



UNIVERSIDAD NACIONAL
DE EDUCACIÓN A DISTANCIA

Escuela Técnica Superior de Ingeniería Informática

PRÁCTICA 2: CLUSTERING

Autora: Sabela Alicia Iglesias Sánchez

Minería de Textos

Máster Universitario
en Ciencia e Ingeniería de Datos

31 de diciembre de 2025

Índice

1. Análisis exploratorio de los datos	2
1.1. Distribución del número de palabras por texto	2
1.2. Frecuencias por grupos temáticos	3
2. Creación del <i>goldstandard</i>	5
3. Preprocesamiento de los textos	5
4. Representación vectorial	7
4.1. Representación 1: Frecuencia de términos (TF)	7
4.2. Representación 2: Frecuencia de términos inversa (TF-IDF)	8
4.3. Implementación	8
4.4. Comparativa de las representaciones	8
5. Clústering	10
5.1. K-Means	10
5.2. Algoritmo aglomerativo	10
5.3. Implementación	10
6. Evaluación de los resultados	11
6.1. Matrices de confusión	12
6.2. Análisis de la estructura espacial de los clúster, comparativa de separabilidad	12
6.2.1. Evaluación de la representación TF	13
6.2.2. Evaluación de la representación TDIDF	13
7. Parte opcional: Método Elbow	14
7.1. Implementación del método Elbow	14
7.2. Reprocesamiento de K-Means para el valor óptimo de K	15
8. Conclusiones	15
A. Código Fuente del Proyecto	16
A.1. Análisis exploratorio de los datos	16
A.1.1. Resumen estadístico	16
A.1.2. Boxplots	16
A.1.3. Frecuencia de los grupos	17
A.2. Creación del goldstandard	18
A.3. Preprocesamiento de los textos	18
A.4. Representación de los textos	20
A.5. Evaluación resultados	21
A.6. Script Principal	22
A.7. Utils	24
A.8. Parte opcional	26

Este documento constituye la memoria de la propuesta de solución de la práctica 2 de la asignatura Minería de Texto del Máster en Ciencia e Ingeniería de Datos. El repositorio completo del proyecto esta disponible en el siguiente enlace: <https://github.com/sabelaalicia/clustering-mt-practica2>.

1. Análisis exploratorio de los datos

El conjunto de datos propuestos en la práctica es un subconjunto de la colección 20 Newsgroups (creada por el CMU Text Learning Group). La colección completa contiene 20 categorías, pero en esta práctica nos limitaremos a una subcolección de 20 categorías. Cada texto del corpus es un mensaje publicado en el foro Usenet sobre alguno de los 7 distintos temas. Además, estos mensajes pueden citar a mensajes anteriores de otros usuarios, por lo que se pueden leer segmentos del debate completo a los que llamaremos *threads*.

Esta sección presenta el Análisis Exploratorio de Datos (EDA) del corpus como paso inicial para la tarea de clústering. Para caracterizar la colección, se realizará un estudio estadístico de dos variables: la distribución del *número de palabras por texto* y la distribución de las *frecuencias por categoría*.

El código fuente correspondiente a este análisis exploratorio está disponible en el notebook *analisis_estadistico.ipynb*, accesible en el repositorio del proyecto, así como en el Anexo A de esta misma memoria.

1.1. Distribución del número de palabras por texto

A continuación se presenta un resumen estadístico de los grupos de texto a analizar. El Cuadro 1 muestra, para cada grupo, el número total de documentos disponibles, el número medio de palabras por documento, la desviación estándar de la longitud de los textos y el coeficiente de variación.

Grupo	Número de documentos	Número medio de palabras	Desviación estándar	Coeficiente de variación
comp.sys.ibm.pc.hardware	124	247.70	373.05	1.51
comp.sys.mac.hardware	146	184.38	108.42	0.59
rec.autos	61	216.18	170.52	0.79
rec.sport.hockey	50	273.88	265.68	0.97
sci.electronics	211	265.78	807.43	3.04
talk.politics.guns	141	414.33	473.05	1.14
talk.politics.mideast	72	726.88	1321.51	1.82

Cuadro 1: Resumen estadístico de los grupos de texto

En primer lugar, observamos que los tamaños medios de los documentos varían mucho entre grupos. Por ejemplo, **talk.politics.mideast** y **talk.politics.guns** presentan medias significativamente más altas que las demás (726,88 y 414,33 palabras respectivamente), mientras que grupos como **comp.sys.mac.hardware** y **rec.autos** tienen medias más bajas (184,38 y 216,18).

Estudiando la dispersión de los datos, vemos cómo algunas desviaciones estándar son muy altas respecto a su media (p. ej. **sci.electronics** con std=807,43 y **talk.politics.mideast** con std=1321,51). Esto sugiere la presencia de documentos muy largos o *threads* con contenido citado que inflan la longitud y generan una distribución sesgada.

Para estudiar con más detalle la distribución del número de palabras de estos textos tomaremos el coeficiente de variación ($\frac{std}{mean}$). Observamos que varios grupos tienen dispersión relativa alta (> 1), lo que indica que la longitud de documentos en esos grupos es muy heterogénea.

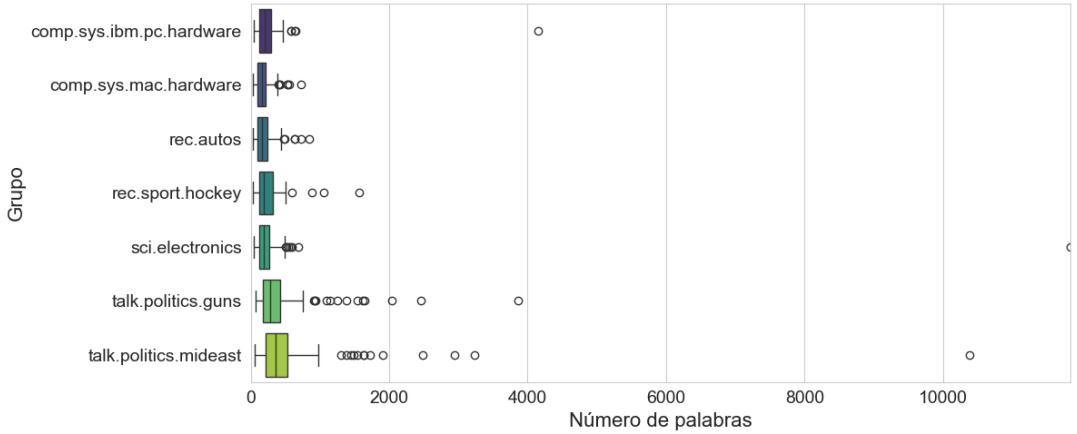


Figura 1: Boxplot completo

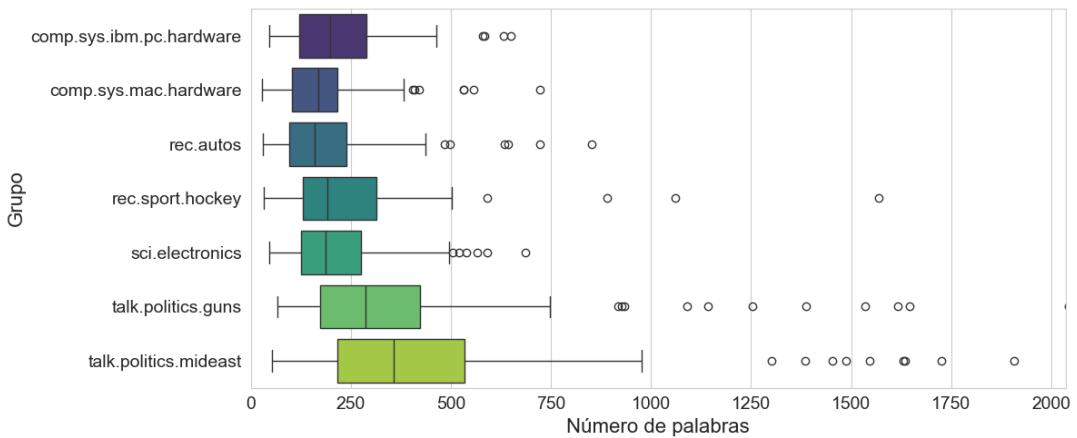


Figura 2: Boxplot del percentil 99 de los datos

En la Figura 1 podemos notar la presencia de outliers excepcionalmente altos. Los grupos `talk.politics.guns` y `talk.politics.mideast` muestran una gran cantidad de valores atípicos de longitud extrema (más de 1000 palabras), lo cual es coherente con su alta variabilidad. Esto puede perjudicar los resultados de nuestra tarea de clústering. Por ello, eliminaremos los registros fuera del percentil 99.

En la Figura 2 podemos ver la gráfica de cajas y bigotes, recortando los ejes de forma que incluyan el percentil 99 de los datos. Esto nos permite estudiar mejor la distribución de cada grupo. Podemos observar mayor variabilidad (caja y bigotes más largos) en los grupos `talk.politics.guns` y `talk.politics.mideast`, relacionadas con debates políticos, así como medias más altas. Esto sugiere que, cuando se trata de estos temas, los usuarios tienden a escribir mensajes más largos y los hilos de citados también crecen. Mientras tanto, los grupos relacionados con temáticas técnicas (`comp.sys`, `sci.electronics`) suelen ser más cortos y concisos, así como tener longitudes más estándar (coeficientes de variación más bajos).

1.2. Frecuencias por grupos temáticos

Antes de abordar la tarea de clústering, examinamos la distribución de los documentos entre los distintos grupos temáticos. Esto nos permitirá identificar posibles desbalances en los datos, lo cual es relevante para interpretar los resultados. A continuación, vemos dos visualizaciones sobre la representación de cada grupo.

