en la que:

- Cada **fila** representa un discurso (documento).
- Cada **columna** representa una palabra distinta del vocabulario total.
- Cada **celda** contiene el número de veces que esa palabra aparece en ese discurso.

Tenemos

- 131 discursos → 131 filas
- 13.068 palabras distintas → 13.068 columnas

Esto da una **matriz teórica de 131 × 13.068 = 1.712.908 elementos**.

Sin embargo, **cada discurso solo utiliza una pequeña fracción de todas esas palabras**. Por ejemplo, un discurso puede contener 300 o 500 palabras distintas, y *no usar* el resto.

Entonces, la gran mayoría de las celdas de la matriz contienen **ceros**, ya que la palabra correspondiente **no aparece** en ese discurso.

Esto se resuleve utilizando matrices dispersas. Una matriz dispersa es una estructura de datos que **almacena únicamente los valores diferentes de cero**, junto con sus posiciones.
No guarda todos los ceros explícitamente, lo que permite **ahorrar muchísima memoria y acelerar los cálculos**.

Librerías como `scikit-learn` usan estructuras como **CSR (Compressed Sparse Row)** O **COO (Coordinate format)**, que guardan:

- El valor distinto de cero (por ejemplo: 3)
- La posición en la fila y la columna donde ocurre

Ejemplo de representación:

Fila	Columna	Valor
0	53	2
0	106	1
1	5	1

Esto indica que la palabra número 53 aparece 2 veces en el discurso 0, la palabra 106 aparece una vez, etc.

En nuestro caso, trabajamos con **131 discursos** y un vocabulario total de **13.068 palabras distintas**, lo que da como resultado una matriz de conteo teórica de **más de 1,7 millones de celdas** (131 × 13.068). Aunque la mayoría de esas celdas contienen ceros (por eso usamos una *matriz dispersa*), la cantidad total de elementos ya da una primera idea del volumen a manejar.

Si **el dataset fuese más grande.** Por ejemplo 10.000 discursos con un vocabulario total de 50.000 palabras distintas, eso generaría una matriz de **500 millones de celdas** (10.000 × 50.000).

Si intentáramos almacenar esa matriz como una estructura densa (es decir, guardando todos los ceros explícitamente), necesitaríamos 500 millones de elementos × 8 bytes (para enteros de 64

bits) = **4.000 millones de bytes, esto es, 4 GB de RAM**. Solo para esa matriz. Sin contar el resto del modelo, los datos originales, ni otros procesos en memoria (y en la práctica, **más del 95 % de esas celdas estarían vacías** (con valor cero), ya que ningún documento utiliza el vocabulario completo).

Si no se usara una representación dispersa (como las que maneja `scikit-learn` por defecto con `CountVectorizer` o `TfidfVectorizer`), **el análisis de texto sería inviable** en computadoras personales. El uso de *matrices sparse* es lo que permite escalar estos métodos a corpus reales (por ejemplo, millones de documentos) **sin colapsar la memoria RAM**.

1.4 PARTE D

Explique brevemente qué es un n-grama. Obtenga la representación numérica Term Frequency - Inverse Document Frequency. Explique brevemente en qué consiste esta transformación adicional.

1.4.1 Marco Teórico

n-gramas

Un **n-grama** es una secuencia continua de *n* palabras tomadas de un texto. Por ejemplo, en la frase:
"we need change"
Los unigramas (1-gramas) son: `"we"`, `"need"`, `"change"`
Los bigramas (2-gramas) son: `"we need"`, `"need change"`
Los trigramas (3-gramas) serían: `"we need change"`
Los n-gramas permiten capturar no solo la frecuencia de palabras individuales, sino también patrones de co-ocurrencia y frases comunes, lo que enriquece la representación del texto.

TF-IDF

TF-IDF es una técnica que convierte el texto en una **matriz numérica ponderada**, buscando reflejar no sólo cuántas veces aparece una palabra (frecuencia), sino también **qué tan relevante es esa palabra** en relación con el resto del corpus.
TF (Term Frequency) mide cuántas veces aparece una palabra en un documento.
IDF (Inverse Document Frequency) penaliza aquellas palabras que aparecen en muchos documentos (como "the", "people") y premia las más distintivas.
El resultado es una matriz en la que cada valor indica la *importancia relativa* de una palabra para un discurso en particular.

TfidfTransformer

La clase `TfidfTransformer` **toma una matriz de conteo de palabras** (como la que genera `CountVectorizer`) y **la transforma en una matriz de valores TF-IDF**, donde cada celda representa la importancia relativa de una palabra en un documento específico.

TF Term Frequency

Para cada palabra en un documento:

$$TF_{ij} = \frac{\text{frecuencia de la palabra } j \text{ en el documento } i}{\text{total de palabras en el documento } i}$$

Esto convierte los conteos a proporciones dentro del mismo documento.

IDF (Inverse Document Frequency)
Este valor busca penalizar las palabras que aparecen en **muchos documentos**, ya que probablemente no aportan información distintiva.

$$IDF_j = \log\left(\frac{N+1}{df_j+1}\right)+1$$

N: número total de documentos
dfj: cantidad de documentos donde aparece la palabra j

Si una palabra aparece en todos los documentos, su IDF será bajo. Si una palabra aparece solo en uno o pocos documentos, su IDF será alto.

Se multiplica TF por IDF

El valor final de cada celda en la matriz TF-IDF es:

$$TFIDF_{ij} = TF_{ij} \times IDF_j$$

Esto pondera cada frecuencia según **la relevancia de esa palabra** en el corpus completo.

Finalmente se tiene que:
- Palabras muy frecuentes pero poco informativas (como “the”, “we”, “people”) tendrán valores bajos en todas las filas.
- Palabras que aparecen con frecuencia solo en ciertos discursos (como “unity”, “change”) tendrán valores altos en esos discursos, y cero en los demás.
- El resultado final ya no son conteos enteros, sino **valores decimales normalizados** que indican “peso” o “importancia relativa”.

A diferencia de Bag of Words, que trata igual a todas las palabras, TF-IDF:
- Suprime palabras comunes en todos los discursos
- Destaca palabras que son relevantes para un discurso en particular

1.4.2 Metodología

Se reutilizó la matriz de conteo obtenida con Bag of Words (`X_dev_counts`) y se aplicó la transformación TF-IDF utilizando `TfidfTransformer` de `scikit-learn`:

```
tfidf_transformer = TfidfTransformer()
X_dev_tfidf = tfidf_transformer.fit_transform(X_dev_counts)
```

Como se explicó, esta transformación no altera la cantidad de filas ni columnas (es decir, sigue habiendo 131 discursos y 13.068 palabras), pero **cambia los valores dentro de la matriz**, asignando mayor peso a las palabras más informativas.

1.4.3 Resultado

La matriz resultante tiene la siguiente forma:
Shape de la matriz TF-IDF: (131, 13068)

Esto significa:
- 131 discursos (uno por fila)
- 13.068 términos únicos en el vocabulario (uno por columna)

Cada celda contiene ahora un valor decimal (float) que representa la importancia relativa de la palabra en ese discurso. Al igual que con Bag of Words, esta matriz es **dispersa**: la mayoría de las celdas son cero, porque cada discurso usa solo una pequeña parte del vocabulario total.

1.5 PARTE E

Muestre en un mapa el conjunto de entrenamiento, utilizando las dos primeras componentes PCA sobre los vectores de tf-idf. Analice los resultados y compare qué sucede si utiliza el filtrado de stop_words para idioma inglés, el parámetro use_idf=True y ngram_range=(1,2). Opcionalmente, también puede analizar qué sucede si no elimina los signos de puntuación. ¿Se pueden separar los candidatos utilizando sólo 2 componentes principales? Haga una visualización que permita entender cómo varía la varianza explicada a medida que se agregan componentes (e.g: hasta 10 componentes).

1.5.1 Marco teórico

En este apartado, para explorar cómo distintos ajustes en la extracción de características de texto influyen en la capacidad de distinguir discursos de diferentes candidatos, se recurre a dos técnicas clave: **TF–IDF** y **Análisis de Componentes Principales (PCA)**:
En paralelo, se experimenta con variantes de `CountVectorizer` para ver si:

- El filtrado de *stop words* en inglés elimina ruido genérico.
- La inclusión de *n-gramas* (unigramas + bigramas) aporta contexto adicional.
- Mantener la **puntuación** añade o genera ruido.

Análisis de Componentes Principales (PCA)
El PCA es una técnica estadística de reducción de dimensionalidad que busca descubrir las direcciones, componentes principales, en las que los datos varían más. Cada componente principal es una combinación lineal de las variables originales (en nuestro caso, las dimensiones del vector TF–IDF) y cumple dos propiedades clave:

- **Maximiza la varianza explicada.**
  La primera componente captura la mayor parte posible de la varianza global; la segunda captura la máxima varianza restante bajo la restricción de ser ortogonal a la primera, y así sucesivamente.
- **Ortogonalidad entre componentes.**
  Todas las componentes principales son mutuamente perpendiculares en el espacio de características, eliminando la redundancia de información.

La utilidad práctica de PCA radica en poder:

- **Visualizar** datos de alta dimensión en dos o tres dimensiones, revelando agrupamientos o patrones.
- **Reducir ruido** y eliminar correlaciones, mejorando el desempeño y la interpretabilidad de modelos posteriores.
- **Medir** cuánta varianza de los datos originales se conserva al descartar componentes de menor importancia.

Límite en el número de componentes
El número máximo de componentes extraíbles es el mínimo entre:

- **Número de muestras** (disursos): 131 en nuestro conjunto de entrenamiento.
- **Número de variables originales** (términos del vocabulario): 13 068 tras TF–IDF.

Por tanto, aunque tuviéramos 13 068 dimensiones, solo podemos definir hasta **131 componentes** (una por cada muestra), pues no puede haber más direcciones de variación independientes que observaciones.

1.5.2 Metodología

Preparación de los datos TF–IDF
Convertimos cada discurso a un vector TF–IDF de dimensión 13 068, donde cada coordenada mide la importancia relativa de una palabra. La matriz resultante de tamaño (131, 13 068) es **dispersa** (sparse) porque cada discurso suele emplear muy pocas de las miles de palabras posibles.

Conversión a formato denso
PCA de scikit-learn requiere matrices densas, por lo que transformamos nuestro "sparse matrix" a un array de NumPy de 131×13 068.

Cálculo de componentes
Se aplicó `PCA(n_components=k)` variando k desde 2 hasta 10 (y en un análisis más extenso hasta 131). Para cada k, se ajustó el modelo y se calculó la **varianza explicada acumulada**.

Visualización e interpretación
Proyección 2D: graficamos las dos primeras componentes para apreciar la separación visual de los discursos según el orador.
Curva de varianza: trazamos la proporción de varianza total explicada en función de k.

Configuraciones comparadas

- **default** (unigramas, sin stop words)
- **stopwords_en** (unigramas, stop words inglés)
- **ngram_1_2** (unigramas + bigramas, sin stop words)
- **stop_en_ngram_1_2** (unigramas + bigramas, stop words inglés)
- **keep_punctuation** (unigramas, sin stop words, tokenización que incluye signos)

Flujo para cada comparación:
a. Ajustar `CountVectorizer` con la configuración correspondiente.
b. Transformar el conjunto de entrenamiento (`X_dev`) a frecuencias de conteo y luego a TF–IDF.
c. Convertir la matriz TF–IDF dispersa a densa y aplicar `PCA(n_components=2)`.
d. Graficar los puntos resultantes en el plano (Componente 1 vs Componente 2), coloreando por candidato.

Visualización: Dos subgráficos lado a lado para comparar directamente cómo cambia la separación en función de cada variante analizada.

1.5.3 Resultados

Mapa 2D de discursos

A continuación se muestra el gráfico PCA=2 de los discursos políticos. Cada punto representa un discurso individual de uno de los tres candidatos analizados (Joe Biden -en azul-, Donald Trump -en naranja- y Bernie Sanders -en verde-). El eje X (Componente Principal 1) captura la dirección de máxima varianza en el conjunto de datos: es decir, la combinación lineal de palabras y temas que más distingue entre los discursos. Valores más positivos o negativos indican uso diferencial de cierto vocabulario o estilo. El eje Y (Componente Principal 2), muestra la segunda mayor fuente de variación ortogonal a la primera: puede reflejar diferencias en el tono.

Este diagrama permite visualizar cómo se agrupan o se separan los discursos según su estilo y contenido, así como detectar discursos atípicos ("outliers") en cada orador.

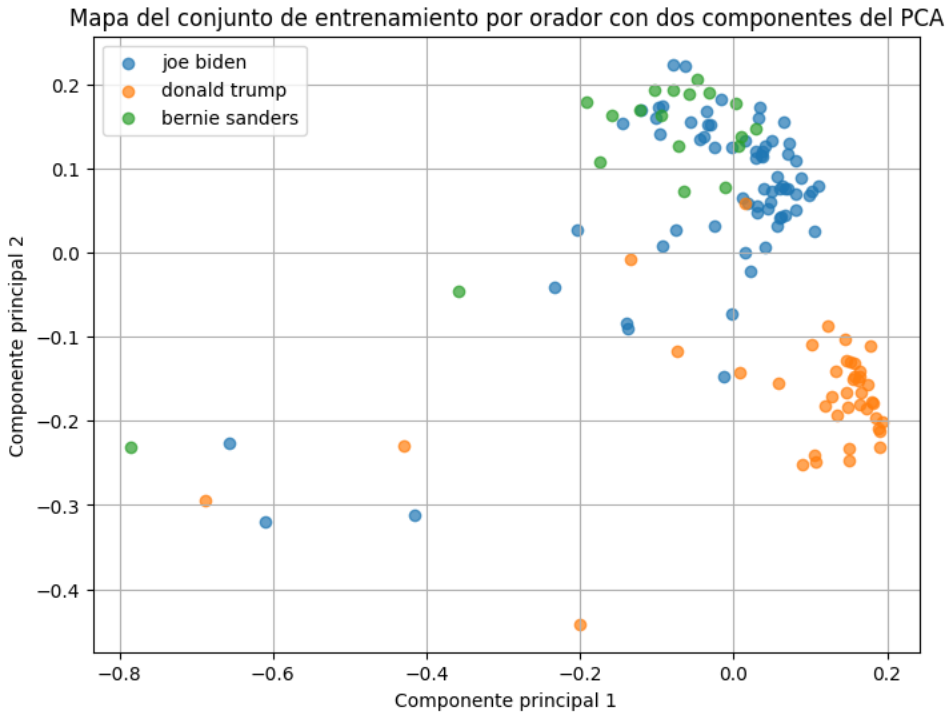


Figura 2: Mapa del conjunto de entrenamiento por orador con dos componentes del PCA

Separación entre oradores

- Donald Trump (naranja) aparece muy desplazado hacia la derecha (valores altos de PC1) y en su mayoría por debajo de PC2 = 0. Esto sugiere que su estilo o vocabulario se diferencia marcadamente de los otros dos, especialmente en lo que PC1 representa.
- Joe Biden (azul) se agrupa en la zona superior-derecha (PC1 cercano a cero o ligeramente positivo y PC2 positivo). Esto indica similitudes internas en su discurso y cierta proximidad temática o estilística a Trump en cuanto a PC1, pero claramente diferenciados en PC2.
- Bernie Sanders (verde) se distribuye en dos zonas: una agrupación cercana a Biden en la parte alta del gráfico (PC2 positivo), pero también algunos discursos más alejados hacia la izquierda (PC1 negativo).

Grado de solapamiento

- Hay cierto solapamiento entre Biden y Sanders en la zona alta, lo que sugiere que comparten temas o estilo narrativo (por ejemplo, discurso centrado en temas progresistas), pero ambos se distinguen claramente de Trump.
- La dispersión de Sanders es mayor, indicando que sus discursos varían más en contenido o tono a lo largo del tiempo, tiene sentido puesto que se cuenta con menos muestras lo que afecta su representatividad.

Grado de dispersión

- Los discursos de Trump se ven más aglomerados que los de los demás candidatos. Que los puntos naranjas aparezcan todos en una zona relativamente compacta indica que, a lo largo de la muestra, su estilo y vocabulario son muy consistentes. PC1 y PC2 capturan los ejes de variación más importante, y Trump no "se mueve" mucho en ninguno de ellos: reutiliza de forma muy estable un núcleo de palabras, temas y tonos. Esto se reafirma con en análisis de diversidad léxica realizado en la Tarea 1, donde se vió que Trump era el candidato con menor porcentaje de palabras diststintas en sus discursos.
- Biden y Sanders más dispersos: al contrario, la mayor dispersión de los azules (Biden) y verdes (Sanders) señala más diversidad interna: cambios de tema, registros (más o menos formales, más optimistas o críticos), tal vez diferentes formatos (conferencias, entrevistas, mítines) o públicos a los que se dirigen. Esto sugiere que adaptan su discurso con mayor flexibilidad según el contexto.

Puntos aislados hacia la izquierda

- Esos pocos discursos que aparecen muy alejados a la izquierda (PC1 muy negativo) son outliers: hablan o enfatizan aspectos muy distintos al "discurso medio" de cada candidato.
- Podrían ser: Discursos centrados en temáticas atípicas (por ejemplo, un discurso especializado en política exterior o en un tema muy concreto). Momentos de tono inusualmente negativo o muy técnico. Charlas de perfil bajo (entrevistas íntimas, podcasts) donde el estilo varía mucho respecto al estilo estándar de los mítines.
- Sería interesante revisar estas muestras "extremas" para entender qué palabras o temas los diferencian, y así interpretar mejor qué aportan PC1 y PC2.

Curva de varianza acumulada (2 – 10 componentes)

El gráfico muestra, en el eje X, el número de componentes principales (de 2 a 10) que incluimos en el análisis, y en el eje Y la varianza explicada acumulada por esas componentes. A medida que añadimos más componentes, aumenta la proporción de variabilidad de los discursos que capturamos, pero con rendimientos decrecientes: los primeros ejes aportan mayores ganancias y los últimos muy pequeñas.

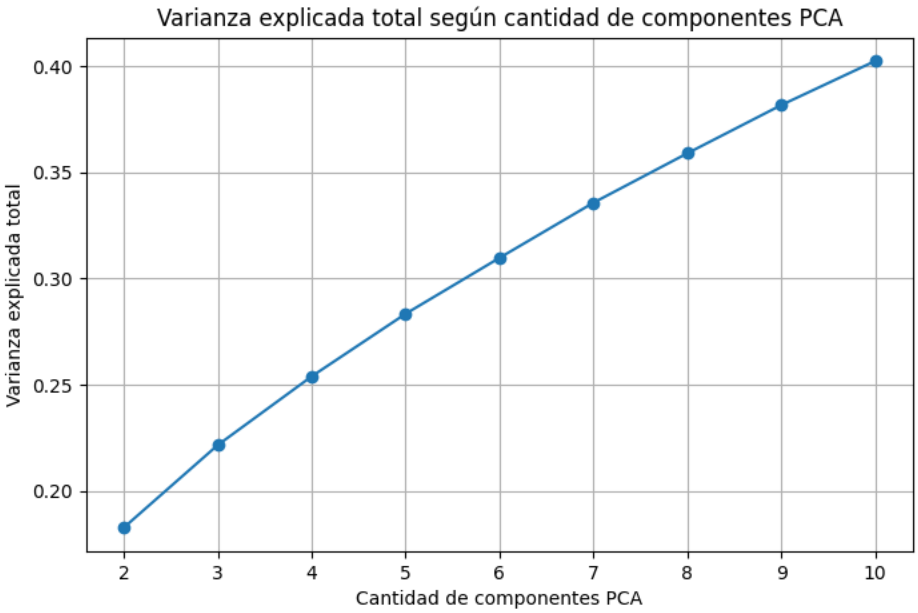


Figura 3: Varianza explicada según cantidad de componentes PCA (hasta 10)

Análisis del gráfico

- Con 2 componentes apenas capturamos ~18 % de la variabilidad de los discursos. Con 5 componentes subimos a ~28 %. Llegando a 10 componentes, alcanzamos ~40 % de la varianza total. Esto ilustra que una proyección en 2D solo recoge una fracción de la información total.
- Ganancias marginales decrecientes: como es de esperarse, cada componente adicional aporta menos "nueva" varianza que la anterior.
- Bajo porcentaje acumulado: Aunque la curva sube de forma continua, un 40 % con 10 componentes sigue siendo relativamente bajo para muchos análisis (se suele buscar al menos 70–80 % para asegurar de que el modelo represente bien los datos). Esto sugiere que el corpus de discursos es muy heterogéneo y se necesita incorporar muchas más componentes para abarcar la mayor parte de la variabilidad.

Elección del número de componentes

A partír de lo anterior surge la pregunta ¿cuántas componentes principales utilizar?, y a continuación, ¿cuántas se necesitarían para explicar el 99% de la varianza?

Este gráfico muestra, de forma acumulada, la varianza explicada por el modelo PCA a medida que vamos incorporando más componentes (desde 1 hasta ~131), y lleva marcada con la línea roja discontinua la frontera del 99 % de varianza explicada.

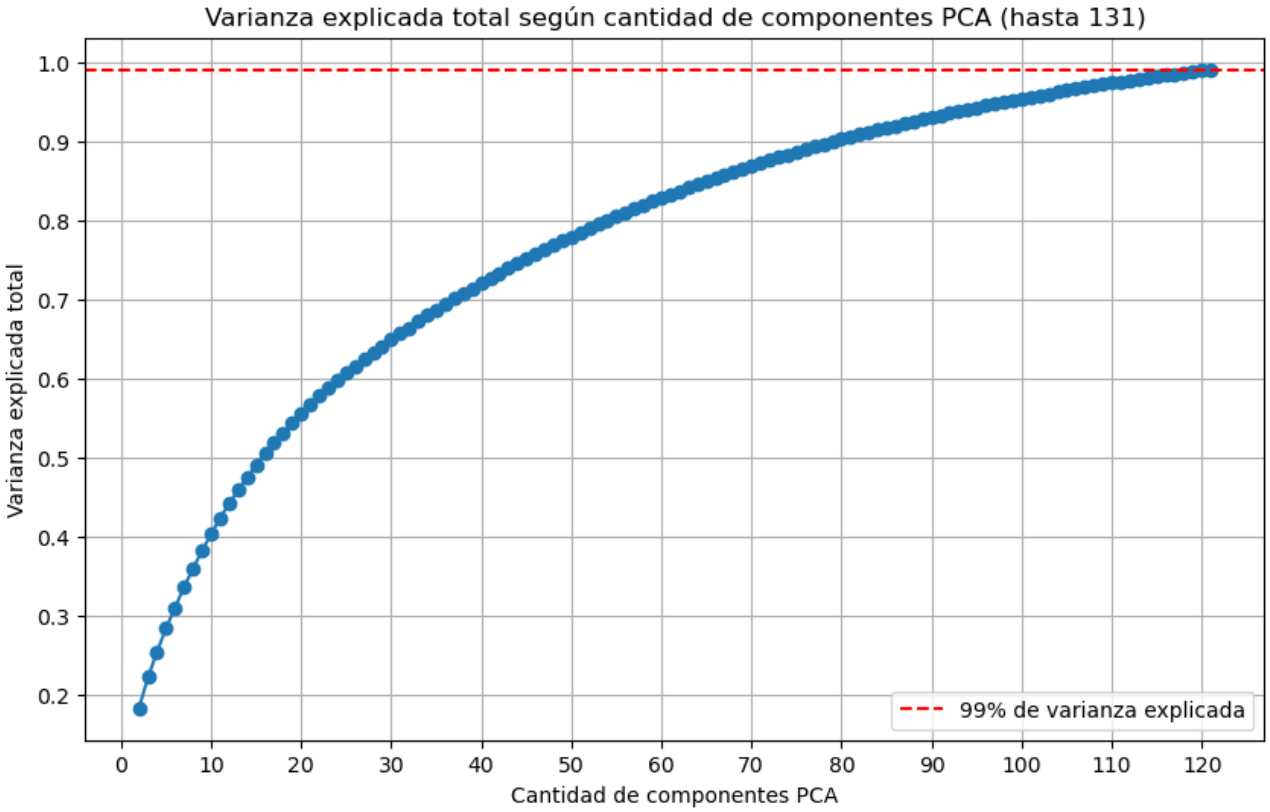


Figura 4: Varianza explicada según cantidad de componentes PCA (hasta 131)

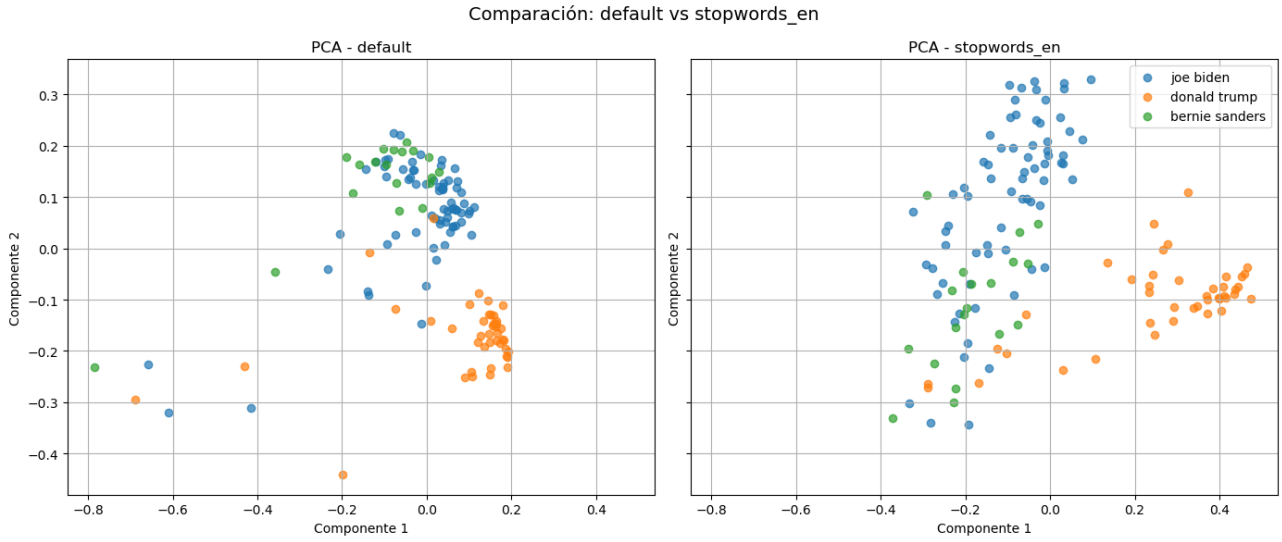
Este gráfico muestra una curva de crecimiento suave donde el porcentaje de varianza explicada crece de forma monótona sin un “codo” muy pronunciado que marque un punto natural de corte; la curva es bastante convexa.
Para alcanzar el umbral del 99 %, el gráfico indica que hacen falta prácticamente todas las componentes (del orden de 120–130 sobre 131), esto revela que el espacio semántico de los discursos es muy alto en dimensión y que cada componente retiene algo de señal; no hay redundancia masiva.
Para ver donde “cortar” hay algunas reglas entre las que se encuentran por ejemplo

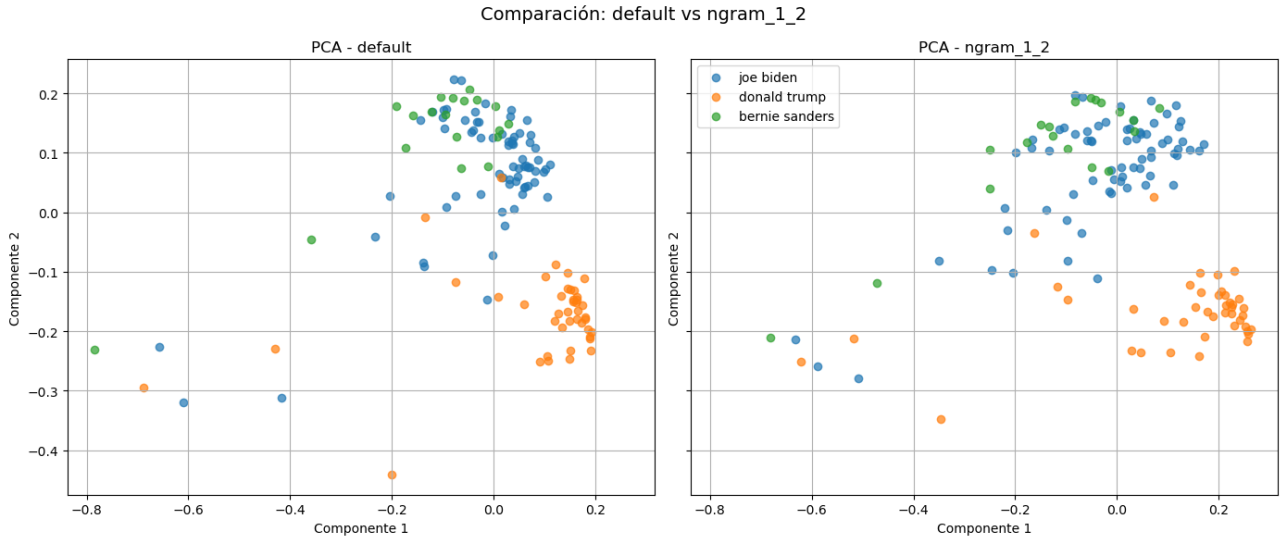
- Método del codo (scree test): busca el punto en el que la pendiente de la curva empieza a ser casi horizontal.
- Kaiser–Guttman: quedarse con aquellas componentes cuya varianza individual (autovalor) sea ≥ 1.

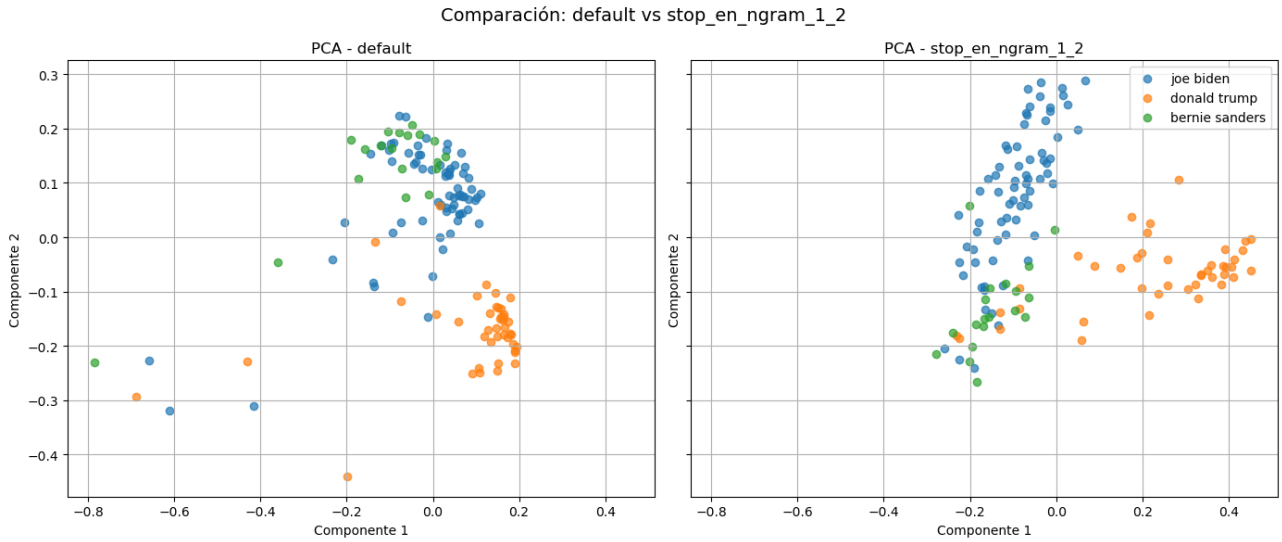
Si deseáramos reconstruir casi toda la varianza, deberíamos quedarnos con casi el conjunto completo de ejes, lo cual prácticamente anula la reducción de dimensión. En general para visualización o interpretación, suele bastar con un umbral más moderado (por ejemplo, 70–80 %) que en nuestro caso se alcanzaría hacia las 40–50 componentes.

Configuraciones comparadas

A continuación se muestran dos subgráficos lado a lado (“default” vs. cada variante) para comparar cómo cambia la separación de clusters.


Comparación: default vs stopwords_en
PCA - default
PCA - stopwords_en
joe biden
donald trump
bernie sanders
Componente 1
Componente 2


Comparación: default vs ngram_1_2
PCA - default
PCA - ngram_1_2
joe biden
donald trump
bernie sanders
Componente 1
Componente 2


Comparación: default vs stop_en_ngram_1_2
PCA - default
PCA - stop_en_ngram_1_2
joe biden
donald trump
bernie sanders
Componente 1
Componente 2

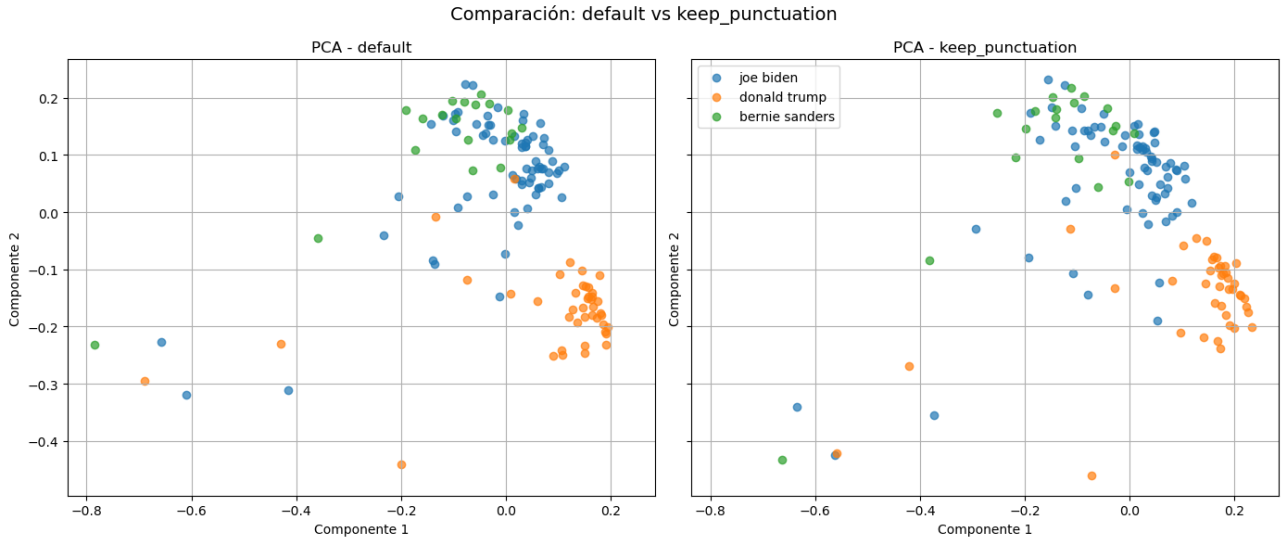


Figura 5: Gráficos comparativos con variantes (para 2 PCA)

Profundizando en los gráficos comparativos, podemos extraer varias lecturas sobre el comportamiento de cada configuración:

Default vs. Stop Words en inglés
Al eliminar las stop words, las nubes de puntos correspondientes a Biden y Sanders se dispersan aún más. Esto puede deberse a que muchas de estas palabras —aunque sean consideradas vacías— están presentes en casi todos los discursos, y actúan como una base léxica común.
Su presencia contribuye a mantener los vectores de los documentos relativamente similares, lo que compacta su proyección. Al eliminarlas, se pierde esa base compartida y las diferencias léxicas más específicas se vuelven dominantes, acentuando la dispersión.
Este comportamiento sugiere que eliminar indiscriminadamente stop words puede no ser la mejor opción en corpus homogéneos o especializados como el político. Podría ser preferible conservar algunas de ellas o trabajar con una lista ajustada al dominio.

Default vs. n-gramas (1,2)
La inclusión de bigramas no modifica sustancialmente la forma de los clusters: Trump sigue aislado, pero Biden y Sanders permanecen solapados. Esto podría explicarse porque, aunque los bigramas capturan combinaciones de palabras (p. ej. “health care”), esos patrones no aportan información discriminativa suficiente en dos dimensiones: su varianza queda repartida en multitud de pares poco frecuentes. Para que los n-gramas se traduzcan en separación clara, podría ser necesario aumentar la dimensionalidad del PCA o seleccionar únicamente los bigramas más frecuentes/significativos.

Default vs. Stop + n-gramas
Aquí se obtiene la mejor compactación: Biden forma un cluster definido, Sanders se acerca pero sin mezclarse en exceso, y Trump se mantiene separado. Esta configuración parecería ser la más prometedora para tareas de clasificación aunque el solapamiento parcial de Sanders sigue señalando la necesidad de más datos o de características adicionales.

Default vs. Mantener puntuación
Incluir signos de puntuación (puntos, comas, etc.) prácticamente no cambia la topología: Sanders incluso se dispersa un poco más. Esto podría significar que los signos de puntuación son en gran medida ruido para este análisis léxico: no contribuyen a la varianza central que define la identidad discursiva de cada candidato. Por lo anterior es preferible descartarlos, pues aumentan el vocabulario total (y por tanto la dimensionalidad) sin ganar poder discriminativo.

En conclusión

- **Separabilidad**: Donald Trump es consistentemente distinguible en cualquiera de las transformaciones, reflejando un estilo vocabular propio.

- **Biden vs. Sanders**: Su solapamiento, incluso en la mejor configuración, sugiere que comparten un núcleo léxico muy similar o que el volumen de datos de Sanders (mucho menor) dificulta capturar su sello particular.

- **Límites del PCA 2D**: Aunque útil para un vistazo rápido, restringirnos a dos componentes puede ocultar separaciones que aparecen en dimensiones superiores. El análisis de varianza explicada muestra que hacen falta muchas más para captar buena parte de la variabilidad.

2 Parte 2: Entrenamiento y Evaluación de Modelos

2.1 PARTE A

Entrene el modelo Multinomial Naive Bayes para clasificar los discursos según a qué candidato o candidata pertenece el texto. A continuación, utilice el modelo para clasificar los discursos del conjunto de test, y reporte el valor de accuracy y la matriz de confusión. Reporte el valor de precision y recall para cada candidato. Explique cómo se relacionan estos valores con la matriz anterior. ¿Qué problemas puede tener el hecho de mirar sólamente el valor de accuracy? Considere qué sucedería con esta métrica si el desbalance de datos fuera aún mayor entre candidatos. Sugerencia: utilice el método from_predictions de ConfusionMatrixDisplay para realizar la matriz.

2.1.1 Marco teórico

Multinomial Naive Bayes (MNB)

El algoritmo **Multinomial Naive Bayes (MNB)** es un algoritmo de clasificación supervisada ampliamente utilizado en problemas de **clasificación de texto**, como análisis de sentimiento, detección de spam o, como en este caso, identificación del autor de un discurso. Es parte de la familia de clasificadores **Naive Bayes**, que se basan en aplicar el **teorema de Bayes** bajo la suposición (naive) de que todas las características (palabras, en este contexto) son **independientes entre sí**.

El algoritmo calcula la probabilidad de que un documento D (representado como un conjunto de palabras -en este caso, un discurso-) pertenezca a una clase C (en este caso, un candidato) como:

$$P(C|D) \propto P(C) \cdot \prod_{i=1}^{n} P(w_i|C)$$

El producto recorre todas las palabras del documento, donde:

- *P(C)* es la probabilidad de la clase (frecuencia de discursos de ese candidato).
- *Wi* es la i-ésima palabra del documento.
- *P(wi/C)* es la probabilidad de que esa palabra aparezca en documentos de la clase C.

El modelo toma en cuenta **la frecuencia de ocurrencia de cada palabra**, lo que lo hace ideal para conteo de palabras como en **bag of words** o **TF-IDF**, es muy rápido de entrenar y predecir, incluso con grandes cantidades de texto) y funciona bien incluso con conjuntos pequeños, si el vocabulario está bien representado.
Por otra parte, el modelo, supone independencia entre palabras, lo cual no es realista (las palabras tienen contexto), es sensible al desbalance de clases (algo que veremos más adelante) y no capta relaciones semánticas o sinónimos entre palabras.

Métricas de desempeño

Para medir **qué tan bien predice MNB las clases (candidatos) correctas** sobre un conjunto de datos que el modelo **nunca vio durante el entrenamiento** (el *conjunto de test*) se utilizan varias **métricas de desempeño**.

Accuracy (Precisión global)
Mide la proporción de predicciones correctas sobre el total de ejemplos. Si de 100 discursos, el modelo clasifica correctamente 90, la accuracy es 0.90 (90%).

$$\text{Accuracy} = \frac{\text{Número de predicciones correctas}}{\text{Total de predicciones}}$$

Como punto negativo, si las clases están desbalanceadas (por ejemplo, 80 discursos de Biden, 10 de Trump y 10 de Sanders), un modelo que siempre diga "Biden" tendrá 80% de accuracy aunque no aprenda nada de las otras clases.

Matriz de Confusión
Se trata de una tabla que compara las **clases reales** con las **clases predichas** por el modelo, permitiendo ver cuántas predicciones fueron correctas y en qué casos se cometieron errores. Cada fila representa una clase real, y cada columna una clase predicha. Idealmente, todos los valores estarían en la diagonal (predicciones correctas), mientras que los valores fuera de la diagonal indican errores de clasificación. Esta matriz permite calcular métricas clave como *precision*, *recall* y *F1-score* para cada clase.

Precision
Mide la **exactitud de las predicciones positivas del modelo**. Es decir, de todos los discursos que el modelo dijo que eran de un candidato, ¿cuántos realmente lo eran?
Ejemplo: si el modelo predijo 10 discursos como de Trump, pero solo 6 lo eran en verdad, la precision es 6/10 = 0.60.

$$\text{Precision}_C = \frac{\text{True Positives}_C}{\text{True Positives}_C + \text{False Positives}_C}$$

Precision responde: ¿cuántos de los que marqué como positivos lo eran en realidad?

Recall (Sensibilidad)
Mide la **capacidad del modelo para encontrar todos los ejemplos positivos reales**. Es decir, de todos los discursos que realmente pertenecen a un candidato, ¿cuántos logró identificar correctamente el modelo?
Ejemplo: si había 10 discursos de Sanders y el modelo detectó correctamente 7, el recall es 7/10 = 0.70.

$$\text{Recall}_C = \frac{\text{True Positives}_C}{\text{True Positives}_C + \text{False Negatives}_C}$$

Recall responde: ¿cuántos positivos reales detecté?

Observación:
Recall se fija en los **casos reales**: ¿cuántos de los que debía detectar, detectó?
Precision se fija en los **casos predichos**: ¿cuántos de los que predijo eran correctos?

F1-Score
¿Qué mide?
Es la media armónica entre precision y recall. Balancea ambas métricas.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Sirve especialmente cuando tenemos un **desequilibrio entre clases** (como en este caso, donde hay muchos más discursos de Biden que de Sanders o Trump).
Ejemplo:

- Si Precisión = 1.0 (todo lo que predice como "Trump" es efectivamente Trump),
- Recall = 0.1 (detecta solo 10% de los discursos reales de Trump).

En este caso, el modelo parecería "preciso", pero **no útil**, porque **se le escapan la mayoría** de los discursos verdaderos. El F1-score penaliza esto y da un valor más realista del rendimiento.

- **F1-score = 1** → modelo perfecto (detecta todo y no se equivoca).
- **F1-score = 0** → no detecta nada útil (o se equivoca siempre).

2.1.2 Metodología

Se construyó un *pipeline* con tres pasos:
CountVectorizer: vectoriza los textos en frecuencias de palabras.
TfidfTransformer: ajusta las frecuencias con el peso TF-IDF.
MultinomialNB: entrena el modelo de Naive Bayes sobre los vectores generados.
El modelo fue entrenado sobre el conjunto de desarrollo (X_dev, y_dev) y luego se predijo el conjunto de prueba (X_test). Se calcularon todas las métricas solicitadas y se visualizó la matriz de confusión usando:

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
print(classification_report(y_test, y_pred, digits=3))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred).plot()
```

2.1.3 Resultados

Accuracy: 0.5439 → El modelo acertó en el 54% de los casos del conjunto de test.
F1-Score: 0.3832 → Hay desequilibrio entre precisión y recall, indicando un modelo poco confiable aún.

Matriz de confusión:

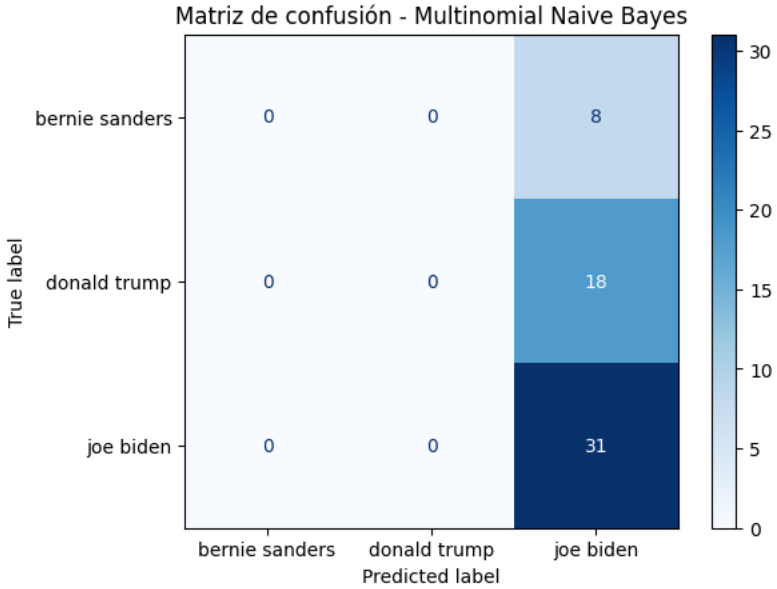


Figura 6: Matriz de confusión – Multinomial Naive Bayes

Precisión, Recall, F1-score:
Bernie Sanders y Donald Trump: todas las métricas son 0.000.
Joe Biden: precisión = 0.544, recall = 1.0, F1 = 0.705.

El desempeño del modelo fue insatisfactorio: **predice exclusivamente "Joe Biden"** para todos los discursos del test, sin importar su clase real. Esto lo vemos porque **todas las predicciones caen en la última columna de la matriz de confusión**. Como consecuencia el **recall de Biden es 1.0**, porque detectó correctamente todos los discursos de Biden pero **el modelo no acierta ni uno solo de Trump ni Sanders**, por lo tanto sus métricas son 0.

Este resultado es típico cuando el conjunto de entrenamiento tiene más ejemplos de una clase (en este caso, Biden). El modelo se “acomoda” prediciendo siempre la clase más frecuente. Aunque 54% parece aceptable, **la accuracy es artificialmente alta** porque refleja que simplemente el modelo acertó en la clase más numerosa.

El modelo **no aprendió a distinguir entre los candidatos**. Este resultado **evidencia la necesidad de abordar el desbalance de clases**, por ejemplo, con técnicas de muestreo (submuestreo o sobremuestreo). También puede ser útil explorar modelos más complejos o ajustar los hiperparámetros del vectorizador o del clasificador como se realizará más adelante.

2.2 PARTE B

Explique cómo funciona la técnica de validación cruzada o cross-validation. Implemente una búsqueda de hiperparámetros usando GridSearchCV. Genere una visualización que permita comparar las métricas (e.g: accuracy) de los distintos modelos entrenados, viendo el valor promedio y variabilidad de las mismas en todos los splits (e.g: en un gráfico de violín).

2.2.1 Marco Teórico

La validación cruzada k-fold

La **validación cruzada** es una técnica para evaluar modelos de forma más robusta. Para esto se divide el conjunto de entrenamiento en **k** partes (llamadas *folds*), aproximadamente del mismo tamaño.

Luego, se realizan **k iteraciones**: en cada iteración, **se toma una de esas partes como conjunto de validación**, las otras **k - 1 partes se usan para entrenar el modelo**.

En lugar de entrenar y testear una sola vez, se divide el conjunto de entrenamiento en varios **folds** o particiones (por ejemplo, 5).

En cada iteración:

- Se entrena el modelo con 4 folds.
- Se valida (evalúa) en el fold restante.
  Esto se repite 5 veces, cambiando el fold de validación

Al final, se promedian las métricas (como *accuracy o F1-score*) de esas k pruebas, obteniendo una evaluación más **robusta y confiable** del modelo.
Por ejemplo, si usamos $k = 5$, el modelo se entrena 5 veces con distintos conjuntos de datos, y cada parte es usada una vez como validación.
Esta técnica se usa porque reduce el riesgo de que los resultados dependan demasiado de cómo se dividieron los datos**.** Es especialmente útil cuando el dataset no es muy grande.

GridSearchCV

Es una herramienta de *scikit-learn* que automatiza dos tareas clave:

- **Buscar la mejor combinación de hiperparámetros para un modelo.**
- **Evaluar cada combinación usando validación cruzada.**

Se le da un modelo (por ejemplo, un Pipeline con CountVectorizer + TfidfTransformer + MultinomialNB) y una "grilla" (grid) de hiperparámetros: una lista de combinaciones que se desea probar. Por cada combinación de esa grilla, GridSearchCV entrena y valida el modelo **usando validación cruzada (cross-validation)**. Guarda la métrica promedio (por ejemplo, accuracy o F1-score) de cada combinación. Al final, dice **cuál fue la mejor combinación** y deja el modelo ya entrenado con esos parámetros óptimos. En particular, se utilizó el F1-score macro como métrica de evaluación, que promedia por clase sin ponderar por frecuencia.

Hiperparámetros
Los **hiperparámetros** son **parámetros de configuración del modelo** (y del preprocesamiento) que **no se aprenden automáticamente**. Hay que definirlos antes del entrenamiento. Distinto de los **parámetros del modelo**, que sí se aprenden a partir de los datos.
En este contexto, se usa un pipeline con tres etapas: **CountVectorizer**, **TfidfTransformer**, **MultinomialNB**, cada una tiene hiperparámetros que se pueden ajustar, estos son:

vect__ngram_range
Pertenece a: CountVectorizer
Qué hace: Define si se usan solo palabras individuales (**unigramas**, ngram_range=(1,1)) o también combinaciones de dos palabras seguidas (**bigramas**, ngram_range=(1,2)).
Ejemplo: Para "United States": Unigramas → "united", "states": Bigramas → "united states"

tfidf__use_idf
Pertenece a: TfidfTransformer
Qué hace: Activa o desactiva el uso de **IDF** (inverse document frequency), que penaliza las palabras comunes en todo el corpus. True: aplica la fórmula TF-IDF completa. False: solo cuenta las veces que una palabra aparece en el documento (TF).

clf__alpha
Pertenece a: MultinomialNB
Qué hace: Es el parámetro de **suavizado de Laplace**.
alpha=1.0 → suavizado fuerte (predicciones más conservadoras)
alpha=0.01 → suavizado débil (modelo más flexible pero más propenso al overfitting)
Previene que una palabra con cero frecuencia en una clase haga que la probabilidad de esa clase sea cero.

Se usa una grilla porque no se sabe de antemano cuál combinación de parámetros es mejor. Por ejemplo: quizás usar bigramas no mejora nada, quizás el IDF penaliza demasiado y es mejor no usarlo, quizás un alpha bajo da mejor recall, pero peor precision.
Entonces, GridSearchCV prueba todas las combinaciones que se le pasan y devuelve qué combinación funcionó mejor y cuál fue el rendimiento promedio de cada una.

2.2.2 Metodología

Se definió un **pipeline de procesamiento** compuesto por tres etapas:
- CountVectorizer: transforma el texto en vectores numéricos (bag-of-words).
- TfidfTransformer: aplica el escalado TF-IDF.
- MultinomialNB: modelo de clasificación Naive Bayes para texto.

A continuación, se definió una **grilla de hiperparámetros** a explorar:
- vect__ngram_range: se probaron unigramas (1,1) y unigramas + bigramas (1,2) para capturar más contexto lingüístico.
- tfidf__use_idf: se probó con y sin penalización por frecuencia global (IDF).
- clf__alpha: se evaluaron valores de suavizado de Laplace (1.0, 0.1, 0.01), lo cual regula cuánto se suavizan las probabilidades de palabras raras.

Se ejecutó GridSearchCV con validación cruzada de 5 folds (cv=5)

Extracción de resultados
Accesamos gs_clf.cv_results_ para obtener, para cada combinación, los F1-score obtenidos en cada uno de los 5 folds (split0_test_score, …, split4_test_score).
Construimos un DataFrame "largo" donde cada fila es un único fold de una combinación concreta, con columnas:

- split_id (p.ej. "split 9" para la novena combinación)
- F1-score (valor de ese fold)

Visualización

- **Violin plot** de F1-score por split_id, mostrando la distribución de los 5 folds para cada combinación.

Se superpusieron tres elementos de referencia:

- **Promedio por combinación** (puntos rojos pequeños).
- **Mejor combinación** (estrella roja en el promedio más alto).
- **Línea punteada** con el mejor score global (≈ 0.946).

Debajo, una tabla relaciona cada split_id con su conjunto de hiperparámetros.

2.2.3 Resultados

La búsqueda devolvió la **mejor combinación de hiperparámetros** evaluada según la métrica de *F1-Score Promedio* en los cinco folds de validación, los mejores hiperparámetros encontrados fueron:

- **clf__alpha**: 0.01,
- **tfidf__use_idf**: True,
- **vect__ngram_range**: (1, 1):
- **Mejor score (cross-validation)**: 0.9139

Esto indica que el modelo logra su mejor rendimiento promedio cuando:

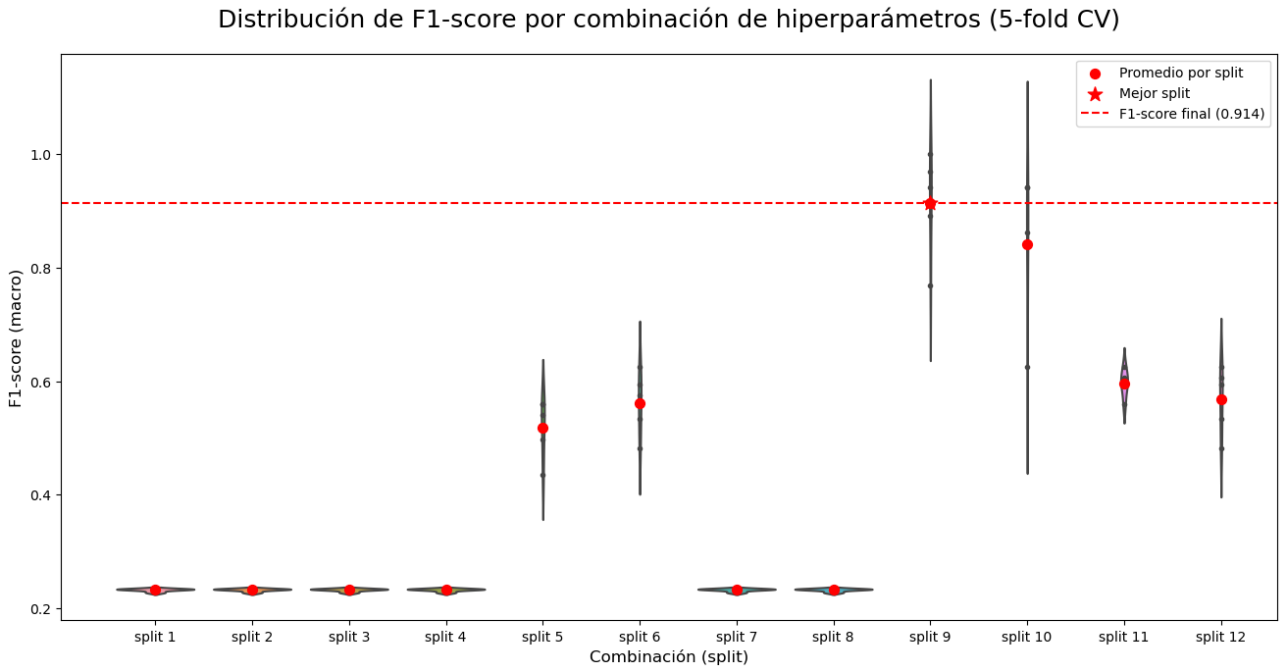
clf__alpha: 0.01: Esto indica un **suavizado débil** en el modelo Naive Bayes. Un alpha bajo permite que el modelo aprenda más directamente de los datos sin suavizar demasiado las probabilidades.

tfidf__use_idf: True: Significa que **sí se está usando IDF** en el cálculo de TF-IDF. Esto penaliza las palabras que aparecen en muchos documentos y da más peso a las que son más distintivas para cada documento/clase. En este caso, usar IDF mejoró el rendimiento del modelo.

vect__ngram_range: (1, 1). Solo se están usando **unigramas** (palabras individuales). Aunque se probaron bigramas también, en este caso, el modelo funcionó mejor **sin bigramas**. Esto puede pasar si los bigramas no aportan tanta información adicional o si el corpus no es muy grande.

Como se mencionó, también se generó una visualización que muestra la distribución del **F1-score macro** obtenido en los 5 folds de validación cruzada para cada combinación de hiperparámetros evaluada. En esta representación, **cada "split" corresponde a una combinación distinta** de hiperparámetros explorada por GridSearchCV.
La gráfica incluye:

- La **distribución de los valores de F1-score macro** por combinación, representada mediante un **gráfico de violín**.
- Los **valores individuales de F1-score macro** obtenidos en cada fold (marcados como **puntos negros pequeños**).
- El **promedio de F1-score macro** por combinación (indicado con un **punto rojo grande**).
- La **mejor combinación de hiperparámetros** según el promedio de F1-score macro (destacada con una **estrella roja grande**).
- Una **línea horizontal roja punteada** que marca el valor del mejor f1-score promedio global alcanzado (GridSearchCV.best_score_ = 0.914).



Split	Combinación de hiperparámetros
split 1	{'clf__alpha': 1.0, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 1)}
split 2	{'clf__alpha': 1.0, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
split 3	{'clf__alpha': 1.0, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 1)}
split 4	{'clf__alpha': 1.0, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 2)}
split 5	{'clf__alpha': 0.1, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 1)}
split 6	{'clf__alpha': 0.1, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
split 7	{'clf__alpha': 0.1, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 1)}
split 8	{'clf__alpha': 0.1, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 2)}
split 9	{'clf__alpha': 0.01, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 1)}
split 10	{'clf__alpha': 0.01, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
split 11	{'clf__alpha': 0.01, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 1)}
split 12	{'clf__alpha': 0.01, 'tfidf__use_idf': False, 'vect__ngram_range': (1, 2)}

Figura 7: Distribución de F1-Score por combinación de hiperparámetros (5-fold CV)

Combinaciones débiles (`α=1.0` o `use_idf=False`) mostraron F1-score macro muy baja y sin variabilidad (todos los folds rondando ~0.23), indicando sobre-suavizado o falta de TF-IDF.

Configuraciones intermedias (`α=0.1`, `use_idf=True`) mejoraron hasta ~0.56 promedio, pero con alta dispersión entre folds.

Mejor configuración: `vect__ngram_range=(1,1)` (solo unigramas), `tfidf__use_idf=True`, `clf__alpha=0.01` obtuvo un **f1-score medio ≈ 0.914** con menor dispersión entre folds.

El gráfico de violín confirma que esta combinación no solo alcanza el valor medio más alto, sino que sus puntuaciones en los 5 folds son consistentemente elevadas.

En conjunto, este análisis demuestra cómo la búsqueda sistemática y la visualización de la dispersión de las métricas en cada partición permiten seleccionar hiperparámetros robustos y estables bajo validación cruzada.

2.3 PARTE C

Elija el mejor modelo (mejores parámetros) y vuelva a entrenar sobre todo el conjunto de entrenamiento disponible (sin quitar datos para validación). Reporte el valor final de las métricas y la matriz de confusión. Discuta las limitaciones de utilizar un modelo basado en bag-of-words o tf-idf en cuanto al análisis de texto.

2.3.1 Metodologia

Se tomó el modelo con los **mejores hiperparámetros encontrados** mediante GridSearchCV: clf__alpha: 0.01, tfidf__use_idf: True y vect__ngram_range: (1, 1)
Se construyó un nuevo pipeline con esos valores y se entrenó sobre **todo el conjunto de entrenamiento (X_dev, y_dev)** para no dejar fuera información útil.
El modelo final se aplicó al conjunto de test (X_test), y se evaluó mediante:

- classification_report de scikit-learn para obtener precisión, recall y F1 por clase.
- accuracy_score para obtener el valor global de exactitud.
- ConfusionMatrixDisplay para visualizar la matriz de confusión.

2.3.2 Resultado

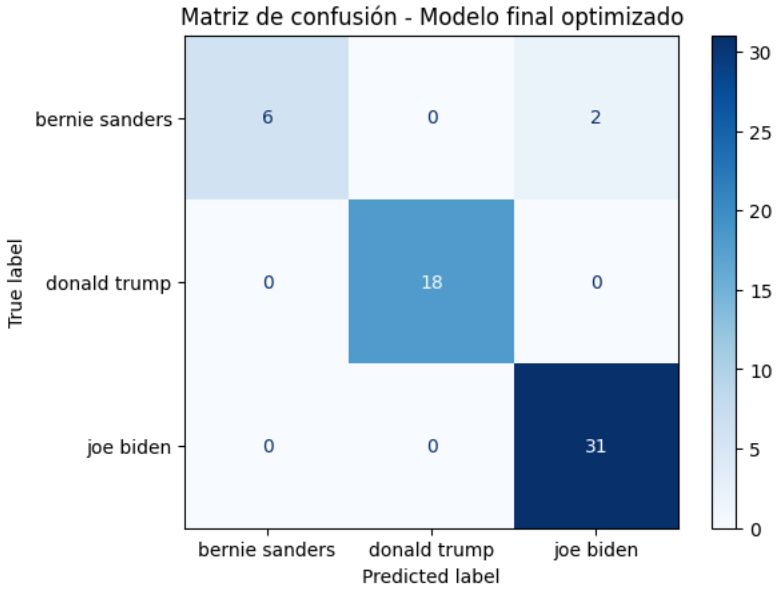


Figura 8: Matriz de confusión – Modelo Final Optimizado

El modelo alcanzó un **accuracy global de 96.49% y valor de F1-Score macro de 94.20,** lo que refleja una excelente capacidad predictiva general, equilibrando precisión y cobertura entre las clases. A nivel de clases:

- **Donald Trump** y **Joe Biden** fueron clasificados correctamente en todos o casi todos los casos: Trump: F1 = 1.000 (clasificación perfecta). Biden: F1-score = 0.969, con recall de 1.000.
- La clase **Bernie Sanders** tuvo un menor recall (0.750), lo que indica que el modelo confundió algunos ejemplos de esta clase con otras, posiblemente debido a que tiene **menos datos de entrenamiento**.

Limitaciones del Enfoque Bag-of-Words / TF-IDF

Aunque el modelo logró un excelente desempeño, es importante tener en cuenta las limitaciones de los métodos de representación utilizados:

1. **Bag-of-Words y TF-IDF no capturan el orden ni la semántica** de las palabras. Por ejemplo, las frases “no apoyo a Trump” y “apoyo a Trump” pueden parecer similares.
2. Son **sensibles a ruido y sinónimos**: si el mismo concepto se expresa con palabras distintas, el modelo puede no detectarlo.
3. **No tienen contexto**: cada palabra es tratada como independiente, lo cual limita el análisis de frases complejas o irónicas.

Por estas razones, en tareas más complejas o de mayor escala, suelen emplearse modelos basados en **representaciones semánticas más profundas** como word embeddings (Word2Vec, GloVe) o modelos preentrenados como BERT.

2.4 PARTE D

Evalúe al menos un modelo más (dentro de scikit-learn) aparte de Multinomial Naive Bayes para clasificar el texto utilizando las mismas features de texto. Explique brevemente cómo funciona y compare los resultados con el anterior.

2.4.1 Marco Teórico

En esta etapa se solicita evaluar al menos un modelo adicional al Multinomial Naive Bayes, utilizando las mismas representaciones de texto (bag-of-words y TF-IDF). El propósito es comparar el desempeño entre diferentes enfoques de clasificación y observar cómo responden ante el mismo conjunto de características.

Se eligió un modelo de clasificación basado en máquinas de vectores de soporte (SVM), específicamente implementado mediante SGDClassifier de *scikit-learn*, que permite aproximar el entrenamiento de un SVM lineal mediante descenso de gradiente estocástico. Esta técnica es especialmente útil para conjuntos de datos grandes y dispersos, como aquellos derivados de texto vectorizado.

Support Vector Machine (SVM)

Es un algoritmo de aprendizaje supervisado muy utilizado en clasificación de textos, imágenes y más. Su objetivo principal es **encontrar una frontera (hiperplano)** que **separe de forma óptima** las clases en los datos. En un problema de clasificación binaria, esa frontera separa dos grupos (por ejemplo, discursos de Trump vs. discursos de Biden).

SVM busca **la línea recta (en 2D)** o **el hiperplano (en dimensiones mayores)** que **divide las clases con el mayor margen posible**. Ese margen es la distancia entre la frontera y los puntos de entrenamiento más cercanos de cada clase. Estos puntos se llaman **vectores de soporte**. SVM no solo separa las clases, sino que **elige la separación más “segura”** (más alejada de los datos límite).

Los textos, convertidos en vectores TF-IDF, suelen tener **miles de dimensiones** (una por palabra). SVM **funciona muy bien en espacios de alta dimensión**. Es eficaz incluso cuando el número de características es mucho mayor que la cantidad de ejemplos (como en este caso). Suele ser más robusto que Naive Bayes cuando las clases están desbalanceadas o cuando hay ambigüedad en los datos.

`SGDClassifier`

`SGDClassifier` es un clasificador de **scikit-learn** que entrena modelos lineales (como regresión logística o SVM lineal) utilizando **Stochastic Gradient Descent** (descenso de gradiente estocástico). Es **rápido, eficiente y escalable**, especialmente para datasets grandes y dispersos como los de texto.

Busca minimizar una función de pérdida (por ejemplo, el error entre las predicciones y los valores reales). Lo hace **actualizando los pesos del modelo** de manera progresiva, usando un pequeño grupo de ejemplos a la vez (en lugar de todo el conjunto). Es "**estocástico**" porque introduce aleatoriedad al elegir los ejemplos usados en cada paso. En lugar de calcular el error global, **ajusta el modelo a medida que ve más datos**, lo cual lo hace más liviano para grandes volúmenes de texto.

Puede entrenar distintos modelos dependiendo de la **función de pérdida** (`loss`) que se le indique:
`loss='hinge'` → entrena un **SVM lineal**.
`loss='log_loss'` → entrena una **regresión logística** (útil para clasificación).
`loss='modified_huber'`, `squared_hinge`, etc. → variantes con diferente sensibilidad a errores.

Hiperparámetros utilizados:

Parámetro	Qué hace
`loss`	Define el tipo de modelo. Por ejemplo: `'hinge'` para SVM, `'log_loss'` para regresión logística.
`penalty`	Tipo de regularización: `'l2'` (reduce pesos grandes), `'l1'` (sparse), `'elasticnet'` (combinado).
`alpha`	Controla la **fuerza de la regularización**. Valores bajos = menos regularización (riesgo de overfitting).
`max_iter`	Número máximo de iteraciones (épocas) sobre el conjunto de entrenamiento.
`tol`	Tolerancia para detener el entrenamiento si la mejora es pequeña.
`random_state`	Semilla para reproducibilidad.

2.4.2 Metodología

Se construyó un *pipeline* de procesamiento idéntico al utilizado anteriormente, con tres etapas:

- `CountVectorizer` para transformar el texto en una matriz de conteo de n-gramas.
- `TfidfTransformer` para aplicar la ponderación TF-IDF.
- `SGDClassifier` como clasificador lineal.

A continuación, se realizó una búsqueda de hiperparámetros mediante `GridSearchCV`, evaluando múltiples combinaciones de:

- Rango de n-gramas (unigramas y bigramas),
- Uso de IDF en la ponderación,
- Parámetro de regularización (`alpha`),

- Tipo de penalización (`l1`, `l2`),
- Función de pérdida (`hinge` para SVM o `log_loss` para regresión logística).

La validación cruzada (`cv=5`) permitió comparar las combinaciones sobre particiones independientes del conjunto de entrenamiento, mejorando la robustez de la selección.

2.4.3 Resultados

La mejor configuración hallada fue:

- **`clf__alpha:`** `0.001`
- **`clf__penalty:`** `l1`
- **`clf__loss:`** `hinge`
- **`tfidf__use_idf:`** `True`
- **`vect__ngram_range:`** `(1, 2)`

El mejor desempeño promedio en validación cruzada fue un F1-score macro de **0.9742**.

Una vez entrenado el modelo óptimo sobre todo el conjunto de desarrollo, se evaluó sobre el conjunto de test, obteniendo: **Accuracy final: 0.9649** y F1-Score macro en test (SVM optimizado): 0.9540.

Desempeño por clase:

Bernie Sanders:	Precision = 1.00,	Recall = 0.88,	F1-score = 0.93
Donald Trump:	Precision = 0.94,	Recall = 0.94,	F1-score = 0.94
Joe Biden:	Precision = 0.97,	Recall = 1.00,	F1-score = 0.98

La **matriz de confusion** en este caso es:

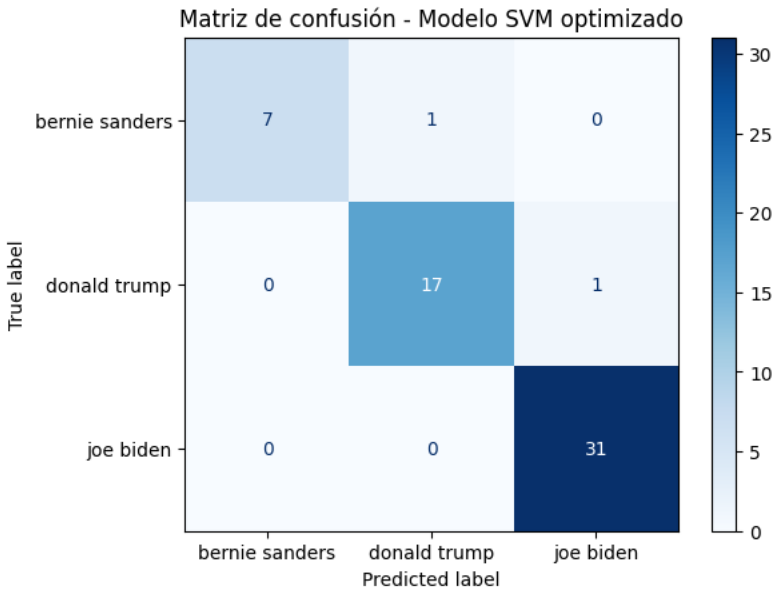


Figura 9: Matriz de confusión – Modelo SVM optimizado

El modelo SVM superó al Multinomial Naive Bayes en casi todos los indicadores, especialmente en la capacidad de distinguir entre clases minoritarias como Sanders. Esto muestra que, con una

representación TF-IDF y una correcta sintonización de parámetros, los modelos lineales como SVM pueden ofrecer mejor generalización y balancear el rendimiento entre clases, incluso ante datos moderadamente desbalanceados.

2.5 PARTE E

Evalúe el problema cambiando al menos un candidato. En particular, observe el (des)balance de datos y los problemas que pueda generar, así como cualquier indicio que pueda ver en el mapeo previo con PCA. Puede ser útil comentar acerca de técnicas como sobre-muestreo y submuestreo, no es necesario implementarlo.

2.5.1 Marco Teórico

A continuación se busca reconfigurar el problema de clasificación de discursos sustituyendo a Bernie Sanders por Mike Pence, luego:

- **Se observa el nivel de desbalance** de ejemplos entre clases y sus posibles efectos en el modelo.
- **Se analiza indicios** de separación entre las nuevas clases en el espacio reducido por PCA.
- **Se discuten** técnicas como sobremuestreo o submuestreo que podrían mitigar el desbalance (sin necesidad de implementarlas).

Desde el punto de vista teórico, cuando una clase tiene muchos menos ejemplos, los algoritmos de aprendizaje tienden a verse “arrastrados” por las clases mayoritarias, lo que puede producir sesgos en las predicciones (por ejemplo, ignorar casi por completo la clase minoritaria).

2.5.2 Metodología

Reconfiguración del dataset

Se construyó un nuevo DataFrame con las transcripciones de Joe Biden, Donald Trump y Mike Pence.
Se “derritió” (melt) el formato ancho a largo para tener una fila por discurso–candidato.
Se filtraron las filas vacías y se aplicó la función `clean_text()` para normalizar los textos.

Partición estratificada

Se dividieron los datos en 70 % “dev” (entrenamiento) y 30 % test, manteniendo la proporción de discursos por candidato (estratificación).

Verificación del balance

Se contó cuántos discursos hay de cada candidato en dev y en test, comprobando un ligero desbalance:
Dev → Biden 70, Trump 42, Pence 16
Test → Biden 31, Trump 18, Pence 7

Representación de texto y PCA

Se vectorizó con Bag-of-Words y TF-IDF.
Se aplicó PCA a la matriz TF-IDF para reducir a 2 dimensiones y graficar cada punto (discurso) coloreado por candidato.

Entrenamiento de un segundo modelo (SVM)

Se usó un pipeline con `CountVectorizer`, `TfidfTransformer` y `SGDClassifier` (SVM lineal).
Se optimizaron hiperparámetros vía `GridSearchCV` (pérdida, penalización, alfa, rango de n-gramas, uso de IDF).

Se reportó la precisión en test y la matriz de confusión.

2.5.3 Resultados

Mapa de discursos con 2 PCA

A continuación se muestra el gráfico PCA=2 de los discursos políticos. Cada punto representa un discurso individual de uno de los tres candidatos analizados (Joe Biden -en azul-, Donald Trump -en naranja- y Mike Pence-en verde-). El eje X (Componente Principal 1) captura la dirección de máxima varianza en el conjunto de datos y El eje Y (Componente Principal 2), muestra la segunda mayor fuente de variación ortogonal a la primera.

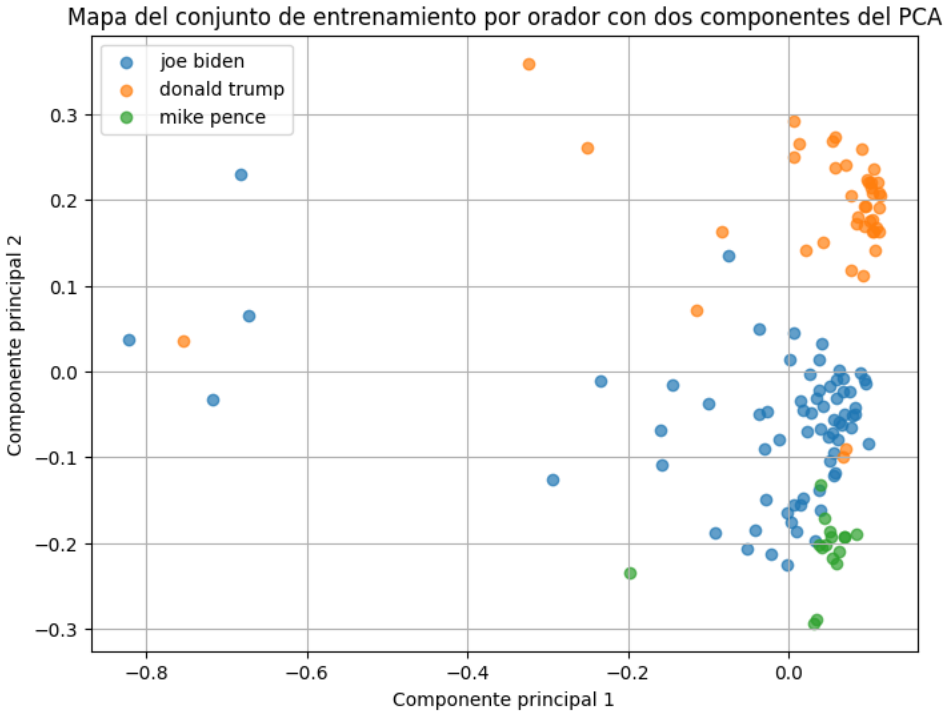


Figura 10: Mapa del conjunto de entrenamiento por orador con dos componentes del PCA

Donald Trump: Nuevamente sus discursos están muy agrupados en la parte superior derecha del gráfico. La separación entre Trump y los otros oradores es muy clara tanto en este gráfico como en el del análisis con Sanders, lo que refuerza la idea de que Trump tiene un estilo discursivo marcadamente diferente.

Joe Biden: Su nube de puntos es más dispersa y extendida, especialmente hacia el eje X negativo. Esto indica mayor variabilidad en su estilo discursivo, temas o estructuras utilizadas.

Mike Pence: Sus discursos están agrupados en un espacio cercano al grupo de Biden, pero más acotados en el eje vertical. A diferencia de Sanders sus discursos no están tan claramente solapados con los de Biden, y son más acotados, no tan dispersos. A través de lo anterior concluimos que:

- Mike Pence parece más limitado y homogéneo en estilo que Sanders, con un posicionamiento más definido y estrecho.
- Sanders y Biden comparten más territorio discursivo que Biden y Pence, aunque Sanders muestra mayor dispersión que Pence.

Métricas de desempeño

Se aplicó SVM optimizando sus párametros de forma análoga a lo explicado en las secciones anteriores y se obtuvo lo siguiente:

SVM optimizado (test):
Accuracy en test (SVM optimizado): 0.9107
F1-Score macro en test (SVM optimizado): 0.8778
Donald Trump: precision 1.00, recall 0.89 → muy preciso pero pierde algunos discursos suyos.
Joe Biden: precision 0.88, recall 0.97 → predice Biden en exceso, con pocos falsos negativos.
Mike Pence: precision 0.83, recall 0.71 → la clase más débil: confusiones frecuentes con Biden.

Este modelo SVM optimizado, si bien sólido, **obtuvo un desempeño inferior al modelo anterior con Bernie Sanders** (f1_macro = 0.8778 vs. 0.9540), lo que evidencia el impacto del cambio de clase minoritaria.
La escasa representación de Mike Pence y su similitud léxica con Biden afectaron tanto su recall como su F1-score.

Matriz de Confusión

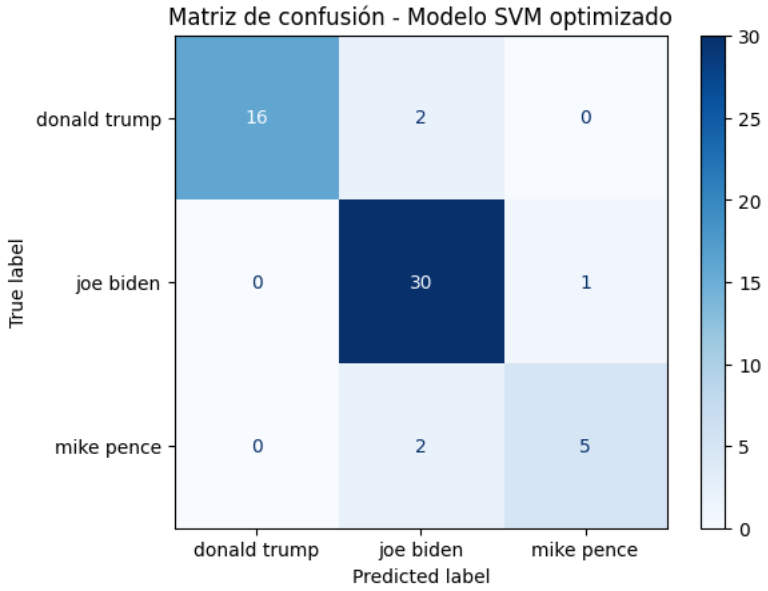


Figura 11: Matriz de confusión: Modelo SVM optimizado

Como ya se dijo Pence posee pocos discursos (16 en dev), lo que le da una representación débil frente a Biden y Trump. Por otro lado, Biden y Pence aparecen más próximos entre sí, con cierta superposición, lo que explica la dificultad para distinguirlos en el modelo.

Si bien Trump se separa claramente del resto, formando un cluster bien diferenciado, es interesante notar que dos discursos de Trump fueron predecidos como de Biden, esto puede verse también claramente en el mapa de los 2PCA, donde se observan dos puntos naranjas en medio de la nube azul de Biden.

Si se decidiera continuar entrenando con este dataset, sería altamente recomendable aplicar técnicas de sobremuestreo sobre la clase de Mike Pence para mejorar particularmente su recall y equilibrar la capacidad del modelo de detectar correctamente discursos de todas las clases.
A continuación, se describen brevemente las principales estrategias utilizadas para abordar este tipo de problemáticas en datasets desbalanceados.

Sobre-muestreo y submuestreo

Cuando trabajamos con conjuntos desbalanceados —como en nuestro caso, donde uno de los oradores (por ejemplo, Mike Pence o Bernie Sanders) tiene muchos menos ejemplos que Joe Biden o Donald Trump—, los modelos tienden a sesgar sus predicciones hacia las clases mayoritarias. Para mitigar este efecto existen dos familias de técnicas de remuestreo:

Sobremuestreo (Oversampling)
Consiste en **aumentar el número de ejemplos de la clase minoritaria** hasta aproximarlo al de la mayoritaria, sin tocar (o tocando lo mínimo) los ejemplos de las clases abundantes.
Esto se puede lograr con:

- **Duplicación simple,** replicando instancias existentes.
- **SMOTE (Synthetic Minority Over-sampling Technique):** generar ejemplos "sintéticos" interpolando entre observaciones minoritarias cercanas en el espacio de features.

Ventajas:

- Permite al modelo "ver" suficientes ejemplos de la clase rara, reduciendo el sesgo.
- SMOTE introduce variabilidad y puede mejorar la capacidad de generalización.

Desventajas:

- Si duplicamos instancias exactas, podemos **sobreajustar** (el modelo memoriza esas réplicas).
- SMOTE puede generar ejemplos irreales si la distribución de la clase minoritaria es muy heterogénea.

Submuestreo (Undersampling)
Consiste en **reducir el número de ejemplos de la(s) clase(s) mayoritaria(s)** para igualar el tamaño de la minoritaria.
Esto se puede lograr con:

- **Selección aleatoria:** eliminar al azar ejemplos de la clase mayoritaria.
- **Cluster-Centroids:** agrupar instancias mayoritarias y reemplazarlas por sus centroides (una forma de "resumir" la mayoría).

Ventajas:

- Evita el sobreajuste a la clase mayoritaria.
- Reduce el tamaño del dataset y, por tanto, los tiempos de entrenamiento.

Desventajas:

- Puede **perder información** valiosa si borramos ejemplos que aportan diversidad al perfil de la clase mayoritaria.
- Si la clase mayoritaria tiene sub-patrones importantes, el submuestreo azaroso puede eliminar justo esos patrones.

Implementar esas técnicas —o una combinación de ambas— suele mejorar significativamente métricas como precision y recall en las clases menos representadas, sin sacrificar demasiado la performance global (accuracy). Por eso, incluso cuando no las usemos directamente en esta entrega, entenderlas y considerarlas es clave para un análisis sólido de problemas con clases desbalanceadas.

2.6 PARTE F

Busque información sobre al menos una técnica alternativa de extraer features de texto. Explique brevemente cómo funciona y qué tipo de diferencias esperaría en los resultados. No se espera que implemente nada en esta parte.

A continuación se propone una alternativa al enfoque Bag-of-Words / TF-IDF, basada en **representaciones densas de palabras** (word embeddings), junto con una breve explicación de su funcionamiento y sus posibles ventajas e impactos en los resultados.

Word Embeddings (p. ej. Word2Vec, GloVe)

En lugar de contar ocurrencias aisladas de cada término (bag-of-words), los **word embeddings** generan representaciones continuas que capturan la similitud semántica entre palabras.

Entrenamiento a partir de contexto

Modelos como Word2Vec o GloVe aprenden vectores de alta dimensión (por ejemplo, 100–300 coordenadas) para cada palabra, de modo que aquellas que comparten contexto aparecen «cerca» en el espacio vectorial.

Representación de documentos

Un discurso completo puede resumirse mediante el promedio (o ponderación) de los vectores de sus palabras, obteniéndose un único punto en el mismo espacio.

Ventajas

- **Captura relaciones semánticas:** sinónimos o términos relacionados («economy» vs. «finance») quedan próximos, mejorando la capacidad de un clasificador para agrupar contenidos afines.
- **Generalización:** permite reconocer similitudes léxicas aun cuando no haya coincidencia exacta de palabras.

Salida

Cada palabra queda asociada a un vector real, de modo que la similitud en el espacio vectorial refleja proximidad semántica.

Representación de documentos

- **Promedio de vectores:** promediar los embeddings de todas las palabras de un discurso.
- **TF-IDF ponderado:** usar como pesos las puntuaciones TF-IDF al promediar.
- **Modelos más sofisticados:** doc2vec, Transformers (BERT, RoBERTa) que generan embeddings a nivel de frase/documento.

Resultado

Mejor separación semántica

Discursos que usan palabras distintas pero relacionadas (“healthcare” vs. “medical care”) quedarían más próximos, favoreciendo la clasificación.

Mayor robustez con vocabulario limitado
Las representaciones densas ayudan a generalizar frente a términos poco frecuentes: una palabra rara “equity” se beneficia de su proximidad a “fairness” en el espacio de embedding.

Reducción de dimensionalidad
Al trabajar con vectores de tamaño 100–300 en lugar de miles de columnas sparse, los modelos pueden entrenar más rápido y requerir menos memoria.

Necesidad de preentrenamiento o datos adicionales
A cambio, puede ser necesario usar embeddings ya entrenados en un corpus amplio (Wikipedia, Common Crawl) o entrenar propios si el dominio es muy especializado.

Impacto esperado

Frente a métodos basados en frecuencia pura, los embeddings suelen ofrecer un **mejor desempeño** en tareas de clasificación y clustering de texto, especialmente cuando el vocabulario es amplio y variado.

En síntesis, **los word embeddings** aportan un nivel de comprensión semántica que va más allá del simple conteo de términos, y suelen mejorar el desempeño de modelos basados en aprendizaje profundo o clasificadores lineales al capturar relaciones de significado entre palabras.

Característica	Bag-of-Words / TF-IDF	Word Embeddings
Dimensionalidad	Igual al tamaño del vocabulario → muy alta y dispersa (sparse)	Fija y reducida (p. ej. 100–300) → densa
Captura semántica	No captura: “good” y “great” son independientes	Sí: vectores cercanos para palabras de significado similar
Contexto local/global	Solo cuenta frecuencias, sin orden	Skip-Gram/CBOW usan ventana local; GloVe integra coocurrencias globales
Requerimientos	Muy sencillo, rápido de calcular	Necesita preentrenamiento o embedding preexistentes (carga de vectores)
Uso de memoria	Matrices sparse muy grandes	Vectores densos pequeños; más economiza memoria
Rendimiento en modelos	Funciona bien en muchos casos, pero falla en fenómenos de sinonimia, polisemia	Mejora la generalización y el manejo de palabras raras o sinónimos