PROYECTO CLASIFICACIÓN DE TEXTO

Este artefacto hace parte del entregable **Alcanzando los objetivos de desarrollo sostenible. Un aporte desde el machine learning**, actividad de la asignatura de Machine Learning no Supervisado de la Maestria de Inteligencia Artificial de la Univerisdad de los Andes.

Integrantes:

- Jaime Alberto Rodriguez - 200717791
- Yezid Alejandro Garcia - 200810710

ANALISIS DEL PROBLEMA - COMPRENSIÓN DEL NEGOCIO

Se busca desarrollar un método automatizado que realice clasificaciones sobre textos, alineados con los objetivos de desarrollo sostenible definidos por la ONU. Esto podra ser de utilidad como herramienta de apoyo para analizar escritos, objetivos o proyectos y evaluar de forma prematura su alineación con los mencionados objetivos.

Selección de Algoritmos

Para garantizar el exito del proyecto se han seleccionado 3 algoritmos que si bien tienen como proposito la clasificación, tienen caracteristicas propias que seran de utilidad para el procesamiento de disintos tipos de textos. Consideramos que el uso combinado o comparativo de estos algoritmos ofreceran distintas perspectivas y ventajas en la tarea de realizar clasificaciones de texto, abarcando diferentes características y requisitos de los textos que se procesen:

Algoritmo	Descripción	Ventaja Frente al Problema
Logistic Regression	Es un clasificador lineal que modela probabilidades (multiclase vía softmax o One-vs-Rest) con regularización (L1/L2).	Es rapido y escalable ademas de ser adecuado para espacios de alta dimensión y dispersos como los encontrados en BOW/TF-IDF.
Random Forest	Ensamble de árboles con bagging y submuestreo de características que captura relaciones no lineales.	Su robustez al ruido le permite capturar relaciones no lineales e interacciones entre características.
K-Neighbors (KNN)	Clasificador no parametrico basado en vecinos más cercanos según una métrica de distancia.	Ofrece una solución simple y efectiva sin necesidad de un entrenamiento costoso, gracias a la definición de distancias significativas y vecindarios bien definidos luego de aplicar alguna tecnica de reducción de dimensionalidad.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import nltk
from nltk.tokenize import word_tokenize
from nltk import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.base import TransformerMixin, BaseEstimator


from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.manifold import TSNE
```

Carga y Exploración Inicial de Datos Carga el dataset desde el archivo Train_textosODS.xlsx. Se crea una copia (*data*) para trabajar sobre ella sin modificar el original.

In [2]:
```
raw_data = pd.read_excel('Train_textosODS.xlsx')
data = raw_data.copy()
data.head()
```

Out[2]:

	textos	ODS
0	"Aprendizaje" y "educación" se consideran sinó...	4
1	No dejar clara la naturaleza de estos riesgos ...	6
2	Como resultado, un mayor y mejorado acceso al ...	13
3	Con el Congreso firmemente en control de la ju...	16
4	Luego, dos secciones finales analizan las impl...	5

Split del dataset se dividen los datos en entrenamiento y pruebas 80/20, así se garantiza que el modelo se evalúe con textos que no ha visto durante el aprendizaje, asegurando una medida objetiva de su capacidad de generalización.

In [3]:
```
x_train, x_test, y_train, y_test = train_test_split(data['textos'], data['ODS'], test_size=0.2, random_state=42)
```

Descarga de Recursos para Preprocesamiento Se descargan las _stopwords?, que es el corpus de palabras con bajo valor semantico que provee nltk.

Es un paso preparatorio indispensable para la limpieza de los textos que se realizará dentro del pipeline

In [4]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/I508111/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[4]:
```
True
```

Definición de Estrategias Se define un diccionario que parametriza el pipeline para obtener las bolsas de palabras, a la vez es la parte experimental en la reducción de dimensionalidad.

Cada diccionario representa una combinación única de vectorización de texto (count para Bag-of-Words o tfidf para TF-IDF) y una técnica de reducción de dimensionalidad (svd o pca). Esto permite probar y comparar sistemáticamente cuatro pipelines de preparación de datos, lo cual es fundamental para justificar la elección final del método, como lo solicita el proyecto.

Igualmente, cada diccionario define placeholders para almacenar los datos de entrenamiento y pruebas resultantes de la transformación.

In [5]:
```
bow_list =[
    {
        "vectorized": "count",
        "reduction": "svd",
        "train_data": None,
        "test_data": None
    },
    {
        "vectorized": "tfidf",
        "reduction": "svd",
        "train_data": None,
        "test_data": None
    },
    {
        "vectorized": "count",
        "reduction": "pca",
        "train_data": None,
        "test_data": None
    },
    {
        "vectorized": "tfidf",
        "reduction": "pca",
        "train_data": None,
        "test_data": None
    }
]
```

Preprocesador de Texto Esta clase que hereda de BaseEstimator y TransformerMixin de sklearn para ser compatible con los pipelines, encapsula los pasos de preprocesamiento de texto:

1. Tokenización: Divide el texto en palabras (RegexpTokenizer).
2. Eliminación de Stopwords: Filtra las palabras de bajo valor semántico del Español.
3. Stemming: Reduce las palabras a su raíz (PorterStemmer).

Este transformador es el primer paso del pipeline de preparación de datos.

In [6]:
```
class TextPreprocessorTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        return X.apply(self.text_preprocess)

    def text_preprocess(self, text):
        tokenizer = RegexpTokenizer(r'\w+')
        stemmer = PorterStemmer()

        tokens = tokenizer.tokenize(text)
        tokens = [word for word in tokens if word not in stopwords.words('spanish')]
        tokens = [stemmer.stem(word) for word in tokens]
        return ' '.join(tokens)
```

Transformador a Matriz Densa La clase ToDenseTransformer se crea como un paso auxiliar dentro del pipeline. La salida de los vectorizadores de texto suele ser una matriz dispersa (sparse matrix) para ahorrar memoria. Sin embargo, el algoritmo PCA de sklearn requiere una matriz densa como entrada.

Este transformador se encarga de realizar esta conversión, permitiendo que PCA funcione correctamente dentro del pipeline.

In [7]:
```
class ToDenseTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        return X.toarray()
```

Factory de Pipelines Esta clase construye el objeto Pipeline son sus respecivos pasos para preparar los datos, utiliza las clases previamente definidas y se basa en los parámetros de vectorización ('count' o 'tfidf') y reducción ('svd' o 'pca'):

1. El preprocesamiento de texto (TextPreprocessorTransformer).
2. La vectorización (usando CountVectorizer o TfidfVectorizer).
3. La reducción de dimensionalidad (usando TruncatedSVD o PCA).
    - TruncatedSVD es la base del Análisis Semántico Latente (LSI), una técnica que proyecta documentos y palabras en un espacio de baja dimensión para descubrir conceptos semánticos.
    - Ó, PCA como otro método de extracción de características que logra un fin similar.

In [8]:
```python
class DataProcessingPipelineFactory:
    def get_pipelines_and_params(self, vectorizer, reduction):

        steps = [
            ('preprocess', TextPreprocessorTransformer())
        ]
        if vectorizer == 'count':
            steps.append(('vectorizer', CountVectorizer()))
        if vectorizer == 'tfidf':
            steps.append(('vectorizer', TfidfVectorizer()))
        if reduction == 'svd':
            steps.append(('dimred', TruncatedSVD(n_components=100)))
        if reduction == 'pca':
            steps.append(('to_dense', ToDenseTransformer()))
            steps.append(('dimred', PCA(n_components=100)))

        pipeline = Pipeline(steps)
        return pipeline
```

Ejecución del pipeline y almacenamiento de resultados se recorre la lista bow_list que parametriza la creación de bolsas de palabras. En cada iteración, utiliza el factory de pipelines, los construye, y los aplica a los datos de entrenamiento (fit_transform) y prueba (transform).

Los datos transformados (con las características reducidas) se almacenan de nuevo en cada bolsa de palabras, así quedan listos para ser usados por los modelos de clasificación.

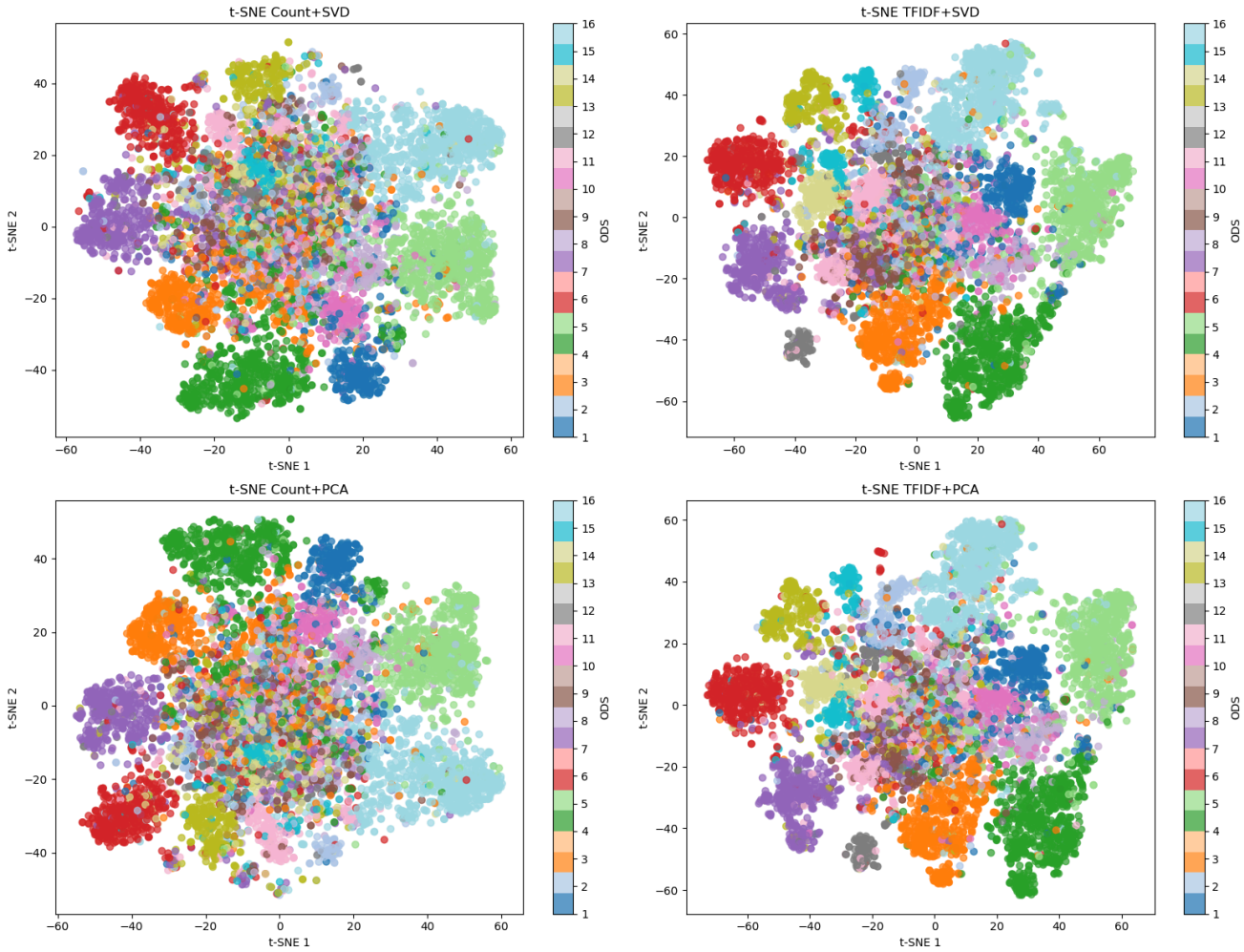
In [9]:
```python
for bow in bow_list:
    pipeline = DataProcessingPipelineFactory().get_pipelines_and_params(bow['vectorized'], bow['reduction'])
    bow['train_data'] = pipeline.fit_transform(x_train)
    bow['test_data'] = pipeline.transform(x_test)
```

Aplicación de t-SNE Con t-SNE se puede visualizar la separabilidad de las clases después de la reducción de dimensionalidad. Se recorren cada uno de los cuatro datasets transformados, reduciendo sus 100 dimensiones a solo 2, para ser graficadas.

Esta visualización ayuda a entender la estructura de los datos y la efectividad de cada pipeline de preprocesamiento.

TODO: detallar un poco cada grafica ---- Gráfica de los Resultados de t-SNE Esta celda genera una matriz de 2x2 gráficos, donde cada gráfico muestra la proyección t-SNE de uno de los cuatro pipelines de preparación de datos. Los puntos están coloreados según su ODS real. Esto permite una comparación visual directa para determinar qué combinación de vectorización y reducción de dimensionalidad agrupa mejor los textos de la misma categoría.

In [10]:
```python
nombres = ['Count+SVD', 'TFIDF+SVD', 'Count+PCA', 'TFIDF+PCA']
labels = y_train.astype(int)

tsne_results = []
for bow in bow_list:
    tsne = TSNE(n_components=2, random_state=42)
    tsne_result = tsne.fit_transform(bow['train_data'])
    tsne_results.append(tsne_result)

fig, axes = plt.subplots(2, 2, figsize=(16, 12))
for i, ax in enumerate(axes.flat):
    scatter = ax.scatter(tsne_results[i][:,0], tsne_results[i][:,1], c=labels, cmap='tab20', alpha=0.7)
    ax.set_title(f't-SNE {nombres[i]}')
    ax.set_xlabel('t-SNE 1')
    ax.set_ylabel('t-SNE 2')
    cbar = plt.colorbar(scatter, ax=ax, ticks=range(1,17))
    cbar.set_label('ODS')

plt.tight_layout()
plt.show()
```



Factory de Pipelines para los Modelos de Clasificación De manera análoga a la fábrica de datos, esta clase se encarga de construir el pipeline para el modelo de clasificación y definir una configuración de hiperparámetros para la optimización (param_grid).

Soporta tres algoritmos: LogisticRegression, RandomForestClassifier y KNeighborsClassifier. Este enfoque modular facilita la prueba y comparación de diferentes algoritmos, esto permite contrastar los diferentes modelos de clasificación.

In [11]:
```python
class ModelPipelineFactory:
    def get_pipelines_and_params(self, algorithm):

        if algorithm == 'logistic':
            model = LogisticRegression(max_iter=2000)
            param_grid = {
                'model__C': [0.1, 1, 10],
                'model__solver': ['lbfgs', 'saga']
            }
        elif algorithm == 'randomforest':
            model = RandomForestClassifier()
            param_grid = {
                'model__n_estimators': [100, 200],
                'model__max_depth': [None, 10, 20]
            }
        elif algorithm == 'kneighbors':
            model = KNeighborsClassifier()
            param_grid = {
                'model__n_neighbors': [3, 5, 7],
                'model__weights': ['uniform', 'distance']
            }
        else:
            raise ValueError(f"Algoritmo no soportado: {algorithm}")

        steps = [
            ('model', model)
        ]

        pipeline = Pipeline(steps)
        return pipeline, param_grid
```

Función helper para visualizar la Evaluación esta función estandariza la visualización de los resultados de aplicar un algoritmos de clasificación a una bolsa de palabras. Recibe las etiquetas verdaderas y las predicciones, y genera dos visualizaciones:

1. Un reporte de clasificación (classification_report) que muestra métricas como precisión, recall y f1-score por cada clase.

2. Una matriz de confusión (confusion_matrix) que visualiza los aciertos y errores del modelo. Esta función es clave para cumplir con el requisito de evaluación del modelo

In [12]:
```
def print_reports(y_test, y_pred, title):
    # Reporte de clasificación como tabla
    report_dict = classification_report(y_test, y_pred, output_dict=True)
    df_report = pd.DataFrame(report_dict).transpose()

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

    # Tabla de reporte de clasificación
    ax1.axis('off')
    table = ax1.table(cellText=df_report.round(2).values,
                      colLabels=df_report.columns,
                      rowLabels=df_report.index,
                      loc='center',
                      cellLoc='center')
    ax1.set_title('Reporte de Clasificación: ' + title)

    # Matriz de confusión con ConfusionMatrixDisplay
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(ax=ax2, cmap=plt.cm.Blues)
    ax2.set_title('Matriz de Confusión')

    plt.tight_layout()
    plt.show()
```

In [13]:
```
algorithms = ['logistic', 'randomforest', 'kneighbors']
#algorithms = ['kneighbors']
```

Ciclos de Entrenamiento, Optimización y Evaluación Este es el núcleo del proyecto, donde se unen todos los componentes anteriores. Ejecuta cada algoritmo de clasificación para cada una de las bolsas de palabras preparadas previamente:

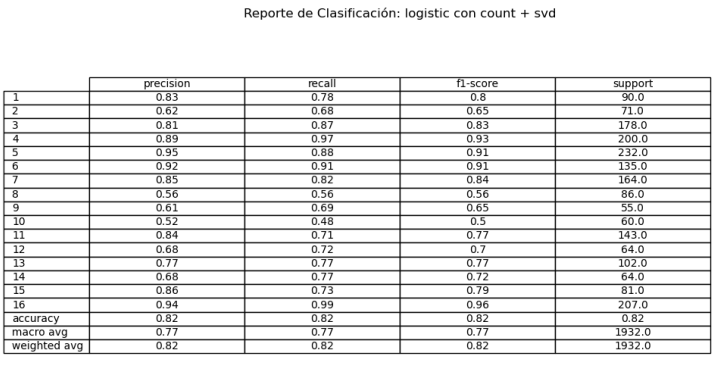
1. Optimización: Utiliza GridSearchCV para encontrar la mejor combinación de hiperparámetros para cada modelo, usando validación cruzada de 3 subconjuntos (folds). Esta es la búsqueda de hiperparámetros.
2. Entrenamiento: El GridSearchCV entrena el modelo con los mejores hiperparámetros encontrados.
3. Evaluación: Realiza predicciones sobre el conjunto de prueba (que son los datos no vistos) y utiliza la función helper para visualizar el rendimiento final.

Los resultados impresos muestran que la combinación de regresión logística con vectorización TF-IDF y reducción de dimensionalidad PCA obtiene el mejor score de validación (0.8600), justificando empíricamente su elección como el mejor modelo.

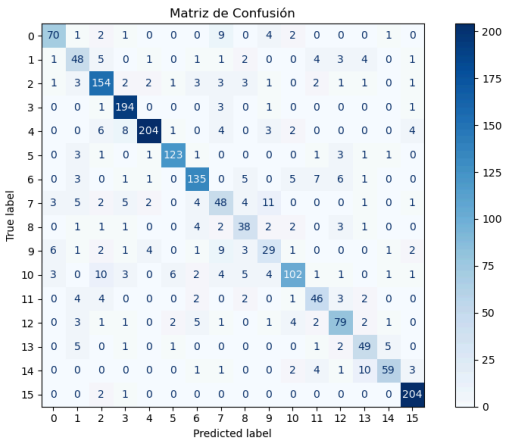
TODO: Detallar cada resultado.

In [14]:
```
for algorithm in algorithms:
    for bow in bow_list:
        title = f"{algorithm} con {bow['vectorized']} + {bow['reduction']}"
        print("ENTRENAMIENTO Y EVALUACION PARA:", title)
        pipelineFactory = ModelPipelineFactory()
        pipeline, params = pipelineFactory.get_pipelines_and_params(algorithm)
        grid = GridSearchCV(pipeline, params, cv=3, scoring='accuracy', n_jobs=-1)
        grid.fit(bow['train_data'], y_train)
        print("Mejores hiperparámetros:", grid.best_params_)
        print("Mejor score de validación:", grid.best_score_)

        y_pred = grid.predict(bow['test_data'])

        print_reports(y_test, y_pred, title)
```

```
ENTRENAMIENTO Y EVALUACION PARA: logistic con count + svd
Mejores hiperparámetros: {'model__C': 0.1, 'model__solver': 'lbfgs'}
Mejor score de validación: 0.8065766955087343
```



	precision	recall	f1-score	support
1	0.83	0.78	0.8	90.0
2	0.62	0.68	0.65	71.0
3	0.81	0.87	0.83	178.0
4	0.89	0.97	0.93	200.0
5	0.95	0.88	0.91	232.0
6	0.92	0.91	0.91	135.0
7	0.85	0.82	0.84	164.0
8	0.56	0.56	0.56	86.0
9	0.61	0.69	0.65	55.0
10	0.52	0.48	0.5	60.0
11	0.84	0.71	0.77	143.0
12	0.68	0.72	0.7	64.0
13	0.77	0.77	0.77	102.0
14	0.68	0.77	0.72	64.0
15	0.86	0.73	0.79	81.0
16	0.94	0.99	0.96	207.0
accuracy	0.82	0.82	0.82	0.82
macro avg	0.77	0.77	0.77	1932.0
weighted avg	0.82	0.82	0.82	1932.0

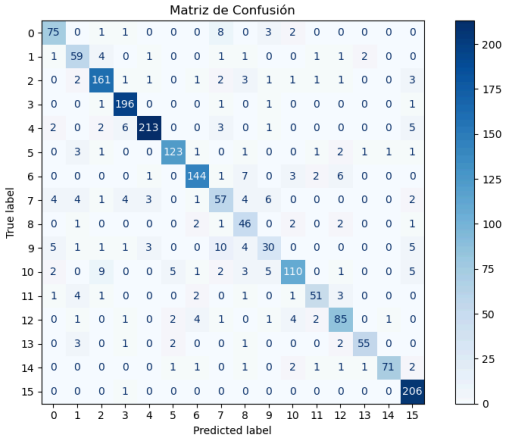



```
ENTRENAMIENTO Y EVALUACION PARA: logistic con tfidf + svd
Mejores hiperparámetros: {'model__C': 10, 'model__solver': 'lbfgs'}
Mejor score de validación: 0.8581050233477417
```

Reporte de Clasificación: logistic con tfidf + svd

	precision	recall	f1-score	support
1	0.83	0.83	0.83	90.0
2	0.76	0.83	0.79	71.0
3	0.88	0.9	0.89	178.0
4	0.92	0.98	0.95	200.0
5	0.96	0.92	0.94	232.0
6	0.92	0.91	0.92	135.0
7	0.92	0.88	0.9	164.0
8	0.66	0.66	0.66	86.0
9	0.64	0.84	0.72	55.0
10	0.62	0.5	0.56	60.0
11	0.88	0.77	0.82	143.0
12	0.86	0.8	0.83	64.0
13	0.82	0.83	0.83	102.0
14	0.93	0.86	0.89	64.0
15	0.97	0.88	0.92	81.0
16	0.89	1.0	0.94	207.0
accuracy	0.87	0.87	0.87	0.87
macro avg	0.84	0.84	0.84	1932.0
weighted avg	0.87	0.87	0.87	1932.0

Matriz de Confusión

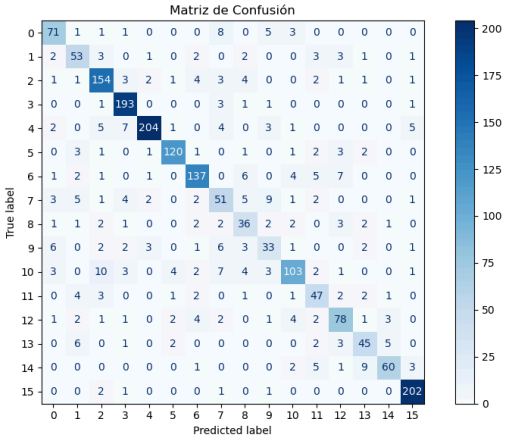


ENTRENAMIENTO Y EVALUACION PARA: logistic con count + pca
Mejores hiperparámetros: {'model__C': 0.1, 'model__solver': 'saga'}
Mejor score de validación: 0.8047643474827941

Reporte de Clasificación: logistic con count + pca

	precision	recall	f1-score	support
1	0.78	0.79	0.78	90.0
2	0.68	0.75	0.71	71.0
3	0.82	0.87	0.84	178.0
4	0.89	0.96	0.93	200.0
5	0.95	0.88	0.91	232.0
6	0.92	0.89	0.9	135.0
7	0.87	0.84	0.85	164.0
8	0.59	0.59	0.59	86.0
9	0.57	0.65	0.61	55.0
10	0.57	0.55	0.56	60.0
11	0.84	0.72	0.77	143.0
12	0.65	0.73	0.69	64.0
13	0.76	0.76	0.76	102.0
14	0.69	0.7	0.7	64.0
15	0.86	0.74	0.79	81.0
16	0.94	0.98	0.96	207.0
accuracy	0.82	0.82	0.82	0.82
macro avg	0.77	0.78	0.77	1932.0
weighted avg	0.82	0.82	0.82	1932.0

Matriz de Confusión

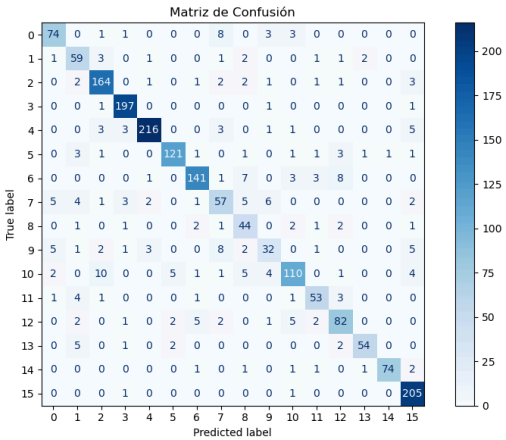


ENTRENAMIENTO Y EVALUACION PARA: logistic con tfidf + pca
Mejores hiperparámetros: {'model__C': 10, 'model__solver': 'lbfgs'}
Mejor score de validación: 0.8623777229602473

Reporte de Clasificación: logistic con tfidf + pca

	precision	recall	f1-score	support
1	0.84	0.82	0.83	90.0
2	0.73	0.83	0.78	71.0
3	0.88	0.92	0.9	178.0
4	0.94	0.98	0.96	200.0
5	0.96	0.93	0.95	232.0
6	0.93	0.9	0.91	135.0
7	0.92	0.86	0.89	164.0
8	0.68	0.66	0.67	86.0
9	0.64	0.8	0.71	55.0
10	0.65	0.53	0.59	60.0
11	0.86	0.77	0.81	143.0
12	0.83	0.83	0.83	64.0
13	0.8	0.8	0.8	102.0
14	0.93	0.84	0.89	64.0
15	0.99	0.91	0.95	81.0
16	0.9	0.99	0.94	207.0
accuracy	0.87	0.87	0.87	0.87
macro avg	0.84	0.84	0.84	1932.0
weighted avg	0.87	0.87	0.87	1932.0

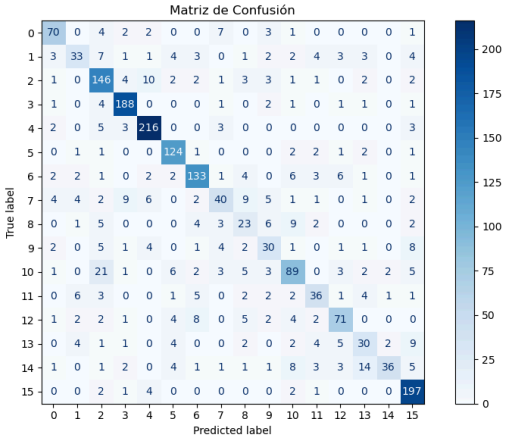
Matriz de Confusión



ENTRENAMIENTO Y EVALUACION PARA: randomforest con count + svd
Mejores hiperparámetros: {'model__max_depth': None, 'model__n_estimators': 200}
Mejor score de validación: 0.7402901808727052

Reporte de Clasificación: randomforest con count + svd

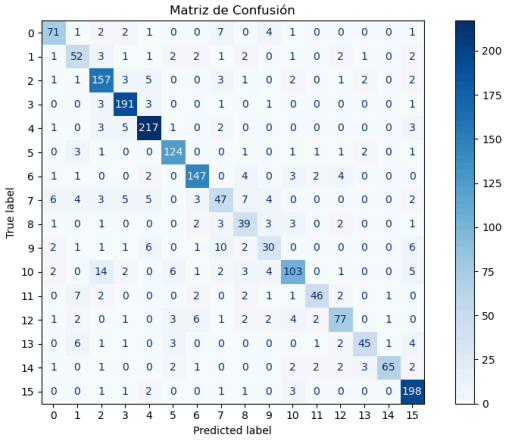
	precision	recall	f1-score	support
1	0.8	0.78	0.79	90.0
2	0.62	0.46	0.53	71.0
3	0.7	0.82	0.75	178.0
4	0.88	0.94	0.91	200.0
5	0.88	0.93	0.91	232.0
6	0.82	0.92	0.87	135.0
7	0.82	0.81	0.82	164.0
8	0.62	0.47	0.53	86.0
9	0.4	0.42	0.41	55.0
10	0.51	0.5	0.5	60.0
11	0.68	0.62	0.65	143.0
12	0.61	0.56	0.59	64.0
13	0.75	0.7	0.72	102.0
14	0.49	0.47	0.48	64.0
15	0.88	0.44	0.59	81.0
16	0.81	0.95	0.88	207.0
accuracy	0.76	0.76	0.76	0.76
macro avg	0.7	0.67	0.68	1932.0
weighted avg	0.75	0.76	0.75	1932.0



ENTRENAMIENTO Y EVALUACION PARA: randomforest con tfidf + svd
Mejores hiperparámetros: {'model__max_depth': None, 'model__n_estimators': 200}
Mejor score de validación: 0.8254782326626987

Reporte de Clasificación: randomforest con tfidf + svd

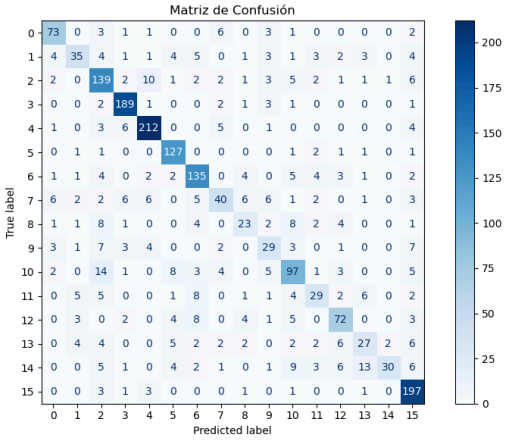
	precision	recall	f1-score	support
1	0.81	0.79	0.8	90.0
2	0.67	0.73	0.7	71.0
3	0.81	0.88	0.85	178.0
4	0.9	0.96	0.92	200.0
5	0.9	0.94	0.92	232.0
6	0.88	0.92	0.9	135.0
7	0.89	0.9	0.89	164.0
8	0.6	0.55	0.57	86.0
9	0.61	0.71	0.66	55.0
10	0.61	0.5	0.55	60.0
11	0.83	0.72	0.77	143.0
12	0.85	0.72	0.78	64.0
13	0.82	0.75	0.79	102.0
14	0.85	0.7	0.77	64.0
15	0.96	0.8	0.87	81.0
16	0.87	0.96	0.91	207.0
accuracy	0.83	0.83	0.83	0.83
macro avg	0.8	0.78	0.79	1932.0
weighted avg	0.83	0.83	0.83	1932.0



ENTRENAMIENTO Y EVALUACION PARA: randomforest con count + pca
Mejores hiperparámetros: {'model__max_depth': None, 'model__n_estimators': 200}
Mejor score de validación: 0.7364058307747628

Reporte de Clasificación: randomforest con count + pca

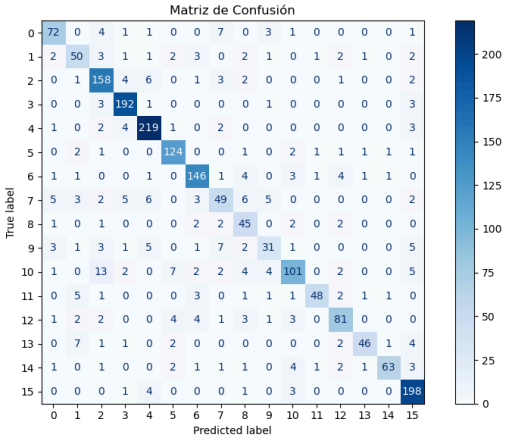
	precision	recall	f1-score	support
1	0.78	0.81	0.8	90.0
2	0.66	0.49	0.56	71.0
3	0.68	0.78	0.73	178.0
4	0.88	0.94	0.91	200.0
5	0.88	0.91	0.9	232.0
6	0.81	0.94	0.87	135.0
7	0.78	0.82	0.8	164.0
8	0.62	0.47	0.53	86.0
9	0.52	0.42	0.46	55.0
10	0.5	0.48	0.49	60.0
11	0.67	0.68	0.68	143.0
12	0.58	0.45	0.51	64.0
13	0.71	0.71	0.71	102.0
14	0.5	0.42	0.46	64.0
15	0.91	0.37	0.53	81.0
16	0.79	0.95	0.86	207.0
accuracy	0.75	0.75	0.75	0.75
macro avg	0.71	0.67	0.68	1932.0
weighted avg	0.75	0.75	0.74	1932.0



ENTRENAMIENTO Y EVALUACION PARA: randomforest con tfidf + pca
Mejores hiperparámetros: {'model__max_depth': 20, 'model__n_estimators': 200}
Mejor score de validación: 0.824443338035571

Reporte de Clasificación: randomforest con tfidf + pca

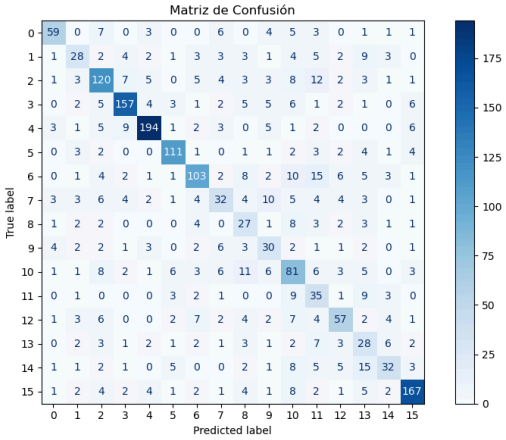
	precision	recall	f1-score	support
1	0.82	0.8	0.81	90.0
2	0.69	0.7	0.7	71.0
3	0.81	0.89	0.85	178.0
4	0.91	0.96	0.93	200.0
5	0.9	0.94	0.92	232.0
6	0.87	0.92	0.9	135.0
7	0.88	0.89	0.88	164.0
8	0.65	0.57	0.61	86.0
9	0.62	0.82	0.71	55.0
10	0.66	0.52	0.58	60.0
11	0.83	0.71	0.77	143.0
12	0.92	0.75	0.83	64.0
13	0.82	0.79	0.81	102.0
14	0.9	0.72	0.8	64.0
15	0.94	0.78	0.85	81.0
16	0.86	0.96	0.91	207.0
accuracy	0.84	0.84	0.84	0.84
macro avg	0.82	0.79	0.8	1932.0
weighted avg	0.84	0.84	0.84	1932.0



ENTRENAMIENTO Y EVALUACION PARA: kneighbors con count + svd
Mejores hiperparámetros: {'model__n_neighbors': 7, 'model__weights': 'distance'}
Mejor score de validación: 0.6447440297925735

Reporte de Clasificación: kneighbors con count + svd

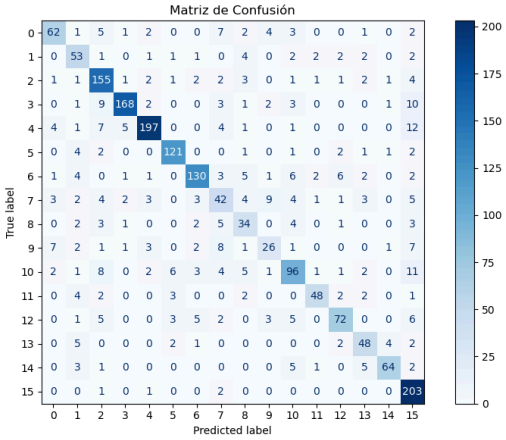
	precision	recall	f1-score	support
1	0.78	0.66	0.71	90.0
2	0.51	0.39	0.44	71.0
3	0.67	0.67	0.67	178.0
4	0.83	0.78	0.81	200.0
5	0.88	0.84	0.86	232.0
6	0.82	0.82	0.82	135.0
7	0.73	0.63	0.68	164.0
8	0.46	0.37	0.41	86.0
9	0.35	0.49	0.41	55.0
10	0.41	0.5	0.45	60.0
11	0.49	0.57	0.52	143.0
12	0.32	0.55	0.41	64.0
13	0.63	0.56	0.59	102.0
14	0.29	0.44	0.35	64.0
15	0.56	0.4	0.46	81.0
16	0.84	0.81	0.82	207.0
accuracy	0.65	0.65	0.65	0.65
macro avg	0.6	0.59	0.59	1932.0
weighted avg	0.67	0.65	0.66	1932.0



ENTRENAMIENTO Y EVALUACION PARA: kneighbors con tfidf + svd
Mejores hiperparámetros: {'model__n_neighbors': 7, 'model__weights': 'distance'}
Mejor score de validación: 0.7716218193888097

Reporte de Clasificación: kneighbors con tfidf + svd

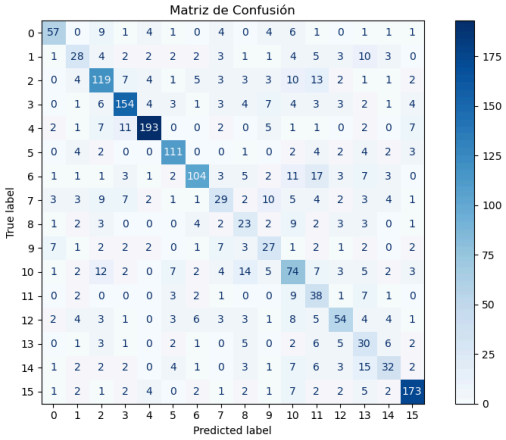
	precision	recall	f1-score	support
1	0.78	0.69	0.73	90.0
2	0.62	0.75	0.68	71.0
3	0.76	0.87	0.81	178.0
4	0.93	0.84	0.88	200.0
5	0.92	0.85	0.88	232.0
6	0.88	0.9	0.89	135.0
7	0.87	0.79	0.83	164.0
8	0.51	0.49	0.5	86.0
9	0.54	0.62	0.58	55.0
10	0.57	0.43	0.49	60.0
11	0.73	0.67	0.7	143.0
12	0.86	0.75	0.8	64.0
13	0.8	0.71	0.75	102.0
14	0.71	0.75	0.73	64.0
15	0.89	0.79	0.84	81.0
16	0.74	0.98	0.84	207.0
accuracy	0.79	0.79	0.79	0.79
macro avg	0.76	0.74	0.75	1932.0
weighted avg	0.79	0.79	0.79	1932.0



ENTRENAMIENTO Y EVALUACION PARA: kneighbors con count + pca
Mejores hiperparámetros: {'model__n_neighbors': 7, 'model__weights': 'distance'}
Mejor score de validación: 0.6407300789825062

Reporte de Clasificación: kneighbors con count + pca

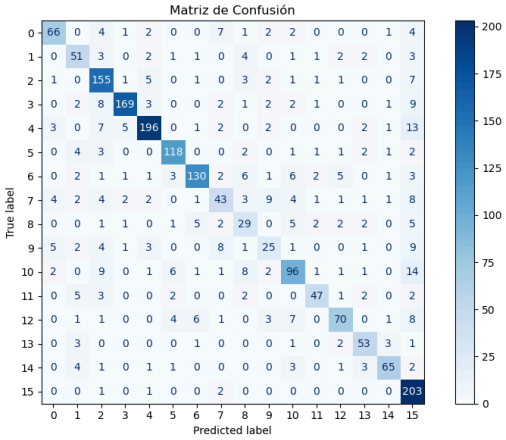
	precision	recall	f1-score	support
1	0.74	0.63	0.68	90.0
2	0.48	0.39	0.43	71.0
3	0.65	0.67	0.66	178.0
4	0.79	0.77	0.78	200.0
5	0.89	0.83	0.86	232.0
6	0.79	0.82	0.81	135.0
7	0.79	0.63	0.7	164.0
8	0.45	0.34	0.38	86.0
9	0.33	0.42	0.37	55.0
10	0.39	0.45	0.42	60.0
11	0.46	0.52	0.49	143.0
12	0.33	0.59	0.42	64.0
13	0.62	0.53	0.57	102.0
14	0.3	0.47	0.36	64.0
15	0.52	0.4	0.45	81.0
16	0.86	0.84	0.85	207.0
accuracy	0.64	0.64	0.64	0.64
macro avg	0.59	0.58	0.58	1932.0
weighted avg	0.67	0.64	0.65	1932.0




```
ENTRENAMIENTO Y EVALUACION PARA: kneighbors con tfidf + pca
Mejores hiperparámetros: {'model__n_neighbors': 7, 'model__weights': 'distance'}
Mejor score de validación: 0.7708449191944338
```

Reporte de Clasificación: kneighbors con tfidf + pca

	precision	recall	f1-score	support
1	0.81	0.73	0.77	90.0
2	0.67	0.72	0.69	71.0
3	0.76	0.87	0.81	178.0
4	0.93	0.84	0.89	200.0
5	0.9	0.84	0.87	232.0
6	0.87	0.87	0.87	135.0
7	0.88	0.79	0.84	164.0
8	0.61	0.5	0.55	86.0
9	0.48	0.53	0.5	55.0
10	0.52	0.42	0.46	60.0
11	0.74	0.67	0.7	143.0
12	0.82	0.73	0.78	64.0
13	0.8	0.69	0.74	102.0
14	0.77	0.83	0.8	64.0
15	0.87	0.8	0.83	81.0
16	0.69	0.98	0.81	207.0
accuracy	0.78	0.78	0.78	0.78
macro avg	0.76	0.74	0.75	1932.0
weighted avg	0.79	0.78	0.78	1932.0



En este proyecto se desarrolló una solución de machine learning para clasificar textos según los 17 Objetivos de Desarrollo Sostenible (ODS), abordando la necesidad de automatizar el análisis de información textual para organizaciones como la UNFPA.

El trabajo se estructuró mediante pipelines de sklearn que integraron sistemáticamente varias etapas. Primero, se realizó un preprocesamiento de texto que incluyó tokenización, eliminación de stopwords y stemming. Posteriormente, se representaron los textos numéricamente utilizando el esquema de bolsa de palabras (BOW), probando tanto la ponderación TF-IDF como el conteo simple de frecuencias.

Un aspecto central fue la reducción de la dimensionalidad, un requisito clave para manejar la complejidad de los datos textuales y mejorar el rendimiento de los modelos. Para ello, se aplicaron y compararon dos técnicas de extracción de características: Análisis de Componentes Principales (PCA) y Descomposición en Valores Singulares (SVD), la cual es la base del Análisis Semántico Latente (LSI). Finalmente, se entrenaron y optimizaron tres algoritmos de clasificación distintos (LogisticRegression, RandomForestClassifier y KNeighborsClassifier) mediante una búsqueda exhaustiva de hiperparámetros con GridSearchCV.

Los resultados de este proceso experimental muestran que el enfoque fue efectivo. La evaluación las combinaciones de vectorización, reducción de dimensionalidad y algoritmos de clasificación permitió identificar una configuración óptima con base en evidencia empírica. El modelo con el mejor desempeño fue el de Regresión Logística, que alcanzó un score de validación del 86%. Este resultado se obtuvo al combinar la representación vectorial de textos mediante TF-IDF con la técnica de reducción de dimensionalidad PCA.

La conclusión principal es que la extracción de características como PCA o SVD es fundamental para transformar la representación de alta dimensionalidad de los textos en un espacio semántico de menor dimensionalidad. Esta nueva representación, que captura conceptos en lugar de términos aislados, permitió que el modelo de clasificación alcanzara un alto rendimiento, cumpliendo con éxito el objetivo de construir una solución consistente para la clasificación automática de textos en el contexto de los ODS.