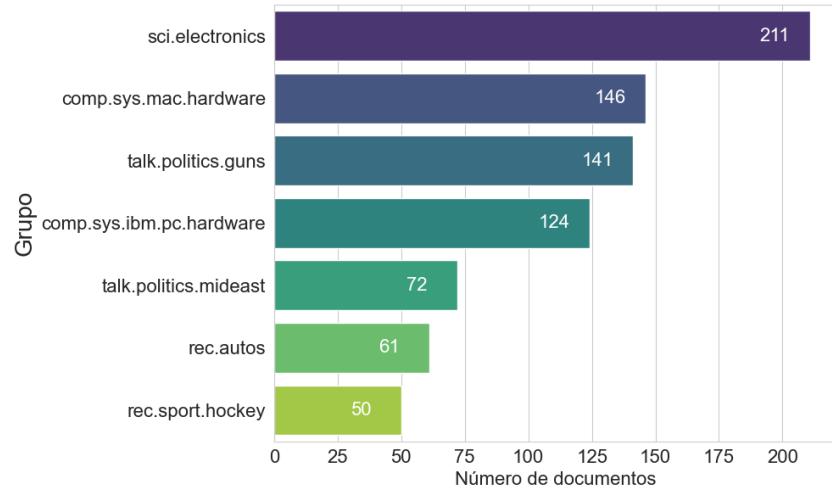


Figura 3: Número de documentos por grupos

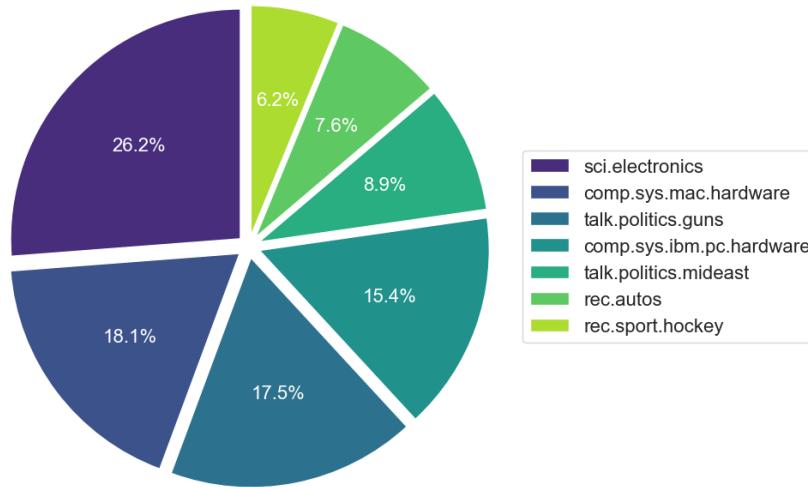


Figura 4: Distribución porcentual de documentos

En las Figuras 3 y 4 podemos notar un claro caso de clases desbalanceadas entre el grupo mayoritario (`sci.electronics` con un 26,2 %) y el minoritario (`rec.sport.hockey` con 6,2 %). Esto supone un riesgo a la hora de realizar nuestra tarea de clústering utilizando el algoritmo K-Means, que podría estar sesgado hacia las clases mayoritarias y provocar problemas de clúster dominante.

Este algoritmo busca minimizar la suma de las distancias cuadradas entre los registros de cada grupo. Sin embargo, en un escenario con clases desbalanceadas, podría ocurrir que las clases más pequeñas fueran absorbidas por clases con mayor representación.

2. Creación del *goldstandard*

Con el fin de evaluar los resultados obtenidos en el proceso de agrupamiento es necesario construir un índice que asocie cada documento con la categoría a la que pertenece de forma correcta. Posteriormente podremos comparar este índice con nuestros resultados para evaluar el rendimiento del modelo. Esta relación texto-grupo se conoce como *goldstandard*.

En nuestro caso, dicho *goldstandard* puede inferirse a partir de la estructura de directorios del corpus: cada carpeta representa un grupo temático y contiene los documentos asociados a dicho grupo. Con el objetivo de construir esta relación, se ha implementado la función auxiliar `crear_goldstandard` (Listing 1), que recorre la jerarquía de carpetas y genera el fichero `goldstandard.csv`, donde se almacena la correspondencia entre cada documento y su etiqueta real.

Listing 1: Función `crear_goldstandard`

```
1 def crear_goldstandard(ruta_raw="data/raw/", salida_csv="data/goldstandard.csv"):
2     rows = []
3     for carpeta in os.listdir(ruta_raw):
4         ruta_carpeta = os.path.join(ruta_raw, carpeta)
5         if os.path.isdir(ruta_carpeta):
6             for fichero in os.listdir(ruta_carpeta):
7                 rows.append({
8                     "fichero": fichero,
9                     "etiqueta_real": carpeta
10                })
11
12 df = pd.DataFrame(rows)
13 df.to_csv(salida_csv, index=False)
```

En el Cuadro 2 se muestra el inicio del fichero generado.

Fichero	Etiqueta real
58343	comp.sys.ibm.pc.hardware
58826	comp.sys.ibm.pc.hardware
58827	comp.sys.ibm.pc.hardware
58828	comp.sys.ibm.pc.hardware
58829	comp.sys.ibm.pc.hardware
...	...

Cuadro 2: Segmento de `goldstandard.csv`

3. Preprocesamiento de los textos

Antes de llevar a cabo la tarea de agrupamiento propiamente, es necesario aplicar un proceso de preprocesamiento sobre los documentos. Esta etapa es crucial para el correcto desempeño de los métodos de clústering. Este proceso de preprocesamiento tiene tres objetivos principales: limpieza, normalización y eliminación de ruido.

Para la **limpieza** de los textos se eliminó la cabecera de cada documento. Esta cabecera incluye metadatos como autor, fecha, identificadores e incluso el nombre del grupo. Por ello debe ser retirada para no intervenir en los resultados de la tarea de agrupamiento. Además, se eliminará la firma final del documento. Estas firmas son muy heterogéneas, por tanto, se realizará una limpieza rudimentaria. En primer lugar se buscará un delimitador `\n--\n` y, si este existe, se borrará el texto posterior. En segundo caso se buscará un email en las 6 últimas líneas y, si este existe, se borrará el texto posterior. La implementación de esta lógica puede consultarse en el Listing 2.

Listing 2: Funciones de limpieza

```
1 def quitar_cabecera(texto):
2
3     partes = re.split(r'\n\s*\n', texto, maxsplit=1)
4     return partes[1] if len(partes) > 1 else texto
5
6 def quitar_firma(texto):
7
8     # delimitador standar
```

```

9     if "\n--\n" in texto:
10        return texto.split("\n--\n")[0]
11
12
13 # si hay email en las 6 lineas, corta desde ahí
14 lines = texto.splitlines()
15 tail = "\n".join(lines[-6:])
16 if re.search(r"\b[\w\.-]+@[ \w\.-]+\.\w+\b", tail):
17     for i, ln in enumerate(lines[:-1], 1):
18         if re.search(r"\b[\w\.-]+@[ \w\.-]+\.\w+\b", ln):
19             cut_index = len(lines) - i
20             return "\n".join(lines[:cut_index])
21
22 return texto
23
24 def limpiar_texto(raw_text):
25
26     texto = quitar_cabecera(raw_text)
27     texto = quitar_firma(texto)
28     return texto

```

Para el proceso de **normalización** se ha decidido convertir todo el texto a minúsculas y eliminar emails, urls y caracteres no alfabéticos, estas son una serie de técnicas estándar en el procesamiento del lenguaje natural siempre y cuando se considere que esta información no es relevante para la extracción de información. La función utilizada para la normalización puede verse en Listing 3.

Listing 3: Función de normalización

```

1 def normalizar_texto(texto):
2 """
3 - Minusculas
4 - Eliminar URLs, emails y caracteres no alfabeticos y numeros
5 """
6
7     texto = texto.lower()
8     texto = re.sub(r'http\S+|www\S+', '', texto)
9     texto = re.sub(r'\b[\w\.-]+@[ \w\.-]+\.\w+\b', '', texto)
10    texto = re.sub(r'In article <[^>]+>, [\w\.-]+@[ \w\.-]+\.\w+ writes:', '', texto)
11    texto = re.sub(r'\d+', '', texto)
12    texto = re.sub(r'^a-zA-Z\s]', ' ', texto)
13    texto = re.sub(r'\s+', ' ', texto).strip()
14
15    return texto

```

Por otro lado, un preprocessamiento imprescindible en este tipo de tareas son la **tokenización** y la **lematización**. Estos procesos se llevaron a cabo a través de la librería *nltk*, que permite dividir el texto en tokens y normalizar cada palabra a su forma canónica. En primer lugar, se aplicó una función de tokenización que separa el texto en tokens y elimina las stopwords y aquellos tokens demasiado cortos con el objetivo de eliminar ruido del texto. Posteriormente, mediante el lematizador, cada token se transforma a su lema. El siguiente bloque presenta las funciones de tokenización y lematización.

Listing 4: Funciones de tokenización y lematización

```

1 def tokenizar_texto(texto):
2     """Tokeniza y elimina stopwords y tokens muy cortos (Utilizando NLTK)"""
3     tokens = nltk.word_tokenize(texto)
4     tokens = [t for t in tokens if t not in stop_words and len(t) > 1]
5     return tokens
6
7 def lematizar_tokens(tokens):
8     """Lematiza los tokens"""
9     return [lemmatizer.lemmatize(t) for t in tokens]

```

El script completo que realiza este preprocessamiento puede consultarse en el archivo *preprocesamiento.py* del repositorio del proyecto así como en el Anexo A.3 de esta memoria.

A modo de ejemplo, para ayudar a visualizar el procesamiento realizado, veamos el archivo *50487.txt* original y el resultado de su preprocessamiento.

Texto sin preprocesar

Newsgroups: comp.sys.mac.hardware
Path: cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!bb3.andrew.cmu.edu!news.sei.cmu.
edu!fs7.ece.cmu.edu!europa.eng.gtefsd.com!howland.reston.ans.net!noc.near.net!saturn.caps.maine.
edu!dartvax!coos.dartmouth.edu!hades
From: hades@coos.dartmouth.edu (Brian V. Hughes)
Subject: Re: File Server Mac
Message-ID: <C52Esv.6M8@dartvax.dartmouth.edu>
Sender: news@dartvax.dartmouth.edu (The News Manager)
Reply-To: hades@Dartmouth.Edu
Organization: Dartmouth College, Hanover, NH
Disclaimer: Personally, I really don't care who you think I speak for.
References: <PKR-050493101102@pkrmac.slac.stanford.edu>
Date: Tue, 6 Apr 1993 13:58:07 GMT
Moderator: Rec.Arts.Comics.Info
Lines: 10

PKR@SLACVM.SLAC.STANFORD.EDU (Patrick Krejcik) writes:

>I saw once an article about a new line of Macs configured to
>work more optimally as file servers.
>Anyone know any more details?

Check out the May issue of MacWorld; the new servers are on the cover. Should be at your favorite newstand.

-Hades

Texto preprocesado

patrick krejcik writes saw article new line mac configured work optimally file server anyone know detail check may issue macworld new server cover favorite newstand hades

4. Representación vectorial

En esta sección, se realizará la representación de los documentos en un espacio vectorial. Durante este proceso se transforma cada documento preprocesado en un vector numérico. La proyección de los textos en este espacio vectorial se realiza en base a una cierta función de peso. Estas funciones asignan un valor numérico (peso) a cada una de las componentes del vector.

Para este trabajo se han empleado dos funciones de peso distintas, con el objetivo de analizar el rendimiento de los modelos para cada uno de los métodos de representación.

4.1. Representación 1: Frecuencia de términos (TF)

Esta representación consiste en crear una matriz documento-término basada en la frecuencia de cada término. Cada elemento del vector de un texto corresponde con la frecuencia de un término en ese mismo texto. De esta forma se captura la información básica del documento.

Este preprocesamiento es un modelo de representación sencillo y de fácil implementación. Captura frecuencias brutas que puede resultar suficiente en tareas de clasificación sencillas (como la detección de *spam*). Sin embargo, puede perder información útil al ignorar la importancia global de cada término. No considera si una palabra es muy común o rara por lo que se puede perder matrices relevantes para la tarea de clústering. Este error será corregido en el segundo modelo de representación.

4.2. Representación 2: Frecuencia de términos inversa (TF-IDF)

La segunda representación será TD-IDF (Term Frequency - Inverse Document Frequency). En ella, el modelo atenúa el peso de los términos más frecuentes en todos los documentos y alza los de términos de un grupo temático.

Este modelo trata de resolver los problemas que presenta la representación TF, es más versátil y considera la importancia global de los términos. Como consecuencia, resulta más adecuado para tareas de agrupamiento. A pesar de ello, siguen existiendo las limitaciones típicas de los modelos de representación basados en frecuencias, como los problemas con palabras extremadamente raras o la falta de contexto semántico.

4.3. Implementación

Para llevar a cabo este proceso utilizaremos los vectorizadores proporcionados por la librería *sklearn*. En el Listing 5 podemos ver el archivo completo *representacion.py*, en él se crean las funciones necesarias para realizar la vectorización posteriormente en el archivo *main.py*. Este archivo está disponible en el Anexo A.6.

Nótese que se han añadido a la lista de stop words *writes* y *article* ya que se observó que eran las dos palabras más frecuentes en los textos con una diferencia significativa. Sin embargo, esto es debido a la estructura de los textos extraídos del foro y no aporta información temática relevante.

Listing 5: Representación TF

```
1 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
2 from sklearn.feature_extraction import _stop_words
3 import os
4 import glob
5
6 def crear_textos(root):
7     archivos = glob.glob(os.path.join(root, "*.txt"))
8     textos = []
9     for ruta in archivos:
10        with open(ruta, "r", encoding="utf-8") as f:
11            textos.append(f.read())
12    return textos
13
14 def vectorizacion_TF(textos):
15     stop_words_custom = list(_stop_words.ENGLISH_STOP_WORDS)+['writes', 'article']
16     vectorizer = CountVectorizer(stop_words=stop_words_custom)
17     matrix_tf = vectorizer.fit_transform(textos)
18     return matrix_tf, vectorizer
19
20 def vectorizacion_TFIDF(textos):
21     stop_words_custom = list(_stop_words.ENGLISH_STOP_WORDS)+['writes', 'article']
22     vectorizer = TfidfVectorizer(stop_words=stop_words_custom)
23     matrix_tfidf = vectorizer.fit_transform(textos)
24     return matrix_tfidf, vectorizer
```

4.4. Comparativa de las representaciones

Con el objetivo de poder examinar ambos métodos se han generado la Figura 5 y la Figura 6, que muestran los pesos de las 10 palabras más pesadas en cada método de vectorización.

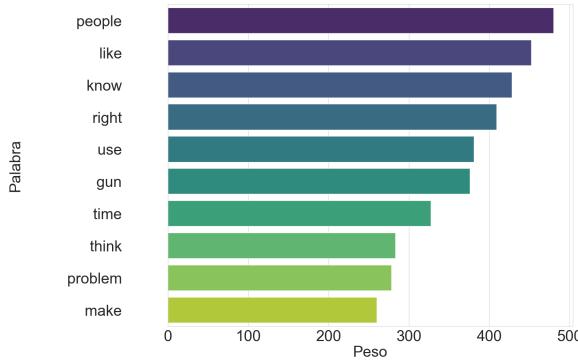


Figura 5: Top 10 palabras por TF

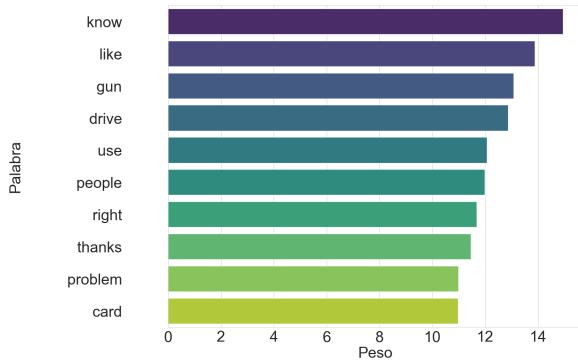


Figura 6: Top 10 palabras por TFIDF

Observando estos gráficos podemos notar que las palabras con mayor peso se repiten en ambos casos, sin embargo, vamos a centrarnos en las diferencias entre ambas gráficas para poder comparar las dos funciones de pesado.

En primer lugar vemos como ciertas palabras comunes en ambas listas se desplazan de posición:

Palabra	Ranking en TF (Figura 5)	Ranking en TF-IDF (Figura 6)
people	1º	6º
like	2º	2º
know	3º	1º

Cuadro 3: Movimiento de palabras comunes

La palabra *people* es la más frecuente en los textos, sin embargo, vemos que ha sufrido cierta penalización por la vectorización TF-IDF, esto es debido a que aparece en una gran variedad de textos del corpus. Las palabras *like* y *know* mantienen su alta posición, esto sugiere que son usadas e documentos específicos.

Por otro lado vemos la existencia de palabras que no aparecen o tienen una posición baja en el ranking TF y suben considerablemente su posición en el ranking TF-IDF. Esto indica que el TF-IDF identificó estos términos como clave para diferenciar semánticamente los textos. Las palabras son *gun*, *drive* y *card*, que están relacionadas, respectivamente, con los grupos temáticos *talk.politics.guns*, *rec.autos* y *comp.sys.ibm.pc.hardware* o *comp.sys.ibm.mac.hardware*.

Palabra	Ranking en TF (Figura 5)	Ranking en TF-IDF (Figura 6)
card	no aparece	10º
drive	no aparece	4º
gun	10º	3º

Cuadro 4: Amplificación de palabras clave

El código fuente para la implementación de esta sección puede visitarse en los archivos *representacion.py* y *utils.py* del repositorio del proyecto, así como en los Anexos A.4 y A.7 de esta memoria.

5. Clústering

El objetivo de esta sección es agrupar automáticamente los documentos de la subcolección en 7 clústeres, intentando que cada grupo contenga documentos temáticamente similares. Esta técnica de aprendizaje no supervisado, es decir, trata de identificar estructuras presentes en el corpus sin utilizar las etiquetas conocidas.

En esta práctica se probaron dos algoritmos de clústering distintos: K-Means y un algoritmo aglomerativo jerárquico.

5.1. K-Means

En primer lugar utilizaremos el algoritmo de aprendizaje automático no supervisado **K-Means**. Este algoritmo itera ajustando los centroides y las asignaciones de los documentos hasta minimizar la suma de las distancias cuadráticas entre cada documento y su centroide asignado. Esto forma clústeres esféricos que, de no ser esta la topología real de los grupos, puede provocar resultados poco satisfactorios.

K-Means es un algoritmo rápido y eficiente para conjuntos de datos grandes y con grupos aproximadamente esféricos y de tamaño similar. Sin embargo, requiere definir previamente el número de clústeres k y es sensible a la inicialización de los centroides.

5.2. Algoritmo algomericativo

Este algoritmo construye una jerarquía de clústeres fusionando iterativamente los documentos más similares. Se comienza con cada documento como un clúster individual y se van combinando según un criterio de similitud promedio hasta alcanzar el número deseado de clústeres. Este algoritmo, al contrario que K-Means, no depende de la inicialización aleatoria, en contrapartida, es más complejo computacionalmente.

5.3. Implementación

Para llevar a cabo este proceso utilizaremos los modelos proporcionados por la librería *sklearn*. En el Listing 6 podemos ver el archivo completo *agrupamiento.py*, en el se crean las funciones necesarias para realizar la tarea de clústering posteriormente en el archivo *main.py*. Este archivo está disponible en el Anexo A.6.

Los resultados del agrupamiento se guardan en un diccionario que se ha guardado en la carpeta *results* del repositorio del proyecto.

Listing 6: Agrupamiento

```
1 import os
2 import numpy as np
3 from sklearn.cluster import KMeans, AgglomerativeClustering
4
5 def guardar_resultados_agrupamiento(resultados_dict, nombre_salida=""
6     resultados_agrupamiento", carpeta="results"):
7     os.makedirs(carpeta, exist_ok=True)
8     ruta_archivo = os.path.join(carpeta, f"{nombre_salida}.npz")
9
10    # Guardar cada array con su clave
11    np.savez(ruta_archivo, **resultados_dict)
12
13    print(f"[OK] Diccionario de resultados guardado en: {ruta_archivo}")
14
15 def run_metodo(X, metodo="kmeans", n_clusters=7):
16    if metodo == "kmeans":
17        modelo = KMeans(n_clusters=n_clusters, random_state=42)
18        labels = modelo.fit_predict(X)
19
20    elif metodo == "agglomerative":
21        if hasattr(X, "toarray"):
22            X = X.toarray()
```

```

23     modelo = AgglomerativeClustering(n_clusters=n_clusters, linkage="ward")
24     labels = modelo.fit_predict(X)
25
26 else:
27     raise ValueError("Metodo no reconocido: usa 'kmeans' o 'agglomerative'")
28
29 return labels
30
31
32 def run_agrupamiento(X_tf, X_tfidf, nombres_archivos):
33     results = {}
34
35     # TF
36     results["tf_kmeans"] = {"labels": run_metodo(X_tf, metodo="kmeans", n_clusters=7),
37                             "nombres": nombres_archivos}
38     results["tf_agglomerative"] = {"labels": run_metodo(X_tf, metodo="agglomerative",
39                                         n_clusters=7),
40                                     "nombres": nombres_archivos}
41
42     # TF-IDF
43     results["tfidf_kmeans"] = {"labels": run_metodo(X_tfidf, metodo="kmeans", n_clusters
44 =7),
45                             "nombres": nombres_archivos}
46     results["tfidf_agglomerative"] = {"labels": run_metodo(X_tfidf, metodo="agglomerative"
47                                         ", n_clusters=7),
48                                     "nombres": nombres_archivos}
49
50     # Guardar todo junto en un .npz
51     guardar_resultados_agrupamiento(results)
52
53 return results

```

6. Evaluación de los resultados

Una vez realizada la tarea de clústering pasamos a la evaluación de los resultados. Para ello nos apoyaremos en 3 métricas Precisión, Cobertura y Medida-F.

- **Precisión** Proporción de elementos correctamente clasificados para cierta clase de todos los que se clasificaron en ella. Además de la precisión por clases calcularemos una precisión total del modelo con el método *macro-averaging*, calculando la media de las precisiones por categoría.
- **Cobertura** Proporción de elementos correctamente clasificados en cierta clase de todos los que realmente pertenecen a ella. Además de la cobertura por clases calcularemos una cobertura total del modelo con el método *macro-averaging*, calculando la media de las precisiones por categoría.
- **Medida - F** Métrica que combina precisión y cobertura.

$$F_1 = \frac{Precision \cdot Cobertura}{Precision + Cobertura}$$

Todo el código utilizado para realizar la evaluación de los resultados está disponible en el Anexo A.5 así como en el archivo *evaluacion.py*. En esta sección no nos centraremos en la implementación del código fuente si no en la evaluación de los resultados del método. El Cuadro 5 muestra los resultados en las tres métricas.

Método	Precisión (macro)	Cobertura (macro)	F1 (macro)
TF y K-Means	0.0110	0.1429	0.0204
TF y Agglomerative	0.1694	0.1470	0.0528
TF-IDF y K-Means	0.1284	0.1404	0.1266
TF-IDF Y Agglomerative	0.2434	0.1554	0.0938

Cuadro 5: Resultados de precisión, cobertura y F1 para cada método de clústering.

Estudiando los resultados expuesto en el Cuadro 5 se puede observar la superioridad de la representación TF-IDF sobre la representación TF simple. En el caso de TF-IDF se obtienen valores de F1 y precisión más altos en ambos algoritmos. Además, el algoritmo aglomerativo superó ligeramente a

K-Means en precisión macro, mientras que K-Means con TF-IDF logró el mejor balance en la métrica F1 (0.1266). Sin embargo, es fundamental notar que todos los valores de F1 son relativamente bajos, lo que indica un rendimiento limitado de los algoritmos bajo cualquier representación.

Para conocer más profundamente la naturaleza de nuestros restados realizaremos dos representaciones clave en la evaluación de tareas de agrupamiento en minería de texto: matrices de confusión y t-SNE plots. Estas gráficas nos ayudarán a conocer en detalle la forma de nuestros clústeres y a discernir el motivo de los resultados insuficientes obtenidos.

6.1. Matrices de confusión

Una matriz de confusión es una herramienta que representa de manera tabular los aciertos y errores de un modelo al comparar las predicciones con los valores reales. La siguiente figura muestra las matrices de confusión de las cuatro predicciones realizadas.

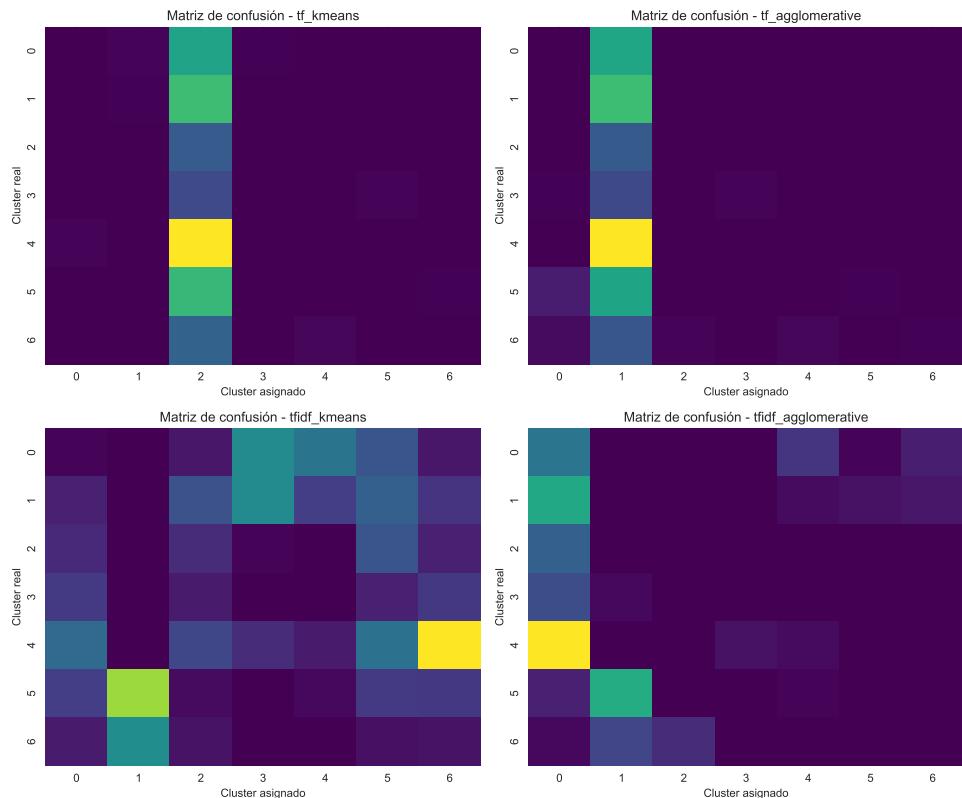


Figura 7: Matrices de confusión de los distintos métodos

En las matrices superiores, correspondientes a la representación TF, se puede ver que hay una gran concentración de colores claros (que indican mayor número de documentos) en columnas específicas (la 2 y la 1 respectivamente). Esto indica que los algoritmos tiende a asignar la mayoría de los documentos a un único clúster. Se observa una diagonal principal débil y los errores están distribuidos por toda la matriz. Esto concuerda con la baja precisión obtenida con TF. Por otro lado, al usar la representación TF-IDF (en las matrices inferiores), la situación mejora ligeramente, aunque aún persiste el problema del clúster dominante.

Los algoritmos, especialmente con la representación TF, fallan al no poder balancear la asignación de clústeres. Estos algoritmos agrupan la mayoría de los documentos en un único clúster dominante, lo cuál provoca una precisión baja.

6.2. Análisis de la estructura espacial de los clúster, comparativa de separabilidad

Para complementar el análisis cuantitativo de las métricas de evaluación, resulta conveniente realizar una representación visual del conjunto de datos. En un primer acercamiento se exploró la reducción de

dimensionalidad mediante el Análisis de Componentes Principales, sin embargo, la variabilidad explicada por las dos primeras componentes resultó insuficiente. Durante la resolución de esta práctica se construyeron las visualizaciones de los clústeres utilizando PCA, pero debido a la escasa variabilidad explicada (11 % en la representación TF y un 1,7 % en la representación TF-IDF) no se consideraron relevantes para añadirlas a esta memoria. Si se quiere consultar estas imágenes o su correspondiente código fuente están disponibles en el Anexo A.7. o en el archivo *utils.py* del repositorio del proyecto.

Ante esta limitación del análisis de componentes principales se optó por emplear t-SNE (t-distributed Stochastic Neighbor Embedding). Este es un algoritmo de aprendizaje no lineal diseñado específicamente para proyectar datos de alta dimensionalidad en espacios de dos. A diferencia de técnicas lineales como PCA, que priorizan la varianza global, t-SNE tiene como objetivo preservar las distancias locales. Esto resulta idóneo para identificar la formación de clústeres. El procedimiento seguido consistió en una reducción mediante PCA a 50 dimensiones (para reducir el ruido) y la aplicación de t-SNE. Este enfoque permite visualizar si los grupos temáticos son separables o si, por el contrario, su solapamiento semántico justifica las dificultades detectadas en la matriz de confusión.

6.2.1. Evaluación de la representación TF

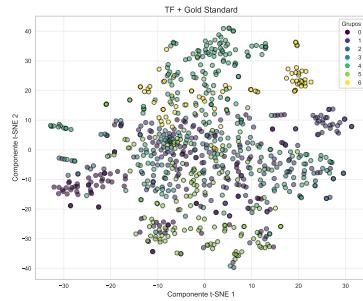


Figura 8: TF + Goldstandar

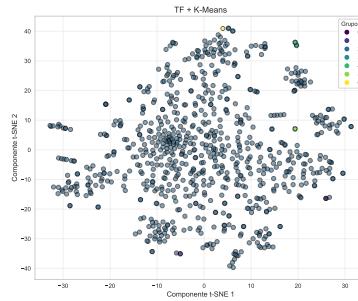


Figura 9: TF + K-means

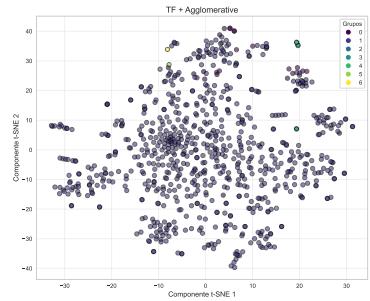


Figura 10: TF + Agglomerative

La Figura 8 muestra la distribución real de los 7 grupos temáticos. Se puede apreciar un solapamiento considerable entre las diferentes islas de puntos. Esto indica que, para esta representación, muchos documentos de distintos temas están semánticamente cerca. Esto es clave para comprender el porqué lo de los resultados pobres en la tarea de clústering utilizando esta representación. Tanto en la Figura 9 como en la 10 se puede ver como, ambos algoritmos de clústering fallan al encontrar estas islas temáticas y agrupan la mayoría de puntos en el mismo el clúster, como se vió en el análisis de las matrices de confusión. Esto puede deberse a una representación pobre que no es capaz de reflejar las diferencias semánticas de los datos o a un solapamiento real.

6.2.2. Evaluación de la representación TDIDF

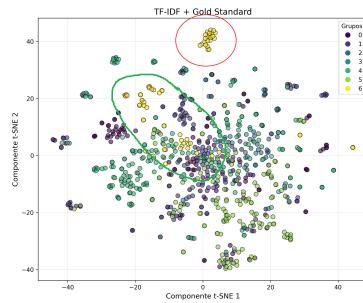


Figura 11: TDIDF + Goldstan-
dar

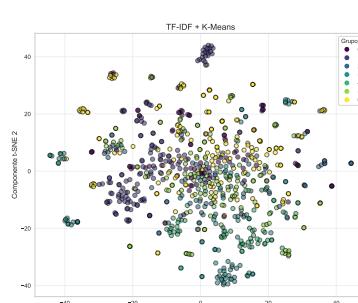


Figura 12: TDIDF + K-means

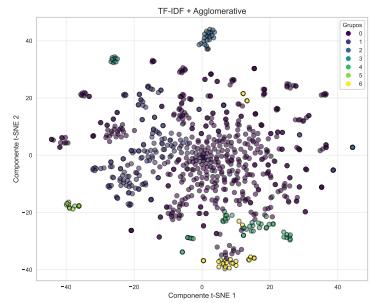


Figura 13: TDIDF + Agglome-
rative

En la Figura 11 se muestra la distribución real de los 7 grupos temáticos en el espacio TDIDF. A diferencia de la representación anterior, aquí se puede apreciar islas de puntos más compactas aunque sigue existiendo solapamiento. Especialmente podemos observar una isla de puntos de la misma temática en la parte superior de la gráfica. En las Figuras 12 y 13, correspondientes respectivamente a los algoritmos K-means y aglomerativo, podemos ver que esa isla superior (señalada en rojo) sí se identificó como un grupo temático, incluyendo incluso la “sombra” mas dispersa que sí se solapa (señalada en verde).

Sin embargo, ambos algoritmos tienen una rendimiento insuficiente, debido a la representación vectorial que no consigue aislar las islas temáticas.

7. Parte opcional: Método Elbow

Para introducir correctamente el método Elbow es necesario conocer previamente el concepto de inercia. Sea X un conjunto de datos $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{R}^d$, agrupado en K clusters C_1, C_2, \dots, C_K con centroides $\mu_1, \mu_2, \dots, \mu_K$, la **inercia** se define como:

$$\text{Inercia} = \sum_{k=1}^K \sum_{x \in C_k} \|x - \mu_k\|^2$$

Es decir, en el contexto de una tarea de agrupamiento utilizando un método basado en centroides, la inercia se define como la suma de cuadrados de la distancia entre cada punto y el centroide de su clúster. Así, es intuitivo pensar que a menor inercia mejor agrupamiento, sin embargo, si solo se atiende a la minimización de la inercia existe el peligro de subdividir un clúster correcto sin mejorar significativamente el resultado. De hecho, si se consideran grupos unipuntuales, la inercia será 0, pero en realidad no se estaría extrayendo ninguna información extra de los datos. Por ello, es necesario buscar un equilibrio entre inercia y número de clústers, de esta necesidad surge el método Elbow.

El método Elbow, o del codo, es un método heurístico que se utiliza para determinar el número de grupos en un conjunto de datos. Dado un modelo de agrupamiento basado en centroides, se aplica este modelo para cierto rango de valores de k . Después, se grafica la inercia obtenida para cada valor de k , I_k frente a k . La curva seguirá una tendencia descendiente debido a la propia naturaleza del concepto de inercia. Es habitual que esta tendencia descendente sea más pronunciada en los primeros valores de k y que, llegado cierto valor k_1 se estabilice y la pendiente no sea tan pronunciada. Este punto es el codo (o *elbow*) que da nombre al método. Para ese valor, k_1 se habrá encontrado el equilibrio entre inercia y número de clústers.

7.1. Implementación del método Elbow

En el problema que nos ocupa vamos a utilizar el método del codo para tratar de averiguar cual es el número de clústers más adecuado para separar nuestros datos utilizando el método K-means. Aplicaremos el método del codo sobre dos variantes de vectorización: TF y TF-IDF. El método aglomerativo lo dejaremos a un lado en esta sección ya que no está basado en centroides.

Para la implementación de este método en python se ha utilizado la librería *scikit-learn* para ejecutar el modelo K-Means para los distintos valores de k . El código fuente de la implementación de este método se puede revisar tanto en el archivo *opcional.py* del repositorio del proyecto como en el Anexo A.8 de esta memoria.

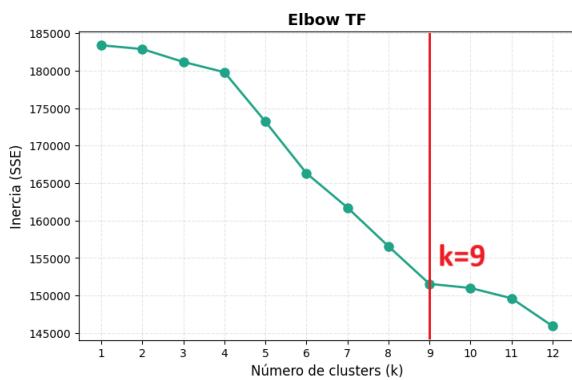


Figura 14: TF Elbow Method

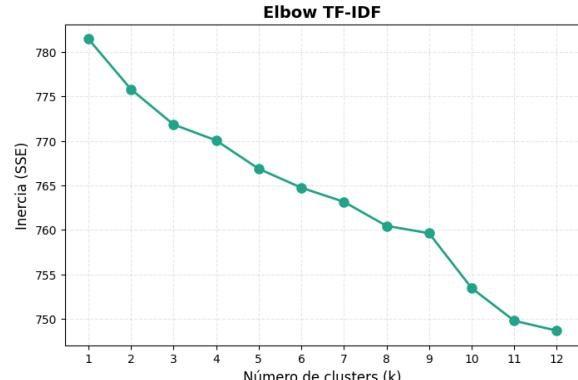


Figura 15: TDIDF Elbow Method

En la Figura 14 se identifica un codo claro en $k = 9$, donde la inercia comienza a estabilizarse. En cambio, en la variante TF-IDF, la inercia desciende de manera más constante, lo que sugiere que la ponderación por importancia de términos (diferencia básica entre TF y TFIDF) suaviza las diferencias

entre clústeres. Por ello, utilizaremos $k = 9$ para la realizar al algoritmo de K-means con la vectorización TF.

7.2. Reprocesamiento de K-Means para el valor óptimo de K

En esta sección se ha realizado un reprocesamiento de K-Means utilizando la vectorización TF para $k = 9$. Aunque en la parte opcional se haya utilizado un valor de k diferente al de la parte obligatoria, para la interpretación y evaluación de los resultados se han conservado las etiquetas originales de la parte obligatoria como identificadores de los documentos. Esto permite mapear cada documento dentro de los nuevos clústeres. Los resultados obtenidos se muestran en la Tabla 6. Se ha obtenido una precisión macro de 0,1208, un recall macro de 0,1103 y un F1 macro de 0,0193, valores bajos. Esto indica que, si bien el algoritmo logra formar grupos coherentes en términos de inercia interna, estos no corresponden con los grupos etiquetados en el *goldstandard*.

Métricas k=9 (TF + KMeans)	
Precisión (macro)	0.1208
Recall (macro)	0.1103
F1 (macro)	0.0193

Cuadro 6: Resultados de evaluación para $k=9$ usando TF + KMeans.

Para la realización de la parte opcional de esta práctica se ha utilizado como texto de referencia el Capítulo 8 del libro “*Hands-On Machine Learning with Scikit-Learn and TensorFlow*” de Aurélien Géron.

8. Conclusiones

En este trabajo se ha abordado la tarea de clústering de documentos sobre una subcolección del corpus *20 Newsgroups*. El objetivo de esta práctica es analizar y comparar diferentes estrategias de representación vectorial y algoritmos de clústering.

En primer lugar, en el análisis exploratorio se han detectado dos características del corpus: **heterogeneidad en la longitud de los documentos** y **desbalance entre las distintas categorías**. Estas dos condiciones influyeron negativamente en los resultados de los algoritmos de clústering, especialmente en K-Means, por tratarse de un método basado en centroides.

En lo relativo a la representación vectorial, la comparativa entre los modelos TF y TF-IDF muestra un mejor comportamiento de este último en la tarea de clústering. La ponderación TF-IDF atenua el peso de los términos muy frecuentes en el corpus y refuerza aquellos que resultan más característicos de determinados grupos temáticos. Esto se pudo observar en el análisis de los términos con mayor peso. Sin embargo, incluso utilizando esta representación, el rendimiento de ambos modelos fue muy limitado. Esto indica que **los modelos basados únicamente en frecuencias de términos no son capaces de capturar de forma completa la información semántica de este corpus**.

En cuanto a los algoritmos de clústering empleados, el método aglomerativo presenta valores de precisión macro ligeramente superiores a los obtenidos por K-Means, especialmente cuando se utiliza la representación TF-IDF. No obstante, en ambos algoritmos se observa un comportamiento similar, con la presencia de un **problema de clúster dominante**, al que se asigna la mayoría de los documentos. Este efecto puede apreciarse claramente tanto en las matrices de confusión como en las representaciones mediante t-SNE.

La exploración visual de la gráfica t-SNE muestra que, incluso en el *gold standard*, las categorías se solapan en ambos espacios vectoriales. Esto indica que el **solapamiento semántico** entre temas es real, o bien que los algoritmos de representación vectorial basados en frecuencias no resultan adecuados para este corpus.

En conjunto, los resultados obtenidos evidencian las limitaciones de los métodos clásicos de clústering y de las representaciones basadas en frecuencias de términos para tareas con fuerte solapamiento semántico y clases desbalanceadas. Como líneas futuras de trabajo, sería interesante explorar representaciones semánticas más ricas, como *word embeddings* o *sentence embeddings*.

A. Código Fuente del Proyecto

A.1. Análisis exploratorio de los datos

A.1.1. Resumen estadístico

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import pandas as pd
6
7 sns.set_style("whitegrid")
8 sns.set_palette("viridis")
9
10
11 # Función para contar palabras
12
13 def count_words(file_path):
14     with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
15         text = file.read()
16         words = text.split()
17         return len(words)
18
19
20
21 # Función para describir la colección
22
23 def describe_collection(root_path='../data/raw'):
24     stats_by_group = {}
25
26     for group_name in os.listdir(root_path):
27         group_path = os.path.join(root_path, group_name)
28
29         if os.path.isdir(group_path):
30             file_lengths = []
31             for filename in os.listdir(group_path):
32                 file_path = os.path.join(group_path, filename)
33                 word_count = count_words(file_path)
34                 file_lengths.append(word_count)
35
36             num_documents = len(file_lengths)
37             mean_words = np.mean(file_lengths)
38             std_words = np.std(file_lengths)
39
40             stats_by_group[group_name] = {
41                 'num_documents': num_documents,
42                 'mean_words': mean_words,
43                 'std_words': std_words
44             }
45
46     # Mostrar resultados
47
48     for group, stats in stats_by_group.items():
49         print(f"Grupo: {group}")
50         print(f"Número de documentos: {stats['num_documents']}")
51         print(f"Número medio de palabras: {stats['mean_words']:.2f}")
52         print(f"Desviación estándar de palabras: {stats['std_words']:.2f}")
53         print(f"Coeficiente de variación: {stats['std_words']} / {stats['mean_words']:.2f}")
54         print("-" * 50)
55
56     return stats_by_group
57
58 stats = describe_collection("../data/raw")
```

A.1.2. Boxplots

```
1
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 def prepare_boxplot_data(root_path):
6     """
```

```

7     Prepara un DataFrame para boxplot con columnas: 'Grupo', 'Palabras'
8     """
9     data = []
10    for group_name in os.listdir(root_path):
11        group_path = os.path.join(root_path, group_name)
12        if os.path.isdir(group_path):
13            for filename in os.listdir(group_path):
14                file_path = os.path.join(group_path, filename)
15                with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
16                    text = f.read()
17                    num_words = len(text.split())
18                    data.append({'Grupo': group_name, 'Palabras': num_words})
19    return pd.DataFrame(data)
20
21 def plot_boxplot(df, x):
22     """
23     Genera un boxplot horizontal mostrando la distribucion de 'Palabras' por 'Grupo'.
24
25     Args:
26         df (pd.DataFrame): DataFrame con las columnas 'Grupo' y 'Palabras'.
27         x (float): Percentil maximo para limitar el eje X.
28     """
29     sns.set_style("whitegrid")
30     plt.figure(figsize=(12, 5))
31
32     ax = sns.boxplot(y='Grupo', x='Palabras', data=df, palette='viridis')
33
34
35     x_max = np.percentile(df['Palabras'], x)
36     ax.set_xlim(0, x_max)
37
38     ax.set_xlabel("Numero de palabras", fontsize=16)
39     ax.set_ylabel("Grupo", fontsize=16)
40
41     ax.tick_params(axis='x', labelsize=14)
42     ax.tick_params(axis='y', labelsize=14)
43
44     plt.tight_layout()
45     plt.savefig(f'../docs/images/boxplot_palabras_por_grupo_percentil_{x}.png')
46     plt.show()
47
48 df = prepare_boxplot_data("../data/raw")
49 plot_boxplot(df, 100)
50 plot_boxplot(df, 99)

```

A.1.3. Frecuencia de los grupos

```

1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import os
5 import numpy as np
6
7 root_path = '../data/raw'
8 counts = []
9
10 for group_name in sorted(os.listdir(root_path)):
11     group_path = os.path.join(root_path, group_name)
12     if os.path.isdir(group_path):
13         n = sum(1 for f in os.listdir(group_path) if os.path.isfile(os.path.join(
14             group_path, f)))
15     counts.append({'grupo': group_name, 'n_documentos': n})
16 df_counts = pd.DataFrame(counts).sort_values('n_documentos', ascending=False).reset_index(
17     drop=True)
18
19 # Grafico de barras
20 sns.set_style('whitegrid')
21 fig, ax = plt.subplots(figsize=(10, max(6, 0.4 * len(df_counts))))
22 sns.barplot(data=df_counts, x='n_documentos', y='grupo', palette='viridis', ax=ax)
23
24 for container in ax.containers:
25     for bar in container.get_children():
26         x_val = bar.get_width()
27         y_val = bar.get_y() + bar.get_height() / 2

```

```

26         label = f"{{int(x_val)}"
27         ax.text(x_val - 20, y_val, label,
28                 color='white', ha='left', va='center', fontsize=16)
29     ax.set_xlabel('Número de documentos', fontsize=16)
30     ax.set_ylabel('Grupo', fontsize=20)
31     ax.tick_params(axis='both', labelsize=16)
32
33 plt.tight_layout()
34 plt.savefig(f"../docs/images/barras.png")
35 plt.show()
36
37 # Grafico de tarta con leyenda externa
38 total_docs = df_counts['n_documentos'].sum()
39 df_counts['porcentaje'] = (df_counts['n_documentos'] / total_docs) * 100
40
41 plt.figure(figsize=(10, 10))
42 colores = sns.color_palette('viridis', len(df_counts))
43
44 wedges, texts, autotexts = plt.pie(
45     df_counts['n_documentos'],
46     labels=None,
47     autopct='%1.1f%%',
48     startangle=90,
49     colors=colores,
50     explode=[0.05] * len(df_counts),
51     textprops={'fontsize': 16, 'color':'white'},
52     radius=1
53 )
54
55 plt.legend(
56     wedges,
57     df_counts['grupo'],
58     loc="center_left",
59     bbox_to_anchor=(1, 0, 0.5, 1),
60     fontsize=16
61 )
62
63 plt.axis('equal')
64 plt.tight_layout()
65 plt.savefig(f"../docs/images/pie_chart_groups.png", bbox_inches='tight')
66 plt.show()

```

A.2. Creacion del goldstandard

```

1 import os
2 import pandas as pd
3
4 def crear_goldstandard(ruta_raw="data/raw/", salida_csv="data/goldstandard.csv"):
5     rows = []
6     for carpeta in os.listdir(ruta_raw):
7         ruta_carpeta = os.path.join(ruta_raw, carpeta)
8         if os.path.isdir(ruta_carpeta):
9             for fichero in os.listdir(ruta_carpeta):
10                 rows.append({
11                     "fichero": fichero,
12                     "etiqueta_real": carpeta
13                 })
14
15     df = pd.DataFrame(rows)
16     df.to_csv(salida_csv, index=False)
17     print(f"Goldstandard creado en [{salida_csv}], con {len(rows)} documentos.")
18
19 if __name__ == "__main__":
20     crear_goldstandard()

```

A.3. Preprocesamiento de los textos

```

1 import os
2 from pathlib import Path
3 import re
4 import nltk
5 from nltk.corpus import stopwords

```

```

6  from nltk.stem import WordNetLemmatizer
7  import numpy as np
8
9  nltk.download('punkt')
10 nltk.download('punkt_tab')
11 nltk.download('stopwords')
12 nltk.download('wordnet')
13 nltk.download('omw-1.4')
14
15 stop_words = set(stopwords.words('english'))
16 lemmatizer = WordNetLemmatizer()
17
18 def quitar_cabecera(texto):
19
20     partes = re.split(r'\n\s*\n', texto, maxsplit=1)
21     return partes[1] if len(partes) > 1 else texto
22
23 def quitar_firma(texto):
24
25     if "\n--\n" in texto:
26         return texto.split("\n--\n")[0]
27
28 # si hay email en ltimas 6 lineas, corta desde ah
29 lines = texto.splitlines()
30 tail = "\n".join(lines[-6:])
31 if re.search(r"\b[\w\.-]+@[\\w\.-]+\.\w+\b", tail):
32     for i, ln in enumerate(lines[::-1], 1):
33         if re.search(r"\b[\w\.-]+@[\\w\.-]+\.\w+\b", ln):
34             cut_index = len(lines) - i
35             return "\n".join(lines[:cut_index])
36
37 return texto
38
39 def limpiar_texto(raw_text):
40
41     texto = quitar_cabecera(raw_text)
42     texto = quitar_firma(texto)
43     return texto
44
45 def normalizar_texto(texto):
46     """
47     - Minusculas
48     - Eliminar URLs, emails y caracteres no alfabeticos y numeros
49     """
50
51     texto = texto.lower()
52     texto = re.sub(r'http\S+|www\S+', '', texto)
53     texto = re.sub(r'\b[\w\.-]+@[\\w\.-]+\.\w+\b', '', texto)
54     texto = re.sub(r'Inarticle<[^>]+>, u[\w\.-]+@[\w\.-]+\.\w+writes', '', texto)
55     texto = re.sub(r'\d+', '', texto)
56     texto = re.sub(r'^a-zA-Z\s$', ' ', texto)
57     texto = re.sub(r'\s+', ' ', texto).strip()
58
59 return texto
60
61 def tokenizar_texto(texto):
62     """Tokeniza y elimina stopwords y tokens muy cortos (Utilizando NLTK)"""
63     tokens = nltk.word_tokenize(texto)
64     tokens = [t for t in tokens if t not in stop_words and len(t) > 1]
65     return tokens
66
67 def lematizar_tokens(tokens):
68     """Lematiza los tokens"""
69     return [lemmatizer.lemmatize(t) for t in tokens]
70
71 def eliminar_outliers(ruta_preprocesada="..../data/preprocessed/", percentil_inf=0.5,
72 percentil_sup=99.5):
73     ruta_preprocesada = Path(ruta_preprocesada)
74     longitudes = []
75
76     for fichero in ruta_preprocesada.glob("*.txt"):
77         with open(fichero, "r", encoding="utf-8") as f:
78             tokens = f.read().split()
79             longitudes.append(len(tokens))

```

```

80     min_len = np.percentile(longitudes, percentil_inf)
81     max_len = np.percentile(longitudes, percentil_sup)
82     print(f"Eliminando documentos fuera del rango [{min_len}, {max_len}] palabras")
83
84     eliminados = 0
85     for fichero in ruta_preprocesada.glob("*.txt"):
86         with open(fichero, "r", encoding="utf-8") as f:
87             tokens = f.read().split()
88             if len(tokens) < min_len or len(tokens) > max_len:
89                 fichero.unlink()
90                 eliminados += 1
91
92     print(f"Documentos eliminados: {eliminados}")
93
94
95 def preprocesar_carpeta(ruta_raw="../data/raw/", ruta_destino="../data/preprocessed/"):
96     ruta_raw = Path(ruta_raw)
97     ruta_destino = Path(ruta_destino)
98     ruta_destino.mkdir(parents=True, exist_ok=True)
99     longitudes = []
100    for carpeta in ruta_raw.iterdir():
101        if carpeta.is_dir():
102            for fichero in carpeta.iterdir():
103                if fichero.is_file():
104                    with open(fichero, "r", encoding="latin1") as f:
105                        texto = f.read()
106                        texto_limpio = limpiar_texto(texto)
107                        texto_normalizado = normalizar_texto(texto_limpio)
108                        texto_tokens = tokenizar_texto(texto_normalizado)
109                        texto_lemmatizado = lematizar_tokens(texto_tokens)
110                        texto_final = " ".join(texto_lemmatizado)
111
112                        longitudes.append(len(texto_lemmatizado))
113                        ruta_destino.mkdir(parents=True, exist_ok=True)
114                        with open(ruta_destino / (fichero.name + ".txt"), "w", encoding="utf-
115                            -8") as f_out:
116                            f_out.write(texto_final)
117
118    eliminar_outliers(ruta_destino)
119    print("Preprocesamiento completado")
120
121
122 if __name__ == "__main__":
123     preprocesar_carpeta()

```

A.4. Representación de los textos

```

1 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
2 from sklearn.feature_extraction import _stop_words
3 import os
4 import glob
5
6 def crear_textos(root):
7     archivos = sorted(glob.glob(os.path.join(root, "*.txt")))
8     textos = []
9     nombres = []
10    for ruta in archivos:
11        with open(ruta, "r", encoding="utf-8") as f:
12            textos.append(f.read())
13            nombres.append(os.path.splitext(os.path.basename(ruta))[0])
14    return textos, nombres
15
16 def vectorizacion_TF(textos):
17     stop_words_custom = list(_stop_words.ENGLISH_STOP_WORDS)+['writes', 'article']
18     vectorizer = CountVectorizer(stop_words=stop_words_custom)
19     matrix_tf = vectorizer.fit_transform(textos)
20     return matrix_tf, vectorizer
21
22 def vectorizacion_TFIDF(textos):
23     stop_words_custom = list(_stop_words.ENGLISH_STOP_WORDS)+['writes', 'article']
24     vectorizer = TfidfVectorizer(stop_words=stop_words_custom)
25     matrix_tfidf = vectorizer.fit_transform(textos)
26     return matrix_tfidf, vectorizer

```

A.5. Evaluación resultados

```

1 import os
2 import numpy as np
3 import pandas as pd
4 from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
5 from sklearn.preprocessing import LabelEncoder
6 from sklearn.decomposition import PCA
7 from sklearn.manifold import TSNE
8 import matplotlib.pyplot as plt
9 import pandas as pd
10 from pathlib import Path
11
12 def filtrar_goldstandard(ruta_gold="data/goldstandard.csv", archivos_procesados=None):
13     df_gold = pd.read_csv(ruta_gold)
14     df_gold_filtrado = df_gold[df_gold["fichero"].astype(str).isin(archivos_procesados)]
15     .copy()
16
17     return df_gold_filtrado
18
19 def cargar_goldstandard(ruta_csv):
20
21     df = pd.read_csv(ruta_csv)
22     etiquetas = df["etiqueta_real"].values
23     le = LabelEncoder()
24     etiquetas_codificadas = le.fit_transform(etiquetas)
25
26     return etiquetas_codificadas, le
27
28 def evaluar_un_metodo(y_true, y_pred):
29     precision = precision_score(y_true, y_pred, average="macro", zero_division=0)
30     recall = recall_score(y_true, y_pred, average="macro", zero_division=0)
31     f1 = f1_score(y_true, y_pred, average="macro", zero_division=0)
32     matriz = confusion_matrix(y_true, y_pred)
33
34     return precision, recall, f1, matriz
35
36 import matplotlib.pyplot as plt
37 import numpy as np
38 from sklearn.decomposition import PCA
39 from sklearn.manifold import TSNE
40 import os
41
42 def plot_pca_tsne(matrix, labels, titulo="Visualización PCA+t-SNE", n_componentes_pca=50, n_componentes_tsne=2, random_state=42, guardar=True, guardar_ruta=None):
43     print(f"Aplicando PCA a {n_componentes_pca} dimensiones...")
44     pca = PCA(n_components=n_componentes_pca, random_state=random_state)
45     matriz_pca = pca.fit_transform(matrix)
46     varianza_explicada = pca.explained_variance_ratio_.sum()
47     print(f"Varianza explicada por PCA: {varianza_explicada:.4f}")
48
49     print(f"Aplicando t-SNE a {n_componentes_tsne} dimensiones...")
50     tsne = TSNE(n_components=n_componentes_tsne, random_state=random_state, perplexity=30, n_iter_without_progress=1000)
51     coordenadas_tsne = tsne.fit_transform(matriz_pca)
52     print("Reducción dimensionalidad completada.")
53
54     unique_labels = np.unique(labels)
55     colores_discretos = plt.cm.viridis(np.linspace(0, 1, len(unique_labels)))
56
57     label_to_color = {label: colores_discretos[i] for i, label in enumerate(unique_labels)}
58
59     plt.figure(figsize=(10, 8))
60
61     scatter = plt.scatter(coordenadas_tsne[:, 0], coordenadas_tsne[:, 1],
62                           c=[label_to_color[label] for label in labels], alpha=0.6,
63                           edgecolors='k', s=50)
64
65     handles = []
66     for label, color in label_to_color.items():
67         handles.append(plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color,
68                               markersize=10, label=str(label)))

```

```

68     plt.legend(handles=handles, title="Grupos", loc='best')
69
70     plt.xlabel('Componente t-SNE 1', fontsize=12)
71     plt.ylabel('Componente t-SNE 2', fontsize=12)
72     plt.title(titulo, fontsize=14)
73
74     plt.grid(True, alpha=0.3)
75
76     if guardar:
77         os.makedirs(os.path.dirname(guardar_ruta), exist_ok=True)
78         plt.savefig(guardar_ruta, dpi=300, bbox_inches='tight')
79         print(f"Gráfico guardado en: {guardar_ruta}")
80
81     plt.show()
82
83     return coordenadas_tsne, labels
84
85
86 def run_evaluacion(resultados_dict, ruta_goldstandard="data/goldstandard.csv",
87                    carpeta_resultados="results", archivos_procesados=None):
88     df_gold = filtrar_goldstandard(ruta_gold=ruta_goldstandard, archivos_procesados=
89                                    archivos_procesados)
90
91     y_true = df_gold["etiqueta_real"].values
92
93     le = LabelEncoder()
94     y_true_cod = le.fit_transform(y_true)
95
96     resultados_eval = {}
97
98     os.makedirs(carpeta_resultados, exist_ok=True)
99     ruta_txt = os.path.join(carpeta_resultados, "resultados_evaluacion.txt")
100
101    with open(ruta_txt, "w", encoding="utf-8") as f:
102        f.write("==RESULTADOS DE EVALUACION==\n\n")
103
104        for nombre, info in resultados_dict.items():
105            y_pred = info["labels"]
106            nombres_pred = info["nombres"]
107            pred_dict = {os.path.splitext(n)[0]: label for n, label in zip(nombres_pred,
108                y_pred)}
109            y_pred_ordenado = np.array([pred_dict[str(n)] for n in df_gold["fichero"]])
110            # Calcular métricas
111            precision, recall, f1, matriz = evaluar_un_metodo(y_true_cod, y_pred_ordenado)
112
113            resultados_eval[nombre] = {
114                "precision": precision,
115                "recall": recall,
116                "f1": f1,
117                "matriz_confusion": matriz
118            }
119
120            f.write(f"--M todo:{nombre}--\n")
121            f.write(f"Precision (macro):{precision:.4f}\n")
122            f.write(f"Cobertura (macro):{recall:.4f}\n")
123            f.write(f"F1 (macro):{f1:.4f}\n")
124            f.write("Matriz de confusión:\n")
125            f.write(np.array2string(matriz))
126
127            print(f"Resultados de evaluación guardados en: {ruta_txt}")
128            return resultados_eval

```

A.6. Script Principal

```

1 from representacion import crear_textos, vectorizacion_TF, vectorizacion_TFIDF
2 from utils import plot_palabras_mas_frecuentes, plot_matrices_confusion, plot_wordcloud
3 from agrupamiento import run_agrupamiento, run_metodo
4 from evaluacion import run_evaluacion, plot_pca_tsne, filtrar_goldstandard
5 from utils import plot_scatter_pca
6 from sklearn.preprocessing import LabelEncoder
7 from opcional import run_elbow_kmeans
8

```

```

9
10 def main():
11
12     ##### Vectorizaci n #####
13
14     #ruta = "C:\\\\Users\\\\sabel\\\\OneDrive\\\\Escritorio\\\\Master\\\\MT_conv2\\\\Practica 2\\\\
15     #clustering-mt-practica2\\\\data\\\\preprocessed"
16     ruta = "C:\\\\Users\\\\Sabela\\\\clustering-mt-practica2\\\\data\\\\preprocessed"
17
18     textos, nombres_archivos = crear_textos(ruta)
19     print(f"Documentos cargados: {len(textos)}")
20
21     X_tf, vectorizer_tf = vectorizacion_TF(textos)
22     print(f"Matriz TF generada: {X_tf.shape}")
23
24     X_tfidf, vectorizer_tfidf = vectorizacion_TFIDF(textos)
25     print(f"Matriz TF-IDF generada: {X_tfidf.shape}")
26
27     plot_palabras_mas_frecuentes(X_tf, vectorizer_tf, vector_type="TF", top_n=10, guardar=False)
28     plot_palabras_mas_frecuentes(X_tfidf, vectorizer_tfidf, vector_type="TFIDF", top_n=10, guardar=False)
29     plot_wordcloud(X_tf, vectorizer_tf, vector_type="TF", guardar=True)
30     plot_wordcloud(X_tfidf, vectorizer_tfidf, vector_type="TFIDF", guardar=True)
31
32     ##### Agrupamiento #####
33     results = run_agrupamiento(X_tf, X_tfidf, nombres_archivos)
34
35     matrices = [X_tf, X_tf, X_tfidf, X_tfidf]
36
37     labels_list = [results['tf_kmeans']['labels'],
38                    results['tf_agglomerative']['labels'],
39                    results['tfidf_kmeans']['labels'],
40                    results['tfidf_agglomerative']['labels']]
41
42     titles = ['TF-KMeans', 'TF-Agglomerative', 'TFIDF-KMeans', 'TFIDF-Agglomerative']
43
44     plot_scatter_pca(matrices, labels_list, nombres=nombres_archivos, titles=titles, guardar=False)
45
46     ##### Evaluaciones #####
47     evals = run_evaluacion(results, ruta_goldstandard="data/goldstandard.csv",
48                            archivos_procesados=nombres_archivos)
49     plot_matrices_confusion(evals, guardar=True)
50
51     #TF con goldstandard
52     df_gold = filtrar_goldstandard("data/goldstandard.csv", nombres_archivos)
53     etiquetas_gold = LabelEncoder().fit_transform(df_gold["etiqueta_real"])
54     plot_pca_tsne(
55         matrix=X_tf.toarray(),
56         labels=etiquetas_gold,
57         titulo="TF+Gold Standard",
58         guardar_ruta="docs/images/pca_tsne_tf_gold.pdf"
59     )
60
61     #TF-IDF con goldstandard
62     plot_pca_tsne(
63         matrix=X_tfidf.toarray(),
64         labels=etiquetas_gold,
65         titulo="TF-IDF+Gold Standard",
66         guardar_ruta="docs/images/pca_tsne_tfidf_gold.pdf"
67     )
68
69     #TF con K-Means
70     plot_pca_tsne(
71         matrix=X_tf.toarray(),
72         labels=results['tf_kmeans']['labels'],
73         titulo="TF+K-Means",
74         guardar_ruta="docs/images/pca_tsne_tf_kmeans.pdf"
75     )
76     #TF-IDF con K-Means
77     plot_pca_tsne(
78         matrix=X_tfidf.toarray(),

```

```

78     labels=results['tfidf_kmeans']['labels'],
79     titulo="TF-IDF+K-Means",
80     guardar_ruta="docs/images/pca_tsne_tfidf_kmeans.pdf"
81   )
82   #TF con Agglomerative
83   plot_pca_tsne(
84     matrix=X_tf.toarray(),
85     labels=results['tf_agglomerative']['labels'],
86     titulo="TF+Agglomerative",
87     guardar_ruta="docs/images/pca_tsne_tf_agglomerative.pdf"
88   )
89   #TF-IDF con Agglomerative
90   plot_pca_tsne(
91     matrix=X_tfidf.toarray(),
92     labels=results['tfidf_agglomerative']['labels'],
93     titulo="TF-IDF+Agglomerative",
94     guardar_ruta="docs/images/pca_tsne_tfidf_agglomerative.pdf"
95   )
96
97
98 ##### Opcional: M todo Elbow #####
99 print("Ejecutando m todo Elbow para TF...")
100 run_elbow_kmeans(X_tf, max_k=12, titulo="Elbow_TF", guardar=True, ruta_guardado="docs
    /images/elbow_tf.png")
101
102 print("Ejecutando m todo Elbow para TF-IDF...")
103 run_elbow_kmeans(X_tfidf, max_k=12, titulo="Elbow_TF-IDF", guardar=True,
    ruta_guardado="docs/images/elbow_tfidf.png")
104
105 labels = run_metodo(X_tf, metodo="kmeans", n_clusters=9)
106 print(labels)
107
108 ##### Evaluacion para k=9 #####
109 resultados_k9 = {
110   "tf_kmeans_k9": {
111     "labels": labels,
112     "nombres": nombres_archivos
113   }
114 }
115
116 eval_k9 = run_evaluacion(resultados_k9, ruta_goldstandard="data/goldstandard.csv",
    archivos_procesados=nombres_archivos)
117
118 metrica_k9 = eval_k9["tf_kmeans_k9"]
119
120 print("\n==== M tricas k=9 (TF+KMeans) ===")
121 print(f"Precision (macro): {metrica_k9['precision']:.4f}")
122 print(f"Recall (macro): {metrica_k9['recall']:.4f}")
123 print(f"F1 (macro): {metrica_k9['f1']:.4f}")
124 print("Matriz de confusión:\n", metrica_k9["matriz_confusion"])
125
126
127
128 if __name__ == "__main__":
129   main()

```

A.7. Utils

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 import os
5 import numpy as np
6 from sklearn.decomposition import PCA
7 from wordcloud import WordCloud
8
9
10 def plot_palabras_mas_frecuentes(X, vectorizer, vector_type="TF", top_n=10, guardar=True):
11     :
12
13     freq = X.toarray().sum(axis=0)
14     espacios = "oooooooooooo"
15     palabras_originales = vectorizer.get_feature_names_out()
16     palabras = [f"{espacios}{p}{espacios}" for p in palabras_originales]
```

```

16 df_freq = pd.DataFrame({'palabra': palabras, 'frecuencia': freq})
17 df_freq = df_freq.sort_values('frecuencia', ascending=False).head(top_n)
18
19 sns.set_style("whitegrid")
20 plt.figure(figsize=(16, 10))
21
22 sns.barplot(x='frecuencia', y='palabra', data=df_freq, palette='viridis')
23 plt.xlabel("Peso", fontsize=30)
24 plt.ylabel("Palabra", fontsize=30)
25
26 plt.xticks(fontsize=30)
27 plt.yticks(fontsize=30)
28
29 plt.tight_layout()
30
31 if guardar:
32     path_guardar = os.path.join("docs", "images")
33     os.makedirs(path_guardar, exist_ok=True)
34     nombre_archivo = f"{vector_type}_top{top_n}_palabras.pdf"
35     ruta_completa = os.path.join(path_guardar, nombre_archivo)
36     plt.savefig(ruta_completa, bbox_inches='tight')
37
38 plt.show()
39
40
41 def plot_wordcloud(X, vectorizer, vector_type="TF", guardar=True):
42
43     freq = X.toarray().sum(axis=0)
44     palabras_originales = vectorizer.get_feature_names_out()
45
46     # Crear diccionario palabra:frecuencia
47     palabra_freq = dict(zip(palabras_originales, freq))
48
49     # Crear wordcloud
50     wordcloud = WordCloud(width=1200, height=700, background_color='white',
51                           colormap='viridis', relative_scaling=0.5)
52     .generate_from_frequencies(palabra_freq)
53
54     plt.figure(figsize=(16, 10))
55     plt.imshow(wordcloud, interpolation='bilinear')
56     plt.axis('off')
57
58     if guardar:
59         path_guardar = os.path.join("docs", "images")
60         os.makedirs(path_guardar, exist_ok=True)
61         nombre_archivo = f"{vector_type}_wordcloud.pdf"
62         ruta_completa = os.path.join(path_guardar, nombre_archivo)
63         plt.savefig(ruta_completa, bbox_inches='tight', dpi=300)
64
65 plt.show()
66
67 def plot_matrices_confusion(resultados_eval, carpeta_salida="results", guardar=True):
68
69     os.makedirs(carpeta_salida, exist_ok=True)
70     ruta_img = os.path.join(carpeta_salida, "matrices_confusion.pdf")
71
72     metodos = list(resultados_eval.keys())
73     n = len(metodos)
74
75     # Definir figura 2x2 (sirve para 4 m todos)
76     fig, axes = plt.subplots(2, 2, figsize=(12, 10))
77     axes = axes.flatten()
78
79     for ax, metodo in zip(axes, metodos):
80         matriz = resultados_eval[metodo]["matriz_confusion"]
81
82         sns.heatmap(
83             matriz,
84             annot=False,
85             cmap="viridis",
86             ax=ax,
87             cbar=False
88         )
89         ax.set_title(f"Matriz de confusión-{metodo}")
90         ax.set_xlabel("Cluster asignado")
91         ax.set_ylabel("Cluster real")

```

```

90
91     # Ocultar ejes sobrantes si hay menos de 4 en todos
92     for i in range(len(metodos), 4):
93         fig.delaxes(axes[i])
94
95     plt.tight_layout()
96
97     if guardar:
98         path_guardar = os.path.join("docs", "images")
99         os.makedirs(path_guardar, exist_ok=True)
100        nombre_archivo = "matrices_confusion.pdf"
101        ruta_completa = os.path.join(path_guardar, nombre_archivo)
102        plt.savefig(ruta_completa, bbox_inches='tight')
103
104    plt.show()
105
106 def plot_scatter_pca(matrices, labels_list, nombres=None, titles=None, guardar=True,
107                      carpeta_salida="docs/images"):
108
109     fig, axes = plt.subplots(2, 2, figsize=(14, 12))
110     axes = axes.flatten()
111     cmap = plt.get_cmap("tab10")
112
113     for i in range(4):
114         X = matrices[i].toarray()
115         labels = np.asarray(labels_list[i])
116
117         pca = PCA(n_components=2, random_state=42)
118         coords = pca.fit_transform(X)
119         evr = pca.explained_variance_ratio_
120         evr1 = evr[0] * 100
121         evr2 = evr[1] * 100
122         evr_total = evr1 + evr2
123
124         ax = axes[i]
125         for j, lab in enumerate(np.unique(labels)):
126             ax.scatter(coords[labels == lab, 0], coords[labels == lab, 1], s=40, alpha=0
127                         .85,
128                         color=cmap(j % 10), label=f"Cluster {lab}")
129
130         ax.set_xlabel("PC1")
131         ax.set_ylabel("PC2")
132         title_base = titles[i] if titles else f"ScatterPCA Plot {i+1}"
133         title_full = f"{title_base} (PC1{evr1:.1f}, PC2{evr2:.1f}, total{evr_total:
134                         .1f})"
135         ax.set_title(title_full)
136         ax.legend(title="Clusters", loc='best', fontsize='small')
137
138         plt.tight_layout(pad=3.0)
139         if guardar:
140             os.makedirs(carpeta_salida, exist_ok=True)
141             plt.savefig(os.path.join(carpeta_salida, "scatter_4plots.pdf"), bbox_inches='
142                         tight')
143
144     plt.show()

```

A.8. Parte opcional

```

1 from sklearn.cluster import KMeans
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 def run_elbow_kmeans(X, max_k=30, titulo="M todo Elbow", guardar=False, ruta_guardado=
6     None):
7     inertias = []
8     Ks = np.arange(1, max_k + 1)
9
10    for k in Ks:
11        model = KMeans(n_clusters=k, random_state=42)
12        model.fit(X)
13        inertias.append(model.inertia_)
14
15    plt.figure(figsize=(8,5))

```

```
15 plt.plot(Ks, inertias, marker='o', linewidth=2, markersize=8, color="#20a386")
16 plt.xticks(Ks)
17 plt.grid(alpha=0.3, linestyle="--")
18
19 plt.title(titulo, fontsize=14, fontweight="bold")
20 plt.xlabel("Número de clusters (k)", fontsize=12)
21 plt.ylabel("Inercia (SSE)", fontsize=12)
22
23 if guardar:
24     plt.savefig(ruta_guardado, bbox_inches="tight")
25     print(f"Gráfico Elbow guardado en: {ruta_guardado}")
26
27 plt.show()
28
29 return Ks, inertias
```