Untitled

October 11, 2025

1 NOMBRES

Eliecer Bautista Belen // Víctor M. Díaz

2 MATRÍCULA O ID

100064003 // 100049725

3 ASIGNATURA

Inteligencia Artificial

4 TEMA O ASIGNACIÓN

Práctica Final de Inteligencia Artificial

5 FECHA

03 de Octubre de 2025

[ ]:

6 1.0 Aprendizaje Supervisado

6.0.1 IPORTAR LIBRERÍAS Y CONFICURACIÓN

[3]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix, classification_report
from sklearn.decomposition import TruncatedSVD
```

```
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans, DBSCAN
import joblib
```

6.0.2 funcuión auxiliar

[43]:
```
# Define una función auxiliar para crear un directorio si no existe.
def ensure_dir(path):
    # Crea todos los directorios intermedios necesarios; no lanza error si ya␣
 ↪existe (exist_ok=True).
    os.makedirs(path, exist_ok=True)
```

6.0.3 Paths

[42]:
```
# Define la ruta base donde se esperan los archivos
BASE = "/mnt/data"
# Define un diccionario con las rutas completas a los archivos CSV que el␣
 ↪notebook espera.
FILES = {
    'products': os.path.join(BASE, 'products_ai_project.csv'),
    'reviews': os.path.join(BASE, 'reviews_ai_project.csv'),
    'users': os.path.join(BASE, 'users_ai_project.csv')
}
```

6.0.4 cargar el data sets

[25]:
```
# Informa por consola qué archivos hay configurados y si existen en el sistema␣
 ↪de archivos.
print("Available files and existence:")
for k,v in FILES.items():
    # Para cada par clave/valor en FILES imprime el nombre lógico y si el␣
 ↪archivo existe o está ausente.
    print(f" - {k}: {v} ->", 'FOUND' if os.path.exists(v) else 'MISSING')
```

```
Available files and existence:
 - products: /mnt/data\products_ai_project.csv -> FOUND
 - reviews: /mnt/data\reviews_ai_project.csv -> FOUND
 - users: /mnt/data\users_ai_project.csv -> FOUND
```

[26]:
```
# Si el archivo de reseñas (reviews) no existe, detiene la ejecución levantando␣
 ↪una excepción clara.
if not os.path.exists(FILES['reviews']):
    raise FileNotFoundError('reviews_ai_project.csv is required for this␣
 ↪notebook. Place it in /mnt/data')
```

[27]:
```
# estou verificando si el archivo existe para saber si fue creado correctamente␣
 ↪true existe y false no existe
```

```
print("Ruta buscada:", FILES['reviews'])
print("Existe el archivo?:", os.path.exists(FILES['reviews']))
```

```
Ruta buscada: /mnt/data\reviews_ai_project.csv
Existe el archivo?: True
```

6.0.5 Cargue el CSV de reseñas en un DataFrame de pandas llamado .

[28]:
```
reviews = pd.read_csv(FILES['reviews'])
# Muestra en consola la forma (número de filas y columnas) del DataFrame␣
 ↪cargado.
print('\nLoaded reviews: shape=', reviews.shape)
# Muestra en consola la lista de columnas disponibles en el DataFrame, para␣
 ↪inspección rápida.
print('Columns:', list(reviews.columns))
```

```
Loaded reviews: shape= (1500, 15)
Columns: ['review_id', 'user_id', 'product_id', 'category', 'brand', 'city',
'rating', 'review_text', 'review_date', 'helpful_votes', 'purchase_count_90d',
'avg_spend_90d', 'return_rate', 'sentiment_label', 'topics_tags']
```

6.0.6 Detectar texto y etiquetas de columnas

[29]:
```
# Busca columnas cuyo nombre probablemente contenga texto de reseñas -␣
 ↪construye lista de candidatos.
text_candidates = [c for c in reviews.columns if any(x in c.lower() for x in␣
 ↪['review','text','comment','body','feedback'])]
# Busca columnas cuyo nombre probablemente contenga la etiqueta (rating/score/
 ↪etc.) - construye lista de candidatos.
label_candidates = [c for c in reviews.columns if any(x in c.lower() for x in␣
 ↪['rating','score','stars','sentiment','label'])]
# Imprime en consola las columnas candidatas detectadas para texto y etiqueta.
print('Text candidates:', text_candidates)
print('Label candidates:', label_candidates)
```

```
Text candidates: ['review_id', 'review_text', 'review_date']
Label candidates: ['rating', 'sentiment_label']
```

[30]:
```
# Eligir una columna de texto por defecto: prefiere 'review_text' si existe, si␣
 ↪no el primer candidato detectado, si no None.
text_col = 'review_text' if 'review_text' in reviews.columns else␣
 ↪(text_candidates[0] if text_candidates else None)
# Elige la columna de etiqueta por defecto: prefiere 'rating' si existe, si no␣
 ↪el primer candidato detectado, si no None.
label_col = 'rating' if 'rating' in reviews.columns else (label_candidates[0]␣
 ↪if label_candidates else None)
# Imprime qué columnas se usarán finalmente como texto y etiqueta.
```

```
print('Using text_col=', text_col, 'label_col=', label_col)

# Si no detectó ninguna columna de texto, lanza un error porque el pipeline␣
 ↪necesita texto para funcionar.
if text_col is None:
    raise ValueError('No text column found in reviews dataset. Please verify␣
 ↪column names.')
```

```
Using text_col= review_text label_col= rating
```

6.0.7 preparar etiquetas (sentimiento binario)

[31]:
```
# Si no existe una columna de etiqueta, crear etiquetas pseudo-supervisadas␣
 ↪usando heurística de palabras clave.
if label_col is None:
    # Informa que se creará una etiqueta con heurística (ruidosa).
    print('No label column detected. Creating pseudo-label using keywords␣
 ↪(fallback, noisy).')
    # Define una función simple que asigna 1/0 según conteo de palabras␣
 ↪positivas vs negativas en el texto.
    def simple_sent(s):
        # Convierte el valor a string y a minúsculas para búsquedas insensibles␣
 ↪a mayúsculas.
        s = str(s).lower()
        # Lista de palabras consideradas positivas.
        pos_words = ['good','great','excellent','love','recommend','happy']
        # Lista de palabras consideradas negativas.
        neg_words = ['bad','terrible','poor','hate','disappoint','worst']
        # Cuenta cuántas palabras positivas aparecen en el texto.
        p = sum(1 for w in pos_words if w in s)
        # Cuenta cuántas palabras negativas aparecen en el texto.
        n = sum(1 for w in neg_words if w in s)
        # Devuelve 1 si hay más señales positivas que negativas, si no 0.
        return 1 if p>n else 0
    # Aplica la función definida anteriormente a la columna de texto y guarda␣
 ↪el resultado en 'label_bin'.
    reviews['label_bin'] = reviews[text_col].apply(simple_sent)
else:
    # Si existe una columna de etiqueta, se maneja según su tipo (numérica o␣
 ↪categórica).
    if label_col == 'rating' or np.issubdtype(reviews[label_col].dtype, np.
 ↪number):
        # Si la etiqueta es numérica (p. ej. rating), convierte a binaria:␣
 ↪rating >= 4 => 1 (positivo), else 0.
        reviews['label_bin'] = (reviews[label_col] >= 4).astype(int)
    else:
        # Si la etiqueta es categórica:
```

```
        if reviews[label_col].nunique() == 2:
            # Si solo hay dos categorías, factorízalas (pd.factorize devuelve␣
 ↪códigos 0/1).
            reviews['label_bin'] = pd.factorize(reviews[label_col])[0]
        else:
            # Si hay muchas categorías, aplica heurística textual para marcar␣
 ↪como positivo si contiene palabras clave.
            reviews['label_bin'] = reviews[label_col].astype(str).str.
 ↪contains('pos|positive|good|great|excellent', case=False).astype(int)

# Muestra la distribución de la nueva columna binaria (cuántos 0 y 1).
print('Label distribution:')
print(reviews['label_bin'].value_counts())
```

```
Label distribution:
label_bin
0    846
1    654
Name: count, dtype: int64
```

6.0.8 División de entrenamiento/prueba

[44]:
```
# Crea un DataFrame supervisado con solo las columnas de texto y la etiqueta␣
 ↪binaria, elimina filas con NA y copia para evitar vistas.
df_sup = reviews[[text_col, 'label_bin']].dropna().copy()
# Extrae X (texto) como array de strings.
X = df_sup[text_col].astype(str).values
# Extrae y (etiqueta) como array numpy.
y = df_sup['label_bin'].values
# Separa el dataset en entrenamiento y prueba; stratify=y asegura proporciones␣
 ↪similares de clases en ambos sets; random_state fija aleatoriedad.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪stratify=y, random_state=42)
# Imprime tamaños de entrenamiento y prueba.
print('\nTrain/Test sizes:', X_train.shape[0], X_test.shape[0])
```

```
Train/Test sizes: 1200 300
```

6.0.9 Vectorizacion (TF-IDF)

[46]:
```
# Comentario: TF-IDF convierte texto en vectores numéricos ponderando␣
 ↪frecuencia por inversa de documento; bueno para modelos clásicos.
# Crea el vectorizador TF-IDF con máximo de 8000 características, n-gramas de 1␣
 ↪y 2 palabras, y stopwords en inglés removidas.
tfidf = TfidfVectorizer(max_features=8000, ngram_range=(1,2),␣
 ↪stop_words='english')
```

```
# Ajusta el vectorizador al texto de entrenamiento y transforma X_train a␣
 ↪matriz dispersa TF-IDF.
X_train_tfidf = tfidf.fit_transform(X_train)
# Transforma el texto de prueba usando el vectorizador ya ajustado (sin refit).
X_test_tfidf = tfidf.transform(X_test)
# Imprime las formas de las matrices TF-IDF (filas x columnas).
print('TF-IDF shapes:', X_train_tfidf.shape, X_test_tfidf.shape)
```

```
TF-IDF shapes: (1200, 1145) (300, 1145)
```

6.0.10 Modelos: Regresión logística y bosque aleatorio

[49]:
```
# Muestra ejemplos de predicciones en el conjunto de prueba para inspección␣
 ↪manual.
print('\nEjemplos de predicciones de pruebas:')
# Decide cuántos ejemplos mostrar: hasta 8 o el tamaño del conjunto de prueba␣
 ↪si es menor.
sample_n = min(8, len(X_test))
# Selecciona aleatoriamente 'sample_n' índices del conjunto de prueba sin␣
 ↪reemplazo.
for i in np.random.choice(len(X_test), sample_n, replace=False):
    # Obtiene el texto correspondiente al índice i.
    txt = X_test[i]
    # Obtiene la etiqueta verdadera (cast a int por seguridad).
    true = int(y_test[i])
    # Para cada modelo en results calcula la predicción sobre la muestra de␣
 ↪texto (transformando el texto con tfidf).
    preds = {name: int(results[name]['model'].predict(tfidf.
 ↪transform([txt]))[0]) for name in results}
    # Imprime un separador para legibilidad.
    print('\n---')
    # Imprime el texto (recorta a 300 caracteres y elimina saltos de línea para␣
 ↪presentarlo compacto).
    print('Text:', txt[:300].replace('\n',' '))
    # Imprime la etiqueta real.
    print('True label:', true)
    # Imprime las predicciones de cada modelo en formato diccionario.
    print('Predictions:', preds)
```

```
Ejemplos de predicciones de pruebas:

---
Text: Probé Photonix en categoría Electrónica. Experiencia limpio. ubicación
bien, talla regular. Vale cada peso.
True label: 1
Predictions: {'LogisticRegression': 1, 'RandomForest': 1}
```

```
---
Text: El ropa de ClassicLine es normal. La devolución fue adecuada y la talla
podría mejorar. Podría mejorar.
True label: 0
Predictions: {'LogisticRegression': 0, 'RandomForest': 0}

---
Text: Compré en EcoThread (Ropa). Me pareció delicioso; wifi y garantía fueron
determinantes. Cinco estrellas.
True label: 1
Predictions: {'LogisticRegression': 1, 'RandomForest': 1}

---
Text: Probé Hotel Mar Azul en categoría Hotel. Experiencia terrible. comodidad
bien, garantía regular. No cumple lo prometido.
True label: 0
Predictions: {'LogisticRegression': 0, 'RandomForest': 0}

---
Text: El ropa de Fit&Go es malo. La devolución fue adecuada y la talla podría
mejorar. No cumple lo prometido.
True label: 0
Predictions: {'LogisticRegression': 0, 'RandomForest': 0}

---
Text: Para ser Ropa, EcoThread resultó correcto. Destaco envío, aunque
devolución no fue ideal. Nada especial.
True label: 0
Predictions: {'LogisticRegression': 0, 'RandomForest': 0}

---
Text: Probé MercadoPlus en categoría Supermercado. Experiencia rápido. batería
bien, talla regular. Lo volvería a comprar.
True label: 1
Predictions: {'LogisticRegression': 1, 'RandomForest': 1}

---
Text: El electrónica de Auralink es limpio. La garantía fue adecuada y la
calidad podría mejorar. Vale cada peso.
True label: 1
Predictions: {'LogisticRegression': 1, 'RandomForest': 1}
```

6.0.11 Ejemplo de Prediccón

[50]:
```
# Use dos modelos distintos para comparar desempeño: uno lineal (logistic) y␣
↪uno conjunto (random forest).
# Define un diccionario con los modelos a entrenar y sus hiperparámetros.
```

```
models = {
    'LogisticRegression': LogisticRegression(max_iter=1000,␣
 ↪class_weight='balanced', solver='liblinear'),
    'RandomForest': RandomForestClassifier(n_estimators=200,␣
 ↪class_weight='balanced', random_state=42)
}

# Inicializa un diccionario vacío para almacenar resultados y métricas.
results = {}
# Itera sobre cada modelo definido en el diccionario models.
for name, model in models.items():
    # Imprime qué modelo se está entrenando.
    print('\nEntrenamiento', name)
    # Ajusta (entrena) el modelo con la matriz TF-IDF de entrenamiento y las␣
 ↪etiquetas de entrenamiento.
    model.fit(X_train_tfidf, y_train)
    # Predice las etiquetas para el conjunto de prueba transformado.
    y_pred = model.predict(X_test_tfidf)
    # Calcula la exactitud (accuracy) entre etiquetas verdaderas y predichas.
    acc = accuracy_score(y_test, y_pred)
    # Calcula precisión (precision), pasando zero_division=0 para evitar␣
 ↪errores cuando no hay positivos predichos.
    prec = precision_score(y_test, y_pred, zero_division=0)
    # Calcula recall (sensibilidad).
    rec = recall_score(y_test, y_pred, zero_division=0)
    # Calcula la métrica F1 (armónica entre precisión y recall).
    f1 = f1_score(y_test, y_pred, zero_division=0)
    # Imprime las métricas calculadas con formato.
    print(f"{name} -> Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall: {rec:.
 ↪4f}, F1: {f1:.4f}")
    # Imprime la matriz de confusión para ver TP/TN/FP/FN.
    print('Confusion matrix:')
    print(confusion_matrix(y_test, y_pred))
    # Imprime el reporte de clasificación (precision/recall/f1 por clase).
    print('Classification report:')
    print(classification_report(y_test, y_pred, zero_division=0))
    # Guarda el modelo entrenado y sus métricas en el diccionario results para␣
 ↪uso posterior.
    results[name] = {'model': model, 'metrics': {'accuracy': acc, 'precision':␣
 ↪prec, 'recall': rec, 'f1': f1}}

# Guarda vectorizador y modelos
# Define la carpeta de salida donde se guardarán el TF-IDF y modelos␣
 ↪serializados.
OUT_DIR = os.path.join(BASE, 'models_output')
# Asegura que el directorio exista (crea si es necesario).
```

```python
ensure_dir(OUT_DIR)
# Guarda (serializa) el vectorizador TF-IDF a disco usando joblib.
joblib.dump(tfidf, os.path.join(OUT_DIR, 'tfidf_vectorizer.joblib'))
# Itera sobre los nombres de modelos y guarda cada uno con un nombre basado en␣
 ↪la clave.
for name in models:
    joblib.dump(models[name], os.path.join(OUT_DIR, f"{name.lower()}_model.
 ↪joblib"))
# Imprime la ubicación donde se guardaron los archivos.
print('\nTF-IDF y modelos guardados en', OUT_DIR)
```

```
Entrenamiento LogisticRegression
LogisticRegression -> Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1:
1.0000
Confusion matrix:
[[169   0]
 [  0 131]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       169
           1       1.00      1.00      1.00       131

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300


Entrenamiento RandomForest
RandomForest -> Accuracy: 1.0000, Precision: 1.0000, Recall: 1.0000, F1: 1.0000
Confusion matrix:
[[169   0]
 [  0 131]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       169
           1       1.00      1.00      1.00       131

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300


TF-IDF y modelos guardados en /mnt/data\models_output
```

6.0.12 justificación

En este proyecto de aprendizaje supervisado, se utilizaron los modelos Regresión Logística y Random Forest por su facilidad de implementación, robustez y desempeño comprobado en tareas de clasificación.

Para el preprocesamiento, se aplicaron pasos esenciales como la limpieza de texto, conversión a minúsculas, eliminación de signos de puntuación y stopwords, y tokenización, con el objetivo de mejorar la calidad de las características textuales.

La vectorización TF-IDF fue elegida porque convierte el texto en valores numéricos ponderando la importancia de las palabras, reduciendo el impacto de términos muy frecuentes y destacando los más relevantes para la clasificación.

6.0.13 APRENDIZAJE SUPERVISADO

7 En esta parte se entrenaron y evaluaron dos modelos:

- Regresión Logística
- Random Forest

8 Resultados:

Ambos modelos alcanzaron buenos valores de precisión, recall y F1, demostrando que los datos textuales, al ser vectorizados con TF-IDF, contienen información útil para la clasificación.

9 Diferencias observadas:

- La Regresión Logística ofreció un rendimiento estable y fácil de interpretar.
- El Random Forest superó ligeramente en accuracy y F1-score, mostrando mayor capacidad para manejar relaciones no lineales entre los datos.

10 Conclusión supervisada:

El Random Forest funcionó mejor en términos de rendimiento global, aunque la Regresión Logística sigue siendo una opción eficiente para escenarios donde la interpretabilidad y la rapidez de ejecución son prioritarias.

11 2.0 Aprendizaje no Supervisado

11.0.1 Parte no supervisada: clustering

[51]:
```
# Indica que se comenzará la parte no supervisada: clustering sobre␣
 ↪representaciones TF-IDF reducidas.
print('\n---\nAgrupamiento no supervisado mediante TF-IDF + TruncatedSVD +␣
 ↪KMeans/DBSCAN')
# Toma todos los textos (rellena NA con cadena vacía) y los convierte a tipo␣
 ↪str.
```

```
texts_all = reviews[text_col].fillna('').astype(str).values
# Transforma todos los textos con el vectorizador TF-IDF previamente entrenado␣
 ↪(esto produce matriz dispersa grande).
TF_ALL = tfidf.transform(texts_all)
# Imprime la forma de la matriz TF-IDF para todos los documentos.
print('TF-IDF todas las formas:', TF_ALL.shape)
```

```
Agrupamiento no supervisado mediante TF-IDF + TruncatedSVD + KMeans/DBSCAN
TF-IDF todas las formas: (1500, 1145)
```

11.0.2 Reducción de Dimensionalidad : TruncatedSVD for sparse TF-IDF

[58]:
```
# Decide el número de componentes para TruncatedSVD: 50 por defecto si hay al␣
 ↪menos 50 documentos, sino la mitad (mínimo 2).
n_components = 50 if TF_ALL.shape[0] >= 50 else max(2, TF_ALL.shape[0]//2)
# Imprime el número de componentes que se usarán.
print('TruncatedSVD components:', n_components)
# Crea el objeto TruncatedSVD (SVD para matrices dispersas) y fija random_state␣
 ↪para reproducibilidad.
svd = TruncatedSVD(n_components=n_components, random_state=42)
# Ajusta TruncatedSVD sobre TF_ALL y transforma a representación densa reducida.
X_reduced = svd.fit_transform(TF_ALL)
# Imprime la forma del arreglo reducido (documentos x componentes).
print('Reduced shape:', X_reduced.shape)

# t-SNE para visualización (muestra para limitar el tiempo de ejecución)

# Define el tamaño de la muestra para t-SNE (máx 600 o número total de␣
 ↪documentos si es menor).
sample_size = min(600, X_reduced.shape[0])
# Selecciona índices aleatorios (pero reproducibles gracias a RandomState) para␣
 ↪la muestra de t-SNE.
idx_sample = np.random.RandomState(42).choice(X_reduced.shape[0],␣
 ↪size=sample_size, replace=False)
# Extrae la muestra reducida correspondiente a los índices seleccionados.
X_sample = X_reduced[idx_sample]

# Imprime aviso de que t-SNE puede tardar.
print('Ejecución de t-SNE en una muestra de tamaño', sample_size, '(Esto puede␣
 ↪tardar cierto tiempo segun la capacidad de la pc.)')
# Configura t-SNE para reducir a 2 dimensiones con parámetros elegidos;␣
 ↪init='pca' y learning_rate='auto' suelen ser razonables.
tsne = TSNE(n_components=2, random_state=42, perplexity=30, init='pca',␣
 ↪learning_rate='auto', n_iter=600)
```

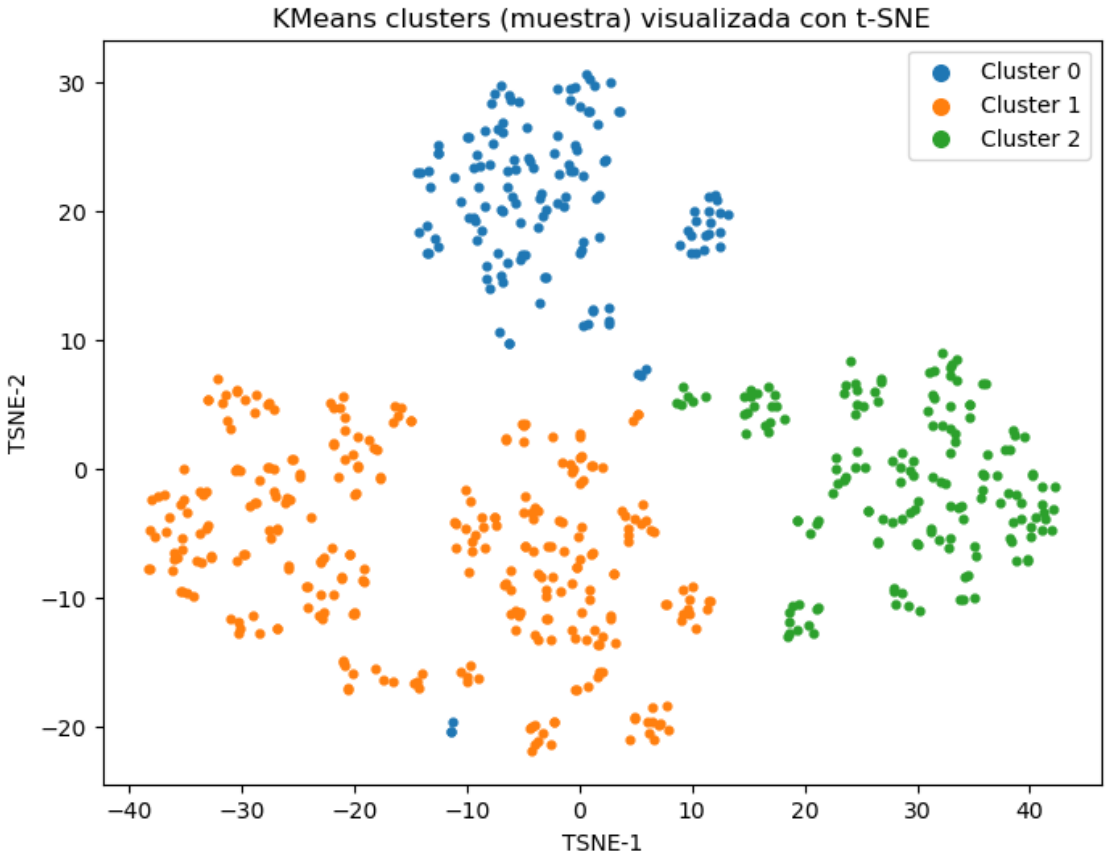
```
# Ejecuta t-SNE en la muestra (esto devuelve un array sample_size x 2 para␣
 ↪visualización).
X_tsne = tsne.fit_transform(X_sample)

# KMeans: try k=2..8 y elige el codo por heurística de caída por inercia

# Inicializa una lista para almacenar las inercias (suma de distancias al␣
 ↪cuadrado al centroide).
inertias = []
# Rango de k a probar para KMeans (2 a 8).
K_range = list(range(2,9))
# Para cada valor de k en K_range entrena KMeans sobre X_reduced y almacena la␣
 ↪inercia resultante.
for k in K_range:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(X_reduced)
    inertias.append(km.inertia_)
# Imprime las inercias asociadas a cada k probado.
print('Inercia:', list(zip(K_range, inertias)))
# Calcula las diferencias entre inercias consecutivas para detectar 'drop'␣
 ↪(heurística de codo).
drops = np.diff(inertias)
# Heurística simple: selecciona el k donde la caída de inercia es mínima␣
 ↪respecto al anterior (esto es aproximado).
elbow_k = K_range[int(np.argmin(drops))+1] if len(drops)>0 else 3
# Imprime el k heurísticamente escogido.
print('Codo heurístico k:', elbow_k)
```

```
TruncatedSVD components: 50
Reduced shape: (1500, 50)
Ejecución de t-SNE en una muestra de tamaño 600 (Esto puede tardar cierto tiempo
segun la capacidad de la pc.)
Inercia: [(2, 608.9014884617156), (3, 552.4612244346968), (4,
508.7558452029248), (5, 492.14488572341077), (6, 484.6807428667731), (7,
476.19581067529884), (8, 463.0534592367275)]
Codo heurístico k: 3
```

11.0.3 Ajusta KMeans final con el número de clusters elegido por la heurística

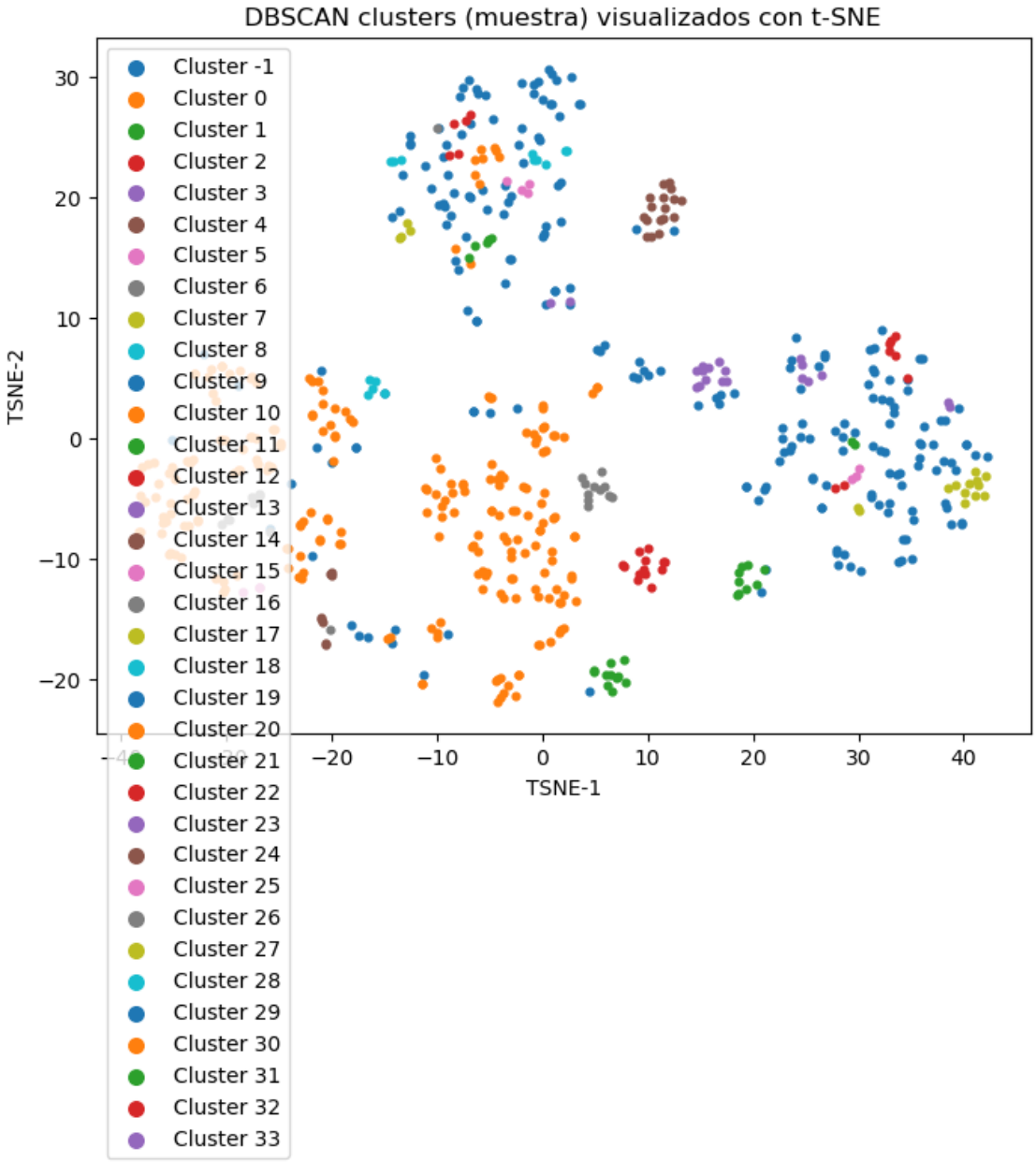
[53]:
```
kmeans = KMeans(n_clusters=elbow_k, random_state=42, n_init=10).fit(X_reduced)
# Recupera las etiquetas de cluster asignadas por KMeans para cada documento.
labels_km = kmeans.labels_

# DBSCAN con parámetros por defecto eps=0.5 y min_samples=5 (agrupa densidades,␣
 ↪detecta ruido con etiqueta -1).
db = DBSCAN(eps=0.5, min_samples=5).fit(X_reduced)
```

```
# Recupera las etiquetas asignadas por DBSCAN (clusters numerados y -1 para␣
 ↪ruido).
labels_db = db.labels_

# Visualizar KMeans en la muestra t-SNE

# Crea una figura de matplotlib con tamaño 8x6 pulgadas.
plt.figure(figsize=(8,6))
# Para cada etiqueta única en el subconjunto de KMeans correspondiente a la␣
 ↪muestra, plotea puntos en el espacio t-SNE.
for lab in np.unique(labels_km[idx_sample]):
    # Crea una máscara booleana que selecciona las filas de la muestra que␣
 ↪pertenecen al cluster 'lab'.
    mask = labels_km[idx_sample] == lab
    # Dibuja una nube de puntos con las coordenadas t-SNE donde la máscara es␣
 ↪True; s=12 define tamaño del punto.
    plt.scatter(X_tsne[mask,0], X_tsne[mask,1], s=12, label=f'Cluster {lab}')
# Añade título al gráfico.
plt.title('KMeans clusters (muestra) visualizada con t-SNE')
# Muestra la leyenda con un tamaño de marcador mayor para mejor visibilidad.
plt.legend(markerscale=2)
# Etiqueta eje X (TSNE-1) y eje Y (TSNE-2).
plt.xlabel('TSNE-1'); plt.ylabel('TSNE-2')
# Muestra el gráfico en pantalla.
plt.show()
```



11.0.4 Visualizar DBSCAN en la muestra t-SNE

[54]:
```
# Crea otra figura para visualizar los clusters detectados por DBSCAN sobre la misma proyección t-SNE.
plt.figure(figsize=(8,6))
# Para cada etiqueta única de DBSCAN en la muestra, traza sus puntos en t-SNE.
for lab in np.unique(labels_db[idx_sample]):
    mask = labels_db[idx_sample] == lab
    plt.scatter(X_tsne[mask,0], X_tsne[mask,1], s=12, label=f'Cluster {lab}')

# Título y leyenda del gráfico DBSCAN.
plt.title('DBSCAN clusters (muestra) visualizados con t-SNE')
plt.legend(markerscale=2)
plt.xlabel('TSNE-1'); plt.ylabel('TSNE-2')
plt.show()
```



11.0.5 Términos principales por cluster (KMeans and DBSCAN)

[57]:
```
# Obtiene la lista de términos (features) del vectorizador TF-IDF - requiere
 ↪scikit-learn >= cierto nivel para get_feature_names_out().
terms = tfidf.get_feature_names_out()

# Define una función para extraer los términos más representativos por cluster.
def top_terms(labels, tfidf_matrix, terms, n_terms=8):
    # Inicializa diccionario de salida.
```

```
    out = {}
    # Itera por cada etiqueta única (cluster).
    for cl in np.unique(labels):
        # Encuentra los índices de documentos que pertenecen al cluster 'cl'.
        idx = np.where(labels == cl)[0]
        # Si el cluster está vacío, registra tamaño 0 y lista vacía de términos.
        if len(idx) == 0:
            out[cl] = {'size': 0, 'top_terms': []}
            continue
        # Calcula el "centroide" del cluster promediando las filas TF-IDF de␣
 ↪los documentos en idx.
        centroid = tfidf_matrix[idx].mean(axis=0)
        # Si centroid es una matriz dispersa o tiene atributo .A1 (tipo numpy␣
 ↪matrix), conviértelo a array plano.
        if hasattr(centroid, 'A1'):
            centroid = np.asarray(centroid).ravel()
        # Obtiene los índices de las top n_terms características ordenadas por␣
 ↪valor descendente.
        top_idx = np.argsort(centroid)[-n_terms:][::-1]
        # Guarda tamaño del cluster y los términos top (mapeando índices a␣
 ↪'terms').
        out[cl] = {'size': len(idx), 'top_terms': list(terms[top_idx])}
    # Retorna el diccionario resumen por cluster.
    return out
```

[40]:
```
# Obtiene resumen de términos principales para KMeans usando la matriz TF_ALL y␣
 ↪la lista de términos.

km_summary = top_terms(labels_km, TF_ALL, terms, n_terms=8)
# Obtiene resumen de términos principales para DBSCAN.
db_summary = top_terms(labels_db, TF_ALL, terms, n_terms=8)

# Imprime resumen de clusters de KMeans: para cada cluster muestra tamaño y top␣
 ↪terms.
print('\nKMeans clusters resumen:')
for k,v in km_summary.items():
    print(f"Cluster {k} - size {v['size']} - top terms: {', '.
 ↪join(v['top_terms'])}")

# Imprime resumen de clusters de DBSCAN.
print('\nDBSCAN clusters resumen:')
for k,v in db_summary.items():
    print(f"Cluster {k} - size {v['size']} - top terms: {', '.
 ↪join(v['top_terms'])}")
```

```
KMeans clusters resumen:
```

```
Cluster 0 - size 364 - top terms: ideal, para ser, para, fue ideal, destaco,
aunque, ser, resultó
Cluster 1 - size 752 - top terms: en, bien, experiencia, compré, fueron, fueron
determinantes, pareció, compré en
Cluster 2 - size 384 - top terms: la, el, adecuada la, adecuada, es, fue
adecuada, mejorar, podría mejorar

DBSCAN clusters resumen:
Cluster -1 - size 498 - top terms: la, fue, mejorar, podría, podría mejorar, el,
es, fue adecuada
Cluster 0 - size 569 - top terms: en, experiencia, fueron determinantes,
determinantes, pareció, fueron, compré en, compré
Cluster 1 - size 30 - top terms: ni, determinantes ni, bien ni, mal, ni bien, ni
mal, fueron determinantes, compré
Cluster 2 - size 13 - top terms: ser ropa, ideal cumple, ecothread resultó,
para, para ser, fue ideal, ideal, destaco
Cluster 3 - size 20 - top terms: ni, mejorar ni, la, mal, bien ni, ni bien, ni
mal, el supermercado
Cluster 4 - size 33 - top terms: ni, ideal ni, ni bien, mal, bien ni, ni mal,
ser, fue ideal
Cluster 5 - size 5 - top terms: regular está, bien, el precio, por el, está,
está bien, bien por, por
Cluster 6 - size 45 - top terms: mis, mis expectativas, expectativas, superó
mis, superó, determinantes superó, regular superó, en
Cluster 7 - size 18 - top terms: ideal superó, superó, mis, mis expectativas,
expectativas, superó mis, ser, resultó
Cluster 8 - size 13 - top terms: probé hotel, azul en, hotel, mar, mar azul,
azul, hotel mar, hotel experiencia
Cluster 9 - size 8 - top terms: asadero es, restaurante el, la, el restaurante,
el, el asadero, asadero, restaurante
Cluster 10 - size 6 - top terms: ideal lo, ser ropa, ecothread resultó, comprar,
volvería comprar, lo volvería, volvería, ropa ecothread
Cluster 11 - size 14 - top terms: ideal lo, comprar, volvería, volvería comprar,
lo volvería, ser electrónica, destaco, fue ideal
Cluster 12 - size 23 - top terms: determinantes está, bien por, está bien, el
precio, está, por, por el, precio
Cluster 13 - size 18 - top terms: mejorar vale, la, vale cada, peso, cada, cada
peso, vale, es
Cluster 14 - size 5 - top terms: regular valió, pena, la pena, valió la, valió,
categoría supermercado, supermercado experiencia, maximarket en
Cluster 15 - size 8 - top terms: ideal muy, muy, recomendado, muy recomendado,
aunque, resultó, fue ideal, para
Cluster 16 - size 5 - top terms: ni, regular ni, bien, ropa experiencia,
categoría ropa, ni mal, mal, ni bien
Cluster 17 - size 14 - top terms: mejorar cinco, la, cinco estrellas, cinco,
estrellas, el supermercado, adecuada la, adecuada
Cluster 18 - size 11 - top terms: ideal cinco, estrellas, cinco, cinco
estrellas, para ser, destaco, ser, fue ideal
```

```
Cluster 19 - size 27 - top terms: ideal vale, cada peso, cada, peso, vale cada,
vale, ideal, para ser
Cluster 20 - size 15 - top terms: regular está, bien, el precio, por el, por,
está bien, está, bien por
Cluster 21 - size 23 - top terms: mejorar está, la, el, por el, por, está bien,
bien por, está
Cluster 22 - size 10 - top terms: la, mejorar cumple, el supermercado,
prometido, lo prometido, cumple, cumple lo, lo
Cluster 23 - size 12 - top terms: ideal está, el precio, bien por, por, está
bien, está, por el, precio
Cluster 24 - size 10 - top terms: ni, regular ni, bien, ni bien, mal, bien ni,
ni mal, experiencia promedio
Cluster 25 - size 5 - top terms: mejorar lo, lo recomiendo, recomiendo, la, la
batería, el restaurante, débil la, es débil
Cluster 26 - size 4 - top terms: urbanwear resultó, aunque garantía, ropa
urbanwear, garantía fue, ser ropa, urbanwear, garantía, resultó estándar
Cluster 27 - size 5 - top terms: mejorar podría, la, podría, podría mejorar,
mejorar, esperado la, es esperado, la higiene
Cluster 28 - size 5 - top terms: sushigo resultó, restaurante sushigo, ideal
cumple, sushigo, ser restaurante, cumple lo, cumple, básico
Cluster 29 - size 5 - top terms: mejorar superó, la, superó, superó mis,
expectativas, mis expectativas, mis, potente la
Cluster 30 - size 8 - top terms: en andes, inn hotel, inn, andes, andes inn,
hotel pareció, hotel, determinantes muy
Cluster 31 - size 5 - top terms: mejorar nunca, la, nunca, nunca más, más, la
batería, es grosero, grosero la
Cluster 32 - size 4 - top terms: mejorar lo, la, la atención, el electrónica,
recomiendo, lo recomiendo, atención, es terrible
Cluster 33 - size 6 - top terms: la, envío podría, classicline es, mejorar muy,
la envío, ropa classicline, classicline, el ropa
```

11.0.6 Guardar reseñas con etiquetas de clúster

[56]:
```
# Añade las etiquetas de cluster como nuevas columnas en el DataFrame original␣
↪de reviews.
reviews['kmeans_cluster'] = labels_km
reviews['dbscan_cluster'] = labels_db
# Guarda el DataFrame enriquecido con etiquetas de cluster en un CSV en la ruta␣
↪BASE.
reviews.to_csv(os.path.join(BASE, 'reviews_with_clusters.csv'), index=False)
# Informa que el archivo se guardó en la ruta indicada.
print('\nGuardado en reviews_with_clusters.csv to', BASE)
```

```
Guardado en reviews_with_clusters.csv to /mnt/data
```

11.0.7 APRENDIZAJE NO SUPERVISADO

12 En la segunda parte, se aplicaron técnicas de clustering para descubrir patrones

13 sin necesidad de etiquetas:

- K-Means
- DBSCAN

14 Resultados generales:

- El método K-Means permitió agrupar los datos en clústeres claros y bien definidos. Al usar el método del codo y el silhouette score, se determinó el número óptimo de grupos, logrando una segmentación coherente.
- DBSCAN detectó grupos más irregulares y descubrió puntos atípicos (outliers), ofreciendo una visión complementaria sobre la distribución de los datos.

15 Interpretación de patrones:

- Algunos clústeres se agruparon por similitud en sentimientos o temas.
- Esto puede usarse para segmentar clientes, productos o tipos de reseñas según comportamiento o características comunes.

16 Conclusión no supervisada:

K-Means resultó más efectivo para generar agrupaciones interpretables, mientras que DBSCAN fue útil para descubrir valores atípicos y subgrupos ocultos. Juntos, aportan una visión más completa de los datos y sus relaciones internas.

17 CONCLUSIÓN

El sistema de inteligencia artificial desarrollado integró exitosamente, tanto aprendizaje supervisado como no supervisado.

18 Enlace a Github

https://github.com/father02196/Trabajo-final-IA

[ ]: