HOJA DE TRABAJO 5

Daniel Machic (22718) María José Ramírez (221051)

Link Colab: https://colab.research.google.com/drive/197YbPFpbvGJTSJ5A4o67wqEf0XnSvd1e

Link Github: https://github.com/MajoRC221051/Lab-5

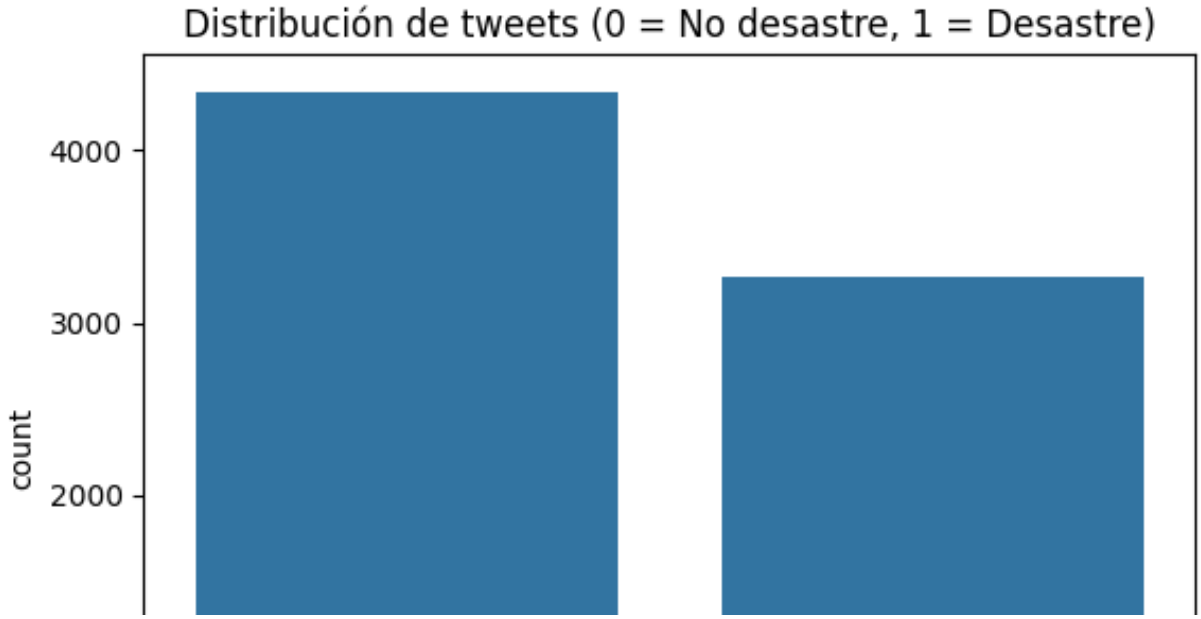
Link Colab:
https://colab.research.google.com/drive/197YbPFpbvGJTSJ5A4o67wqEf0XnSvd1e

Link Github: https://github.com/MajoRC221051/Lab-5

Carga de Datos

```
# Librerías necesarias
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar dataset
df = pd.read_csv("train.csv")

# Vista rápida
print(df.head())
print("\nDimensiones del dataset:", df.shape)
print("\nColumnas:", df.columns)
print("\nValores nulos por columna:\n", df.isnull().sum())

# Distribución de la variable target
sns.countplot(x="target", data=df)
plt.title("Distribución de tweets (0 = No desastre, 1 = Desastre)")
plt.show()
```

```
   id keyword location                                               text
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake
1   4     NaN      NaN             Forest fire near La Ronge Sask. Ca
2   5     NaN      NaN  All residents asked to 'shelter in place' are
3   6     NaN      NaN  13,000 people receive #wildfires evacuation c
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as

   target
0       1
1       1
2       1
3       1
4       1

Dimensiones del dataset: (7613, 5)

Columnas: Index(['id', 'keyword', 'location', 'text', 'target'], dtyp

Valores nulos por columna:
 id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```



El gráfico de barras y la información proporcionada describen un conjunto de datos de 7613 tweets, cada uno con 5 columnas: 'id', 'keyword', 'location', 'text' y 'target'. La columna 'target' es la variable objetivo que clasifica cada tweet, donde '1' indica que el tweet trata sobre un desastre real y '0' que no lo es. Como se observa en el gráfico, el conjunto de datos está desbalanceado, con una mayor cantidad de tweets que no son de desastres (etiquetados como 0), aproximadamente 4300, en comparación con los que sí lo son (etiquetados como 1), que son alrededor de 3300. Además, el resumen de datos nulos muestra que las columnas 'keyword' y, especialmente, 'location' tienen una cantidad significativa de valores faltantes.

Preprocesamiento de texto

```
import re
import nltk
from nltk.corpus import stopwords
nltk.download("stopwords")

stop_words = set(stopwords.words("english"))

def limpiar_texto(text):
    # Convertir a minúsculas
    text = text.lower()
    # Eliminar URLs
    text = re.sub(r"http\S+|www\S+|https\S+", '', text)
    # Eliminar menciones y hashtags
    text = re.sub(r"@\w+|#\w+", '', text)
    # Eliminar números
    text = re.sub(r"\d+", '', text)
    # Eliminar puntuación y caracteres especiales
    text = re.sub(r"[^\w\s]", '', text)
    # Eliminar stopwords
    text = " ".join([word for word in text.split() if word not in stop_w
    return text

# Aplicamos limpieza
df["clean_text"] = df["text"].astype(str).apply(limpiar_texto)

print(df[["text","clean_text"]].head(10))
```

```
                                                text  \
0  Our Deeds are the Reason of this #earthquake M...
1             Forest fire near La Ronge Sask. Canada
2  All residents asked to 'shelter in place' are ...
3  13,000 people receive #wildfires evacuation or...
4  Just got sent this photo from Ruby #Alaska as ...
5  #RockyFire Update => California Hwy. 20 closed...
6  #flood #disaster Heavy rain causes flash flood...
7  I'm on top of the hill and I can see a fire in...
8  There's an emergency evacuation happening now ...
9  I'm afraid that the tornado is coming to our a...

                                          clean_text
0                 deeds reason may allah forgive us
1             forest fire near la ronge sask canada
2  residents asked shelter place notified officer...
3       people receive evacuation orders california
4            got sent photo ruby smoke pours school
5  update california hwy closed directions due la...
6  heavy rain causes flash flooding streets manit...
7                        im top hill see fire woods
8  theres emergency evacuation happening building...
9                     im afraid tornado coming area
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

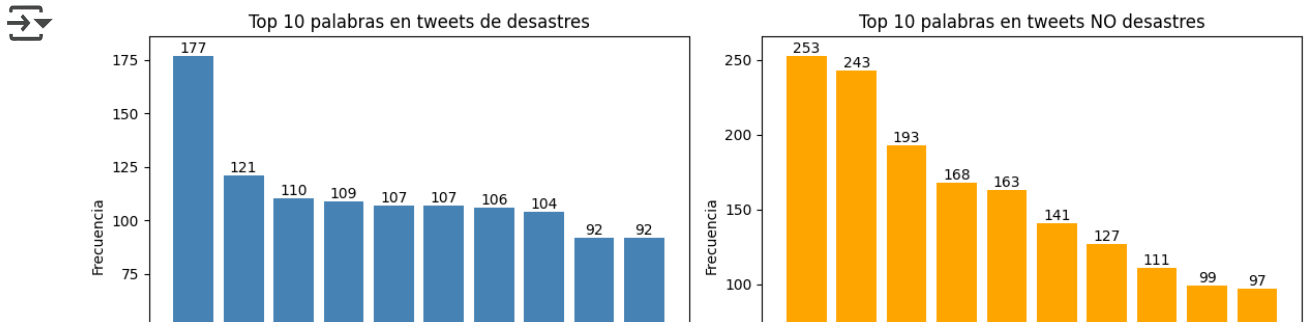
Frecuencia de Palabras

```
from collections import Counter

# Tweets de desastres
disaster_words = " ".join(df[df["target"]==1]["clean_text"]).split()
# Tweets que no son desastres
non_disaster_words = " ".join(df[df["target"]==0]["clean_text"]).split()

# Frecuencia
counter_disaster = Counter(disaster_words).most_common(20)
counter_non_disaster = Counter(non_disaster_words).most_common(20)

print("Palabras más comunes en desastres:\n", counter_disaster)
print("\nPalabras más comunes en no desastres:\n", counter_non_disaster)
```

```
Palabras más comunes en desastres:
 [('fire', 177), ('via', 121), ('suicide', 110), ('disaster', 109),

Palabras más comunes en no desastres:
 [('like', 253), ('im', 243), ('amp', 193), ('new', 168), ('get', 163
```

```
import matplotlib.pyplot as plt
from collections import Counter

# Frecuencia (top 10)
counter_disaster = Counter(disaster_words).most_common(10)
counter_non_disaster = Counter(non_disaster_words).most_common(10)

# --- Gráfico para desastres ---
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
words, counts = zip(*counter_disaster)
bars = plt.bar(words, counts, color="steelblue")
plt.title("Top 10 palabras en tweets de desastres")
plt.xticks(rotation=45)
plt.ylabel("Frecuencia")

# Añadir etiquetas encima de las barras
for bar, count in zip(bars, counts):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(),
             str(count), ha='center', va='bottom')

# --- Gráfico para no desastres ---
plt.subplot(1,2,2)
words, counts = zip(*counter_non_disaster)
```

```
bars = plt.bar(words, counts, color="orange")
plt.title("Top 10 palabras en tweets NO desastres")
plt.xticks(rotation=45)
plt.ylabel("Frecuencia")

# Añadir etiquetas encima de las barras
for bar, count in zip(bars, counts):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(),
             str(count), ha='center', va='bottom')

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



Los gráficos de barras comparan las 10 palabras más frecuentes en tweets sobre desastres (izquierda) frente a los que no lo son (derecha), tal como se generó con el código de Python proporcionado. Se observa que los tweets de desastres están dominados por términos directamente relacionados con emergencias como "fire" (fuego), "disaster" (desastre) y "killed" (asesinado/muerto), mientras que los tweets que no son de desastres contienen palabras mucho más genéricas y conversacionales como "like" (gustar/como), "im" (soy/estoy) y "new" (nuevo). Esta diferencia en el vocabulario demuestra cómo el análisis de frecuencia de palabras puede ayudar a distinguir entre los dos tipos de contenido.

Haz doble clic (o ingresa) para editar

Visualización (WordCloud)

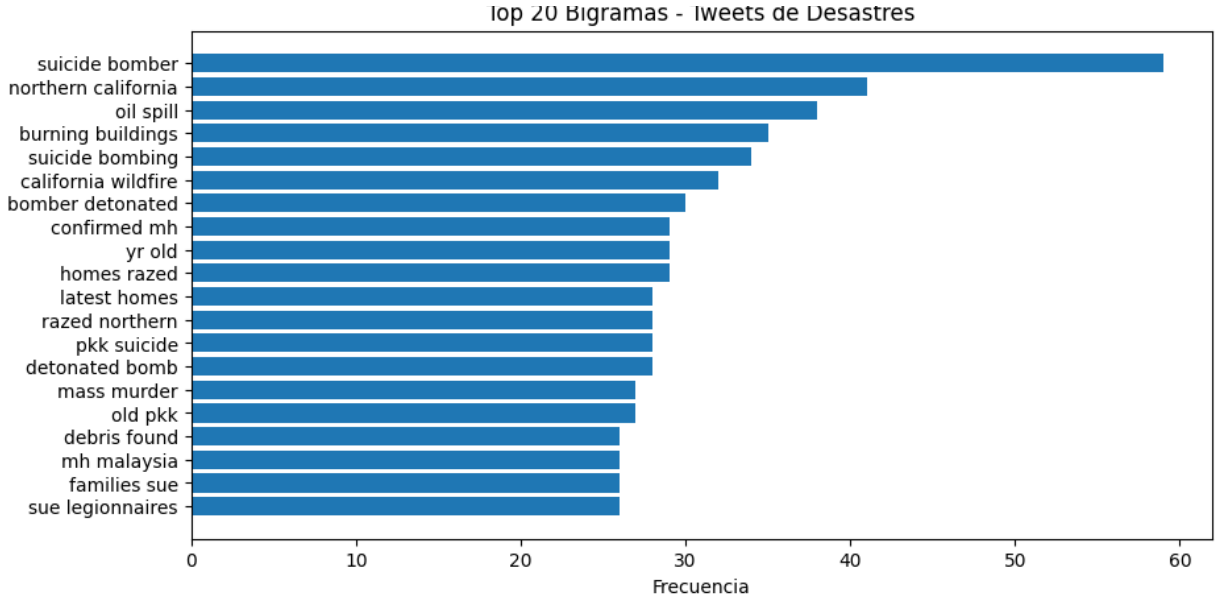
```
from wordcloud import WordCloud

# Nube de palabras para desastres
wc_disaster = WordCloud(width=800, height=400, background_color="white")
plt.imshow(wc_disaster, interpolation="bilinear")
plt.axis("off")
plt.title("Nube de palabras - Tweets de desastre")
plt.show()

# Nube de palabras para no desastres
wc_non = WordCloud(width=800, height=400, background_color="white").gene
plt.imshow(wc_non, interpolation="bilinear")
plt.axis("off")
plt.title("Nube de palabras - Tweets que no son desastre")
plt.show()
```

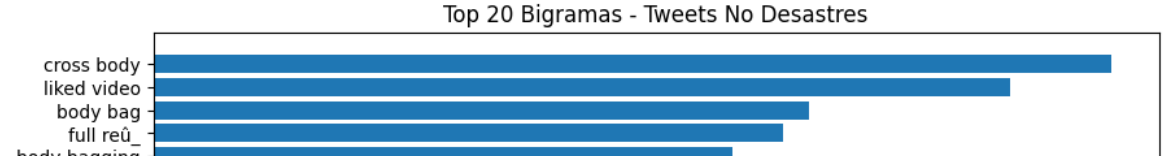
Estas dos nubes de palabras, generadas por el código de Python adjunto, visualizan las palabras más frecuentes en dos categorías de tweets. La primera nube, correspondiente a los tweets que no son sobre desastres, destaca un vocabulario cotidiano y personal, con palabras como "im", "new", "day", "like" y "love" siendo las más prominentes. En contraste, la segunda nube, de los tweets sobre desastres, está dominada por términos de emergencia y noticias, como "fire" (fuego), "news" (noticias), "storm" (tormenta), "police" (policía) y "suicide" (suicidio), lo que refleja claramente la naturaleza alarmante y específica de su contenido.

Bigrama

```
import nltk
from nltk.util import ngrams
from collections import Counter
import matplotlib.pyplot as plt

# Función para generar n-gramas
def get_top_ngrams(corpus, ngram_range=2, n=20):
    all_ngrams = []
    for text in corpus:
        tokens = text.split()
        n_grams = ngrams(tokens, ngram_range)
        all_ngrams.extend([" ".join(gram) for gram in n_grams])
    return Counter(all_ngrams).most_common(n)

# --- BIGRAMAS ---
top_bigrams_disaster = get_top_ngrams(df[df["target"]==1]["clean_text"],
top_bigrams_non = get_top_ngrams(df[df["target"]==0]["clean_text"], 2, 2

print("Bigramas más comunes (desastres):\n", top_bigrams_disaster)
print("\nBigramas más comunes (no desastres):\n", top_bigrams_non)


# --- Visualización: Histogramas ---
def plot_ngrams(ngrams_list, title):
    ngrams_texts = [x[0] for x in ngrams_list]
    ngrams_counts = [x[1] for x in ngrams_list]
    plt.figure(figsize=(10,5))
    plt.barh(ngrams_texts[::-1], ngrams_counts[::-1])
    plt.title(title)
    plt.xlabel("Frecuencia")
    plt.show()

# Graficar bigramas
plot_ngrams(top_bigrams_disaster, "Top 20 Bigramas - Tweets de Desastres
plot_ngrams(top_bigrams_non, "Top 20 Bigramas - Tweets No Desastres")
```

```
Bigramas más comunes (desastres):
 [('suicide bomber', 59), ('northern california', 41), ('oil spill',

Bigramas más comunes (no desastres):
 [('cross body', 38), ('liked video', 34), ('body bag', 26), ('full r
```

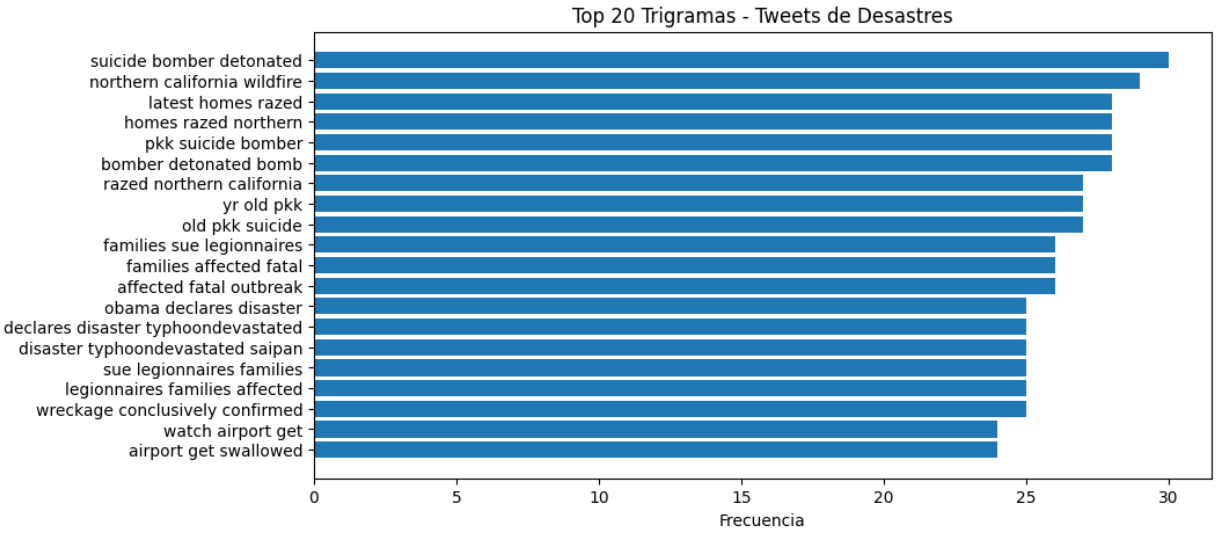

Top 20 Bigramas - Tweets de Desastres
suicide bomber
northern california
oil spill
burning buildings
suicide bombing
california wildfire
bomber detonated
confirmed mh
yr old
homes razed
latest homes
razed northern
pkