Escuela Superior de Cómputo

Reporte

Práctica Integradora

Realizada por

Robles Guzmán Naomi Isabel

Ugalde Téllez Aarón

Para la materia de

Tecnologías de Lenguaje Natural

Impartida por

Elizabeth Moreno Galván

Grupo

5BM1

---

27 de junio de 2025

Índice

1. Introducción

El análisis automatizado permite extraer información valiosa sobre tendencias, sentimientos y patrones de información que circulan en los medios de comunicación [1]. La capacidad de clasificar automáticamente el tipo de contenido y determinar su polaridad emocional podría significar una herramienta estratégica para organizaciones o investigadores.

Este proyecto integrador aborda dos problemáticas centrales en el procesamiento de lenguaje, la clasificación automática de tipos de noticias y el análisis de sentimientos. Para la primera problemática, se implementa un sistema de clasificación supervisada basado en el algoritmo K-Nearest Neighbors (KNN), el cual permite categorizar automáticamente las noticias según su temática específica. Para la segunda problemática, dado a que los documentos no están etiquetados según su tono emocional, se desarrolla un sistema de análisis de sentimientos que combina enfoques léxicos mediante diccionarios especializados y recursos como SenticLex, permitiendo identificar la polaridad emocional de los textos.

La metodología propuesta incluye la etapa de preprocesamiento de datos que contempla la limpieza, normalización y tokenización de textos, así como la implementación de técnicas de balanceo de clases para optimizar el rendimiento de los algoritmos de clasificación. Adicionalmente, se emplea clustering no supervisado mediante K-means para agrupar los textos según sus características sentimentales, proporcionando una perspectiva complementaria al análisis de polaridad.

Retomando el dataset utilizado a lo largo de todo el curso, el cual comprende noticias en español etiquetadas por categorías temáticas, esta característica será clave para evaluar la efectividad de las técnicas de clasificación supervisada. Los resultados esperados buscan demostrar la viabilidad y efectividad de las técnicas implementadas, estableciendo las bases para futuras investigaciones en el campo del procesamiento automatizado de textos.

1.1. Definición de la Problemática

En la actualidad cada día circula una gran cantidad de noticias, por tal motivo, medios de comunicación, empresas y centros de investigación enfrentan el reto de procesar y entender grandes volúmenes de contenido textual de forma rápida y eficiente. Este reto se manifiesta principalmente en tres áreas:

1.1.1. Clasificación de contenido informativo

Clasificar textos como noticias en ocasiones presenta limitaciones como poca escalabilidad, criterios inconsistentes y procesamiento demasiado lento.

1.1.2. Detección de tono y carga emocional

Para determinar si un texto transmite una emoción positiva, negativa o neutra se deben superar desafíos como el sarcasmo y la ironía, así como también la variación cultural y lingüística.

1.1.3. Desequilibrio en los datos

El último gran reto tiene que ver con la distribución de las categorías en un conjunto de datos, frecuentemente suele ser desigual y este caso no es la excepción. El *dataset* en el que está basado este trabajo presenta una tendencia hacia clases dominantes, lo que genera problemas para generalizar y por consecuencia, peor desempeño en clases menos representadas.

1.2. Propuesta de solución

Se propone desarrollar un sistema que automatice el análisis de texto, con dos componentes principales: la clasificación temática y el análisis de sentimientos.

1.2.1. Estrategia de preprocesamiento

Se implementa una serie de pasos para limpiar y preparar los textos antes del análisis:

- **Normalización del texto:** Eliminación de signos de puntuación, uso uniforme de minúsculas y corrección de espacios.
- **Filtrado léxico:** Solo se conservan las palabras que aparecen en un diccionario en español.
- **Lematización:** Reducción de las palabras a su forma base.
- **Eliminación de palabras vacías:** Se eliminan conectores, artículos y otras palabras poco relevantes para el análisis.

1.2.2. Técnicas para balancear los datos

Para corregir el desbalance entre categorías, se aplica una estrategia mixta:

- **Reducción de clases dominantes** Se usa el método Near Miss para reducir de forma controlada los ejemplos de las categorías más representadas, seleccionando solo los ejemplos más relevantes para conservar diversidad.
- **Refuerzo de clases minoritarias**
  1. Se generan nuevos ejemplos mediante Word2Vec, intercambiando palabras por otras similares.
  2. Se complementa con SMOTE para crear representaciones artificiales en el espacio vectorial.

1.2.3. Sistema de análisis de sentimientos por capas

- **Capa básica** Diccionarios propios para identificar palabras positivas, negativas o relacionadas con emociones como alegría, tristeza, enojo y miedo.
- **Capa especializada** Se usa SenticLex para calcular un puntaje continuo que refleje la polaridad emocional del texto, obteniendo métricas agregadas para cada documento del corpus.
- **Capa de agrupamiento** Los documentos se agrupan automáticamente según sus características emocionales, identificando patrones usando el algoritmo K-means.

2. Desarrollo

2.1. Descripción del dataset

El conjunto de datos proporcionado, puede consultarse en el sition *Kaggle* y fue construido con una herramienta de web scraping para el Dataton 2022 de Bancolombia para el entrenamiento de modelos supervisados a utilizar en una recomendación de Noticias [2].

El *dataset* tiene noticias de las categorías:

- Macroeconomía
- Sostenibilidad
- Innovación
- Regulaciones
- Alianzas
- Reputación
- Otra

Además,el documento *CSV* consta de 1217 registros y las columnas:

- **Noticia:** texto de la noticia.
- **Tipo:** Etiqueta del tipo de noticia

La distribución de instancias de noticias por categoría se aprecia en la siguiente Figura. Se observa que las clases Macroeconomía y Alianzas tienen un mayor número de muestras; mientras que el resto de clases, exceptuando reputación, oscilan entre las 100 y 200 instancias. Por su parte, Reputación es la categoría menos representada del *dataset*.

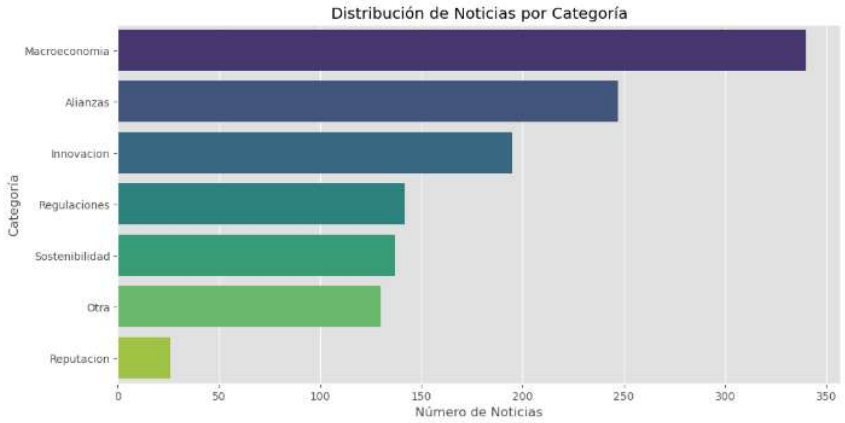

Figura 1: Gráfica de la distribución de registros de noticias por tipo.

Después de analizar el *dataset* con ayuda de Python, se encontró que la longitud promedio de palabras es de 516.55, siendo la noticia más larga de un total de 2964 palabras, y la más corta solo contenía una. En la **Figura 2** se muestra la distribución del número de palabras por noticia en cada una de las siete categorías del conjunto de datos.

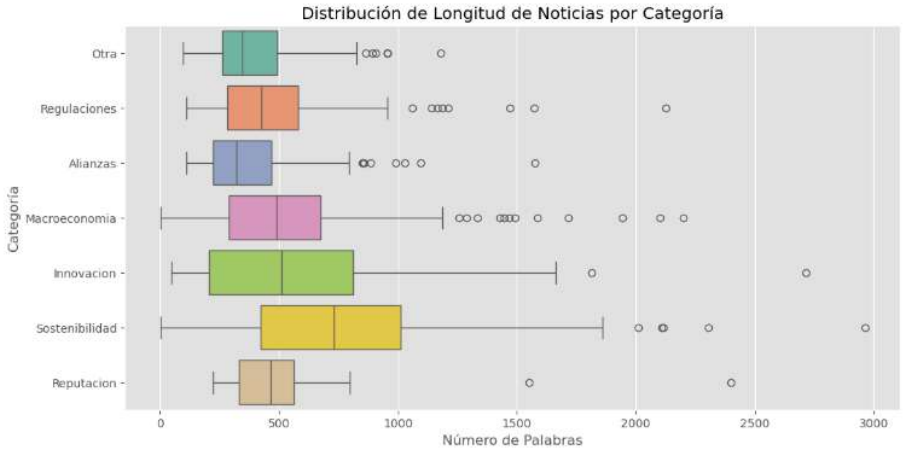

Figura 2: Gráfica de la distribución de la longitud en palabras de las noticias por tipo.

Se observa que las noticias relacionadas con Sostenibilidad e Innovación tienden a ser más extensas, mientras que las de Reputación y Alianzas son más cortas. Además, se identifican múltiples valores atípicos, lo que indica que algunas noticias presentan una longitud significativamente mayor al promedio de su categoría. Esta variabilidad puede influir en el procesamiento y análisis posterior de los textos.

Por otra parte, para conocer las palabras más frecuentes en cada categoría de noticia se hizo uso de la herramienta de *WordCloud*, debido a que permite una fácil visualización de los términos más usados.

Figura 3: Visualización de 3 de las 7 nubes de palabras generadas para cada categoría. El resto puede consultarse en la carpeta del proyecto.

Las nubes de palabras de las clases reflejan que, aunque cada categoría aborda un enfoque temático distinto, todas comparten un campo semántico relacionado con el ámbito empresarial, financiero y económico. Términos como empresa, cliente, mercado y BBVA aparecen de forma reiterada, lo que

indica una base léxica común en el corpus, posiblemente influenciada por el protagonismo de una entidad financiera específica. Sin embargo, también se observan diferencias claras entre categorías: Macroeconomía incluye vocabulario técnico relacionado con indicadores económicos; Reputación se enfoca en términos vinculados a la imagen y posicionamiento de marca; Innovación destaca palabras asociadas a tecnología y servicios digitales, mientras que Sostenibilidad introduce términos del ámbito ambiental. Esto sugiere que, si bien hay una estructura temática general compartida, el contenido de las noticias sí varía según la categoría, característica que permitirá distinguir matices y lograr un buen rendimiento para el clasificador.

2.2. Preprocesamiento de los datos

Se diseñó una clase `class Preprocesa:` para realizar diversas operaciones de limpieza y normalización de textos en español. Apoyándose de la biblioteca string de Python para la lista de caracteres especiales que se eliminan del texto. Los métodos definidos por esta clase se aplican sobre el dataset mediante la siguiente linea de código.

```
preprocesador = Preprocesa()

dataset["news_limpio"] = dataset["news"].astype(str).apply(preprocesador.
    remove_stopwords).apply(preprocesador.remove_punctuation).apply(preprocesador.
    lower_words)

# Guardar el dataset procesado
dataset.to_csv("news.csv", index=False)
```

Ademas, para eliminar palabras no deseadas, se compararon las existentes con un diccionario en español y se eliminaron las que no se encontraran en este.

2.2.1. Clase preprocesa

```
def remove_punctuation(self, text):
```

Esta función dentro de la clase Preprocesa recibe un texto y define un conjunto de caracteres prohibidos que incluye signos de puntuación, caracteres especiales y secuencias no deseadas. Dicho conjunto se une al conjunto de signos de puntuación provisto por `string.punctuation`, para generar una nueva cadena que excluye dichos caracteres y finalmente estandarizar los espacios en blanco, de manera que se retorne un texto libre de puntuaciones [3].

```
def lower_words(self,text):
```

Este sencillo método se encarga de convertir todos los caracteres de la cadena minúsculas, esto para facilitar la comparación entre tokens y palabras vacías.

```
def quitarAcentos(self,s):
```

Esta función reemplaza los caracteres acentuados por su contraparte sin acento. Este proceso garantiza la uniformidad de las palabras, ya que elimina variaciones ortográficas que podrían afectar la tokenización y posteriores análisis. Si bien esta función puede ser útl en algunos cntextos, para esta práctica no se aplicó al *dataset*.

```
def remove_stopwords(self,text):
```

Dentro de este método se define una lista de términos considerados poco informativos, tales como

conectores, artículos o preposiciones comunes, a continuación, cada palabra del texto se compara con dicha lista, y las que coinciden son eliminadas. Como resultado, el texto conserva únicamente aquellas palabras con mayor potencial para el análisis semántico.

2.2.2. Tokenización, Lematización y comparación de los lemas con un diccionario en español

Se importó un diccionario en español para la comparación con los lemmas de los documentos.

```
with open("recursos/diccionario_limpio.txt", "r", encoding="utf-8") as f:
valid_words = set(word.strip().lower() for word in f.readlines())
```

Una vez leído el diccionario, se iteró sobre los documentos y se tokenizó con spacy cada uno de ellos, después de haber tokenizado los documentos, se obtuvieron los lemmas del mismo y se verificó su existencia con el diccionario previamente leído.

```
lemmas_por_documento = []
for text in dataset["news_limpio"]:
    doc = nlp(text)
    lemmas = [
        token.lemma_ for token in doc
        if not token.is_punct and not token.is_space and token.lemma_.lower() in
    valid_words
    ]
    lemmas_por_documento.append(lemmas)
```

2.3. Clasificador de tipos de noticias

Dado que nuestro dataset está etiquetado respecto al tipo de noticia, se optó por un enfoque de aprendizaje supervisado, pues se espera que instancias con características similares compartan la misma clase. KNN es un buen candidato al basarse en la vecindad en el espacio de características.

Por otro lado, dada la sensibilidad de KNN al imbalance de clases, se optó por un enfoque híbrido para balancear las clases; generando documentos similares e intercambiando ciertas palabras de las noticias por otras con un significado similar. Además, una vez vectorizados nuestros documentos, se agregaron muestras en el espacio vectorial mediante la técnica **SMOTE** [4]. Además, realizamos **Undersampling** sobre las clases con más registros para tener un balance en todas las clases.

2.3.1. Balanceo de clases y vectorización

Para empezar, se definió un número de registros por clase, para determinar el numero de clases que se deben generar o cortar por clase. Para este proyecto, se definió un número de 200 registros por muestra. Una vez definido el número de registros deseados, se realizó undersampling sobre las clases que contienen más registros de los esperados, en este caso sobre las clases Macroeconomía y Alianzas.

```
clases_undersampling = ['Macroeconomia', 'Alianzas']
X_undersampled, y_undersampled = aplicar_undersampling(
    documentos_originales,
    etiquetas_originales,
    clases_undersampling,
    200,
    modelo_doc2vec
)
```

Para realizar Undersampling sobre las clases minoritarias, se hizo uso de la librería proporcionada por **Imbalancead Learn** llamada **Near Miss** en su primera versión, que básicamente elimina las muestras mas cercanas a la clase minoritaria [5].

```python
def aplicar_undersampling(documentos: List[List[str]], etiquetas: List[str],
                          clases_objetivo: List[str], n_muestras: int,
                          modelo: Doc2Vec) -> Tuple[np.ndarray, List[str]]:

    print(f"Aplicando undersampling a clases: {clases_objetivo}")

    # Filtrar documentos de las clases objetivo
    indices_objetivo = [i for i, etiq in enumerate(etiquetas) if etiq in
    clases_objetivo]
    docs_filtrados = [documentos[i] for i in indices_objetivo]
    etiquetas_filtradas = [etiquetas[i] for i in indices_objetivo]

    # Vectorizar documentos
    X = np.array([modelo.infer_vector(doc) for doc in docs_filtrados])

    # Configurar estrategia de sampling
    sampling_strategy = {clase: n_muestras for clase in clases_objetivo}

    # Aplicar Near Miss
    nm = NearMiss(version=1, sampling_strategy=sampling_strategy)
    X_resampled, y_resampled = nm.fit_resample(X, etiquetas_filtradas)

    # Convertir a lista si es necesario
    if hasattr(y_resampled, 'tolist'):
        y_resampled = y_resampled.tolist()

    return X_resampled, y_resampled
```

Posteriormente, se realizó Oversampling sobre las clases minoritarias para alcanzar el número de registros deseados, en este caso se llevó a cabo este proceso sobre las clases Regulaciones, Sostenibilidad, Otra y Reputación.

```python
docs_por_clase = separar_documentos_por_clase(documentos_originales,
    etiquetas_originales)

clases_oversampling = ['Regulaciones', 'Sostenibilidad', 'Otra', 'Reputacion']

resultados_oversampling = {}
for clase in clases_oversampling:
    if clase in docs_por_clase:
        vectores_balanceados = aplicar_oversampling_hibrido(
            docs_por_clase[clase],
            clase,
            200,
            modelo_doc2vec
        )
        resultados_oversampling[clase] = vectores_balanceados
    else:
        print(f"Advertencia: Clase '{clase}' no encontrada en los datos")
```

Para aplicar el Oversampling se optó por un enfoque híbrido donde se genero la mitad de los registros faltantes mediante el remplazo de palabras por otras que estén cerca del espacio vectorial de la original y

la otra mitad mediante SMOTE. Esta técnica genera nuevas instancias sintéticas de la clase minoritaria. Mediante la identificación de cada instancia minoritaria, para después buscar sus vecinos más cercanos (generalmente cinco) dentro de la misma clase y creando nuevos puntos interpolando aleatoriamente entre una instancia y alguno de sus vecinos [4].

```
def aplicar_oversampling_hibrido(documentos: List[List[str]], etiqueta: str,
                                 n_muestras_objetivo: int, modelo_doc2vec: Doc2Vec) ->
    np.ndarray:

    if len(documentos) >= n_muestras_objetivo:
        return np.array([modelo_doc2vec.infer_vector(doc) for doc in documentos[:
    n_muestras_objetivo]])

    muestras_faltantes = n_muestras_objetivo - len(documentos)
    muestras_word2vec = muestras_faltantes // 2

    # Se aplica el 1ER M todo de Oversampling
    muestras_nuevas = generar_muestras_word2vec(documentos, muestras_word2vec)

    documentos_combinados = documentos + muestras_nuevas

    X_vectorizado = np.array([modelo_doc2vec.infer_vector(doc) for doc in
    documentos_combinados])
    y_dummy = np.array([0] * len(documentos) + [1] * len(muestras_nuevas))

    # Se aplica el 2DO M todo de Oversampling
    muestras_smote = n_muestras_objetivo - len(X_vectorizado)
    if muestras_smote > 0:
        k_neighbors = min(5, len(X_vectorizado) - 1)
        smote = SMOTE(sampling_strategy={1: len(muestras_nuevas) + muestras_smote},
                      k_neighbors=k_neighbors, random_state=42)
        X_resampled, _ = smote.fit_resample(X_vectorizado, y_dummy)
        X_final = X_resampled[:n_muestras_objetivo]
    else:
        X_final = X_vectorizado


    return X_final
```

En el código anterior se utiliza la función `generar_muestras_word2vec()` cuya definición se presenta a continuación, en esta se usa word2vec para obtener los vectores de las palabras y poder generar términos similares en el espacio vectorial, con una probabilidad de 0.3 de generar nuevas palbras.

```
def generar_muestras_word2vec(documentos: List[List[str]], n_muestras: int) -> List[
    List[str]]:
    if n_muestras <= 0:
        return []
    print(f"Generando {n_muestras} muestras con Word2Vec...")

    modelo_w2v = Word2Vec(documentos, window=10, min_count=1, sg=0, workers=4)

    nuevas_muestras = []

    for _ in range(n_muestras):
        doc_base = documentos[np.random.randint(0, len(documentos))]
        nueva_muestra = []
        for palabra in doc_base:
            if palabra in modelo_w2v.wv and np.random.random() < 0.3:
```

```
                try:
                    similares = modelo_w2v.wv.most_similar(palabra, topn=3)
                    if similares:
                        nueva_palabra = similares[np.random.randint(0, len(similares)
    )][0]
                        nueva_muestra.append(nueva_palabra)
                    else:
                        nueva_muestra.append(palabra)
                except:
                    nueva_muestra.append(palabra)
            else:
                nueva_muestra.append(palabra)
        nuevas_muestras.append(nueva_muestra)
    return nuevas_muestras
```

2.3.2. Entrenamiento del modelo de clasificación KNN

Para determinar el valor de k que optimiza las métricas, se evaluaron valores de k entre 0 y 80, entrenando un modelo KNN con cada uno sobre el conjunto de datos previamente balanceado. Una vez identificado el k óptimo, se entrenó de nuevo el modelo con ese parámetro y se obtuvieron sus métricas junto con la matriz de confusión.

```
k=44
k_nn_model = KNeighborsClassifier(n_neighbors=k)

# Entrenamiento del modelo
k_nn_model.fit(X_train, y_train)
y_pred = k_nn_model.predict(X_test)

# Calculo de metricas y matriz de confusion
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
roc_auc = roc_auc_score(y_test, k_nn_model.predict_proba(X_test), multi_class='ovr')
conf_matrix = confusion_matrix(y_test, y_pred)
```

El resultado de las métricas se puede consultar en la sección de resultados, donde además se hará una comparación entre la eficiencia del clasificador con clases balanceadas y sin balancear.

2.4. Análisis de sentimientos

En esta sección del proyecto se implementa un sistema de análisis de sentimientos en español que combina dos principales enfoques léxicos:

1. **Creación de Covariables Propias:** Se hizo uso de archivos de texto tipo diccionarios que contenían palabras que fueron clasificadas como positivas y negativas. Asimismo se manejaron listas en Python con palabras que podrían indicar un sentimiento de alegría, tristeza, enojo o miedo en una noticia. Estos archivos pueden ser consultados dentro del Drive del proyecto, en la carpeta de recursos.

2. **Puntuación con léxico espacializado:** Por otra parte se hizo uso de una base de datos que asocia palabras o expresiones con valores emocionales (como positividad, negatividad, neutralidad). En este enfoque se asocia a unidades léxicas (lemas) valores de polaridad afectiva en un

intervalo continuo [-1, +1], donde +1 indica máxima positividad, -1 máxima negatividad y 0 neutralidad [6].

2.4.1. Covariables

Se desarrolló la función `extraer_covariables_lemas()` que procesa las listas de lemas por documento previamente obtenidas y calcula métricas sentimentales basadas en diccionarios de palabras positivas, negativas y específicas de emociones (alegría, tristeza, enojo, miedo). Genera tanto conteos absolutos como proporciones relativas, además de frecuencias para palabras clave específicas.

```python
def extraer_covariables_lemas(lista_lemas):

    tokens = [token.lower() for token in lista_lemas]

    num_pos = sum(token in palabras_positivas for token in tokens)
    num_neg = sum(token in palabras_negativas for token in tokens)

    total = len(tokens) if len(tokens) > 0 else 1  # evitar divisi n por 0

    # palabras clave espec ficas
    frecuencia_claves = {}
    for palabra in palabras_clave:
        frecuencia_claves[f"freq_{palabra}"] = tokens.count(palabra)

    num_alegria = sum(1 for token in tokens if token in palabras_alegria)
    num_tristeza = sum(1 for token in tokens if token in palabras_tristeza)
    num_enojo = sum(1 for token in tokens if token in palabras_enojo)
    num_miedo = sum(1 for token in tokens if token in palabras_miedo)

    covariables = {
        "num_palabras_positivas": num_pos,
        "num_palabras_negativas": num_neg,
          ... se regresan todas las listas obtenidas.
    }

    # Agregar frecuencias de palabras clave
    covariables.update(frecuencia_claves)

    return covariables
```

Las funciones `score_sentiment()` y `score_of_docs()` utilizan el recurso SentiLex (archivo XML senticon.es.xml) para asignar puntuaciones numéricas de polaridad a cada documento, calculando el promedio de las puntuaciones de los lemas que aparecen en el léxico.

El archivo XML presenta una estructura jerárquica donde:

- El elemento raíz `<senticon>` especifica el idioma ("es" para español)
- Contiene capas (`<layer>`) que organizan los términos por niveles de análisis
- Cada capa agrupa lemas en categorías `<positive>` y `<negative>`.
- Cada `<lemma>` contiene texto del lema, y atributos:
  - **pos:** categoría gramatical (a=adjetivo, n=sustantivo)
  - **pol:** valor de polaridad numérico (ej. 0.708)

- **std:** desviación estándar de la anotación (medida de acuerdo inter-anotadores)

```xml
<senticon lang="es">
    <layer level="1">
        <positive>
            <lemma pos="a" pol="0.708" std="0.149"> acertado </lemma>
            <lemma pos="a" pol="0.906" std="0.125"> admirable </lemma>
            <lemma pos="n" pol="0.45" std="0.331"> admiración </lemma>
            <lemma pos="v" pol="0.75" std="0.177"> admirar </lemma>
        </positive>
        <negative>
            <lemma pos="v" pol="-0.554" std="0.047"> abatir </lemma>
            <lemma pos="a" pol="-0.25" std="0.0"> abochornado </lemma>
            <lemma pos="v" pol="-0.5" std="0.072"> abochornar </lemma>
            <lemma pos="n" pol="-0.425" std="0.056"> abominación </lemma>
            <lemma pos="v" pol="-0.625" std="0.088"> aborrecer </lemma>
        </negative>
    </layer>
    <layer level="2">
        <positive>
        ...
</senticon>
```

Figura 4: Ejemplo del contenido del archivo senticon.xml que se puede consultar dentro de la carpeta **recursos** del proyecto.

Para empezar a hacer uso del archivo **senticon.xml** en el código de Python, primero se lee un archivo con listas de lemas por documento y se parsea el XML de SenticLex usando `ET.parse()`.

```
        with open('archivos/lemas_por_documento.txt', 'r', encoding='utf-8') as f:
        listas_de_lemmas = [
            ast.literal_eval(line.strip())
            for line in f
            if line.strip()  # descarta l neas vac as
        ]

    tree = ET.parse("recursos/senticon.es.xml")
    root = tree.getroot()
```

Posteriormente se construye el diccionario, haciendo un mapeo sentilex donde se identifica la clave (string del lema) y el valor de la polaridad numérica (ej. 0.906). Solo se almacena la primera ocurrencia de cada lema, esto para priorizar capas superiores.

```
    sentilex = {}
    for layer in root.findall("layer"):
        for sign in ("positive", "negative"):
            for lemma in layer.find(sign).findall("lemma"):
                text = lemma.text.strip()
                pol  = float(lemma.get("pol"))

                if text not in sentilex:
                    sentilex[text] = pol
```

A continuación se manda a llamar a `score_of_docs()` para aplicar `score_sentiment` a cada documento.

```
    scores = score_of_docs(listas_de_lemmas)
```

La función `score_sentiment` calcula el sentimiento promedio de un texto mediante la búsqueda de cada lema en sentilex y la agregación de los valores encontrados. También normalizando por cantidad de términos encontrados.

```
    def score_sentiment(text):
        scores = []
        for lemma in text:
            if lemma in sentilex:
                scores.append(sentilex[lemma])

        return sum(scores) / len(scores)
```

Tras aplicar los cambios al conjunto de documentos lematizados, se guardan las covariables en un nuevo archivo *csv* que se usará para identificar las agrupaciones de los datos basandóse en los sentimientos de las noticias, para esto se usará k-means.

2.4.2. Entrenamiento de modelo de K-means

Se carga el archivo con las covariables y se eligen las columnas a considerar para el entrenamiento del modelo.

```
    dataframe = pd.read_csv('archivos/sentiment_scores_combined.csv')

#Obt n las caracteristicas de inter s
feature_columns = [
    'scores',
    'proporcion_palabras_positivas',
    'proporcion_palabras_negativas',
    'sentimiento_lexico_total',
    'proporcion_alegria',
    'proporcion_tristeza',
    'proporcion_enojo',
    'proporcion_miedo'
]

# Preparar los datos
X = dataframe[feature_columns].fillna(0)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

El siguiente paso consiste en implementar el algoritmo de clustering K-Means con k=3, debido a los grupos predefinidos (correspondientes a sentimientos positivo, negativo y neutro). Utilizando los datos estandarizados, el algoritmo agrupa automáticamente las noticias según la similitud de sus características sentimentales.

```
n_clusters = 3
kmeans = KMeans(n_clusters = n_clusters, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)
```

Una vez entrenado K-Means se calculan métricas de calidad (Silhouette Score e inercia) para evaluar la cohesión y separación de los grupos formados. Los centroides de cada cluster se analizan para

asignar etiquetas semánticas, y se genera una representación tridimensional de los clusters que se puede visualizar en la sección de resultados.

```
silhouette_avg = silhouette_score(X_scaled, cluster_labels)
inertia = kmeans.inertia_

# Analizar centroides para interpretar clusters
centroids = kmeans.cluster_centers_
centroids_df = pd.DataFrame(centroids, columns=feature_columns)

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Elegir tres caracteristicas para la visualizacion
x = X_scaled[:, feature_columns.index('scores')]
y = X_scaled[:, feature_columns.index('proporcion_palabras_positivas')]
z = X_scaled[:, feature_columns.index('proporcion_palabras_negativas')]

plt.show()
```

Por último, las clases asignadas por K-Means a cada instancia, fueron mapeadas como positiva, negativa o neutra y agregadas como columna al dataset original.

proporcion_palabras_negativas	sentimiento_lexico_total	num_palabras_alegria	num_palabras_tristeza	num_palabras_enojo	num_palabras_miedo	proporcion_alegria	proporcion_tristeza	proporcion_enojo	proporcion_miedo	cluster	sentiment_label
0.0	2	0	0	0	0	0.0	0.0	0.0	0.0	1	Negativa
0.00625	-1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	1	1	0	0	0	0.0064516129032258	0.0	0.0	0.0	1	Negativa
0.0	2	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0028328611898017	3	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	3	0	0	0	0	0.0	0.0	0.0	0.0	1	Negativa
0.0060060060060060	2	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	0	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	0	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	12	0	0	0	0	0.0	0.0	0.0	0.0	1	Negativa
0.0030487804878048	-1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	2	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	2	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	0	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	0	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	5	0	0	0	0	0.0	0.0	0.0	0.0	1	Negativa
0.0	0	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	1	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0	5	0	0	0	0	0.0	0.0	0.0	0.0	1	Negativa
0.0	6	0	0	0	1	0.0	0.0	0.0	0.0024390243902439	1	Negativa
0.0027397260273972	4	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva
0.0042194092827004	2	0	0	0	1	0.0	0.0	0.0	0.0021097046413502	0	Positiva
0.0048309178743961	2	0	0	0	0	0.0	0.0	0.0	0.0	0	Positiva

Figura 5: Fragmento del dataset que contiene las covariables y el resultado de la Clusterización.

3. Resultados

3.1. Clasificador de tipos de noticias

3.1.1. Métricas de rendimiento

En el siguiente cuadro se muestra la comparación del rendimiento del clasificador KNN medido con las métricas de *Accuracy*, *Precision*, *Recall* y *F1-Score*. La comparación entre las métricas con y sin balanceo de clases revela mejoras significativas en todos los indicadores de rendimiento

Métricas	**Métricas con balanceo de clases**	**Métricas sin balanceo de clases**
Accuracy	0.925	0.7868
Precision	0.9261	0.8213
Recall	0.925	0.7868
F1-Score	0.9251	0.7837

Cuadro 1: Comparación de métricas entre diferentes métodos de vectorización

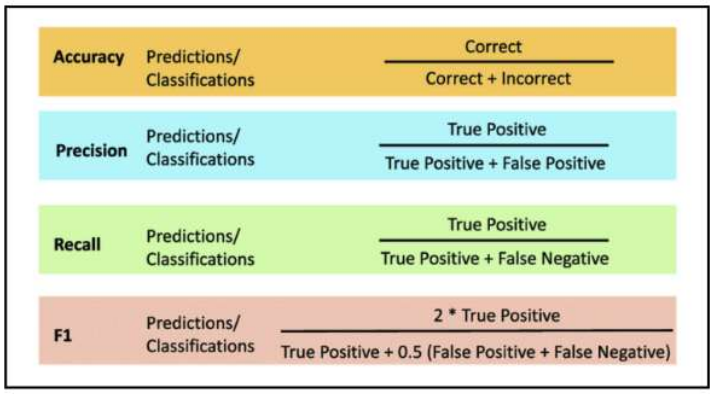

Figura 6: Significado de las métricas de rendimiento calculadas [7]

Se hace evidente que cuando se trabaja sin balanceo de clases, el rendimiento se deteriora significativamente. La diferencia de aproximadamente 0.15 puntos en todas las métricas revela que el desbalance no solo afecta la accuracy general, sino que también compromete tanto la capacidad del modelo para identificar correctamente los casos positivos (recall) como su precisión al hacer predicciones positivas. Este patrón indica que el modelo aprende a predecir principalmente la clase mayoritaria para maximizar la accuracy, pero perdiendo capacidad discriminativa real entre las clases.

3.1.2. Matriz de confusión

Al evaluar el rendimiento con datos balanceados (representado en la Figura 7), la matriz de confusión muestra que los valores más altos se concentran en la diagonal principal, lo que indica una alta tasa de clasificaciones correctas.

También es visible la clasificación perfecta en múltiples clases, como la clase 4 y la clase 5, que muestran 35 y 37 instancias correctamente clasificadas, respectivamente. Los errores de clasificación son mínimos y se distribuyen de manera dispersa, con la mayor confusión entre las clases 0 y 2, donde se

observa un solapamiento de tres instancias, lo cual puede deberse a cierta similitud semántica entre esas categorías.

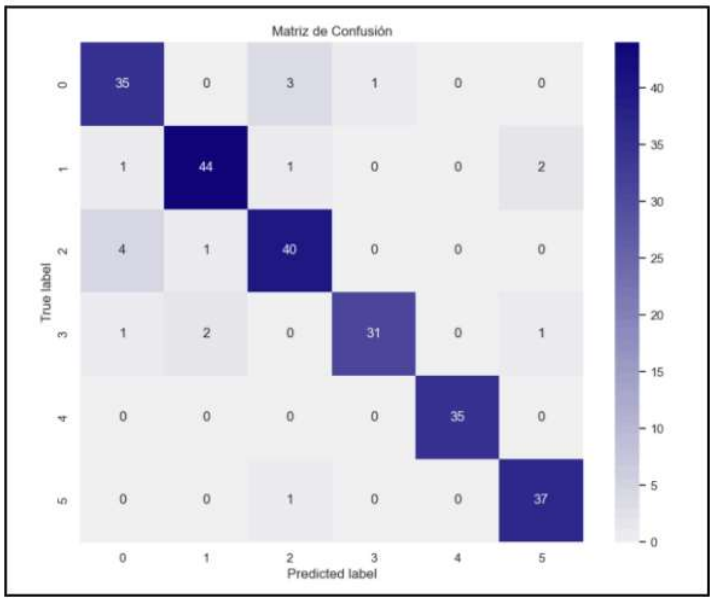

Figura 7: Matriz de confusión con los datos previamente balanceados

En contraste, al observar la matriz de confusión obtenida con datos no balanceados (Figura 8), se evidencian las limitaciones del modelo debido al sesgo hacia las clases mayoritarias. Las clases 0 y 2 presentan un número desproporcionadamente alto de instancias, con 47 y 59, respectivamente, mientras que las clases 4 y 5 están significativamente subrepresentadas, con solo 15 y 3 instancias.

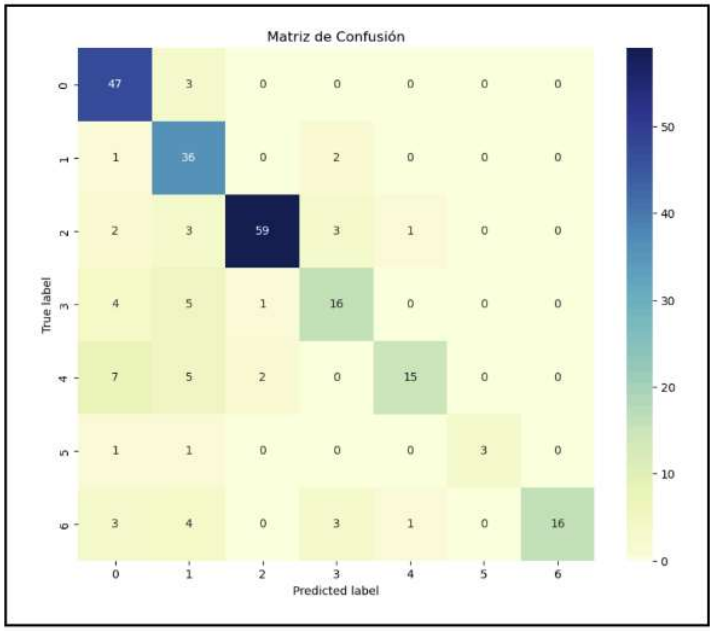

Figura 8: Matriz de confusión con los datos sin balancear

Estos resultados validan la hipótesis de que el desbalance de clases constituía una limitación significativa en el rendimiento del modelo, y que la estrategia híbrida de balanceo que combinó técnicas de undersampling como Near Miss con oversampling mediante Word2Vec y SMOTE logró mitigar efectivamente este problema.

3.1.3. Distribución de Clusters

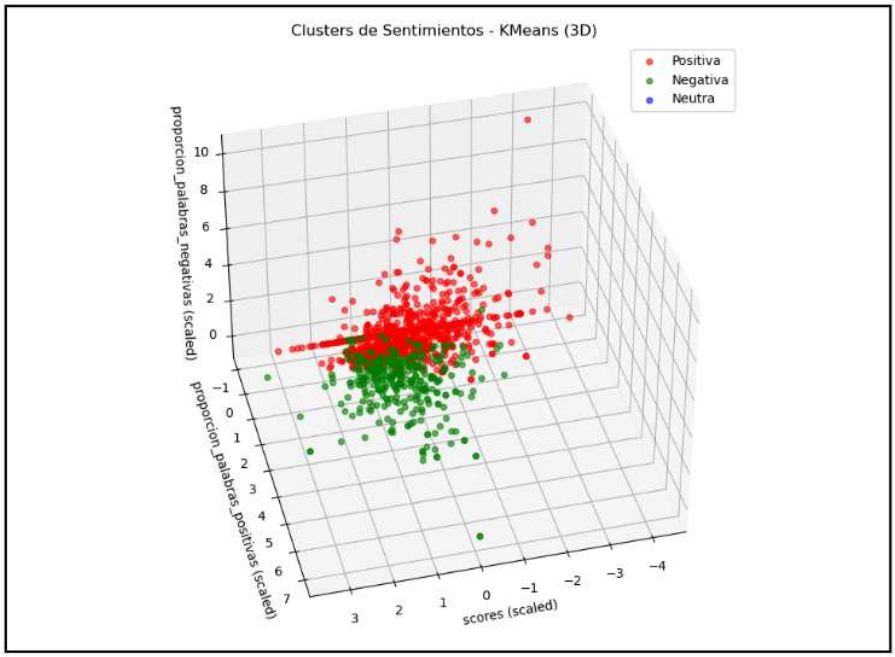

Figura 9: Gráfica 3D del agrupamiento de los datos, tomando 3 características (proporción palabras positivas, proporción palabras negativas y scores)

La visualización tridimensional de los clusters de sentimientos, presentada en la Figura 8, revela que las covariables no lograron distinguir de manera óptima las tres clases que se pretendía identificar.

```
--- MÉTRICAS DE CLUSTERING ---

Silhouette Score: 0.2985
  → Evalúa qué tan bien definidos están los clusters.
  → Valores cercanos a 1 indican una buena separación entre clusters.
  → Valores cercanos a 0 indican clusters superpuestos o mal definidos.

Inercia (Suma de errores cuadráticos intra-cluster): 7251.23
  → Mide la compacidad de los clusters.
  → Es la suma de las distancias cuadradas de cada punto a su centroide.
  → Cuanto menor sea la inercia, más compactos son los clusters.
```

Figura 10: Resultados del rendimiento de K-Means.

El Silhouette Score de 0.2985 indica que los clusters están algo definidos, pero existe cierto solapamiento entre ellos o una separación poco clara. Por otro lado, la inercia de 7251.23 sugiere un nivel razonable de cohesión interna.

4. Conclusiones

En esta práctica integradora se comenzó por realizar un preprocesamiento exhaustivo, pasando por la eliminación de puntuación, la normalización y la depuración de *stopwords*. Este proceso en conjunto con la verificación de lemas frente a un diccionario en español, permitió obtener representaciones limpias y coherentes que demostraron la importancia del preprocesamiento para garantizar la calidad del texto. Gracias a este paso se obtuvo una base sólida para las etapas posteriores.

El balanceo de clases, conseguido mediante una combinación de *undersampling* con Near Miss y un enfoque híbrido de *oversampling* que incorpora generación de muestras con Word2Vec y SMOTE, resultó decisivo para reducir el sesgo hacia las categorías mayoritarias, problema que fue evidente al analizar la matriz de confusión que generaba el clasificador KNN después de entrenarlo con el *dataset* sin balancear. Gracias a este paso, el clasificador KNN mejoró su rendimiento de manera notable: la exactitud pasó de 0.7868 a 0.925 y el *F1-score* de 0.7837 a 0.9251.

Por otro lado, la implementación del análisis de sentimientos basado en diccionarios propios y en el recurso SenticLex permitió asignar puntuaciones continuas en el intervalo $[-1, +1]$ y construir covariables que capturan la polaridad afectiva de cada texto. Estas características sentimentales fueron lo suficientemente discriminativas como para agrupar, mediante K-Means con $k = 3$, las noticias en clústeres medianamente identificables como positivo, negativo y neutro. No obstante, el resultado de las métricas de *silhouette score* e inercia demostraron que el modelo tiene áreas de mejora.

Los resultados en el análisis de sentimientos son los esperados, pues no se apiraba a una gran separación de los clusters debido a la naturaleza del dataset, que está conformada de noticias neutras con temáticas financieras.

En conjunto, este trabajo sienta las bases para mejoras en futuros trabajos relacionados con el procesamiento de lenguaje natural, las técnicas utilizadas demostraron que con una implementación diligente, y experimentación constante, se puede alcanzar altos niveles de rendimiento en tareas de clasificación y agrupamiento de textos.

Referencias

[1] S. J. Ahmed. "Natural Language Processing in News Classification: Unleashing the Power of AI in Media." (21 de jul. de 2023), dirección: https://medium.com/dataduniya/natural-language-processing-in-news-classification-unleashing-the-power-of-ai-in-media-34afc60c58eb (visitado 21-07-2023).

[2] Kaggle, *Spanish news classification*, ver. 1.0, Online; accessed 26-November-2022, Kaggle, nov. de 2022. dirección: https://www.kaggle.com/datasets/kevinmorgado/spanish-news-classification.

[3] S. Vijayarani y M. J. Ilamathi, "Preprocessing techniques for text mining - An overview," *International Journal of Computer Science & Communication Networks*, vol. 5, n.º 1, 2013. dirección: https://www.researchgate.net/profile/Vijayarani-Mohan/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview/links/5e57a0f7299bf1bdb83e7505/Preprocessing-Techniques-for-Text-Mining-An-Overview.pdf.

[4] imbalanced-learn. "SMOTE — Version 0.13.0." (), dirección: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html (visitado 25-06-2025).

[5] imbalanced-learn. "3. Under-sampling — Version 0.13.0." (), dirección: https://imbalanced-learn.org/stable/under_sampling.html#controlled-under-sampling (visitado 25-06-2025).

[6] ResearchGate. "Sentiment Analysis on Twitter Data for Portuguese Language - Scientific Figure." (), dirección: https://www.researchgate.net/figure/Results-without-pre-processing-and-using-the-Sentilex-lexicon_tbl1_262175717 (visitado 24-06-2025).

[7] ResearchGate, *Unlocking the potential of deep learning for marine ecology: Overview, applications, and outlook - Scientific Figure*, [Online]. Available: https://www.researchgate.net/figure/Evaluation-metrics-accuracy-precision-recall-F-score-and-Intersection-over-Union_fig2_358029719, Accessed: 2025-06-26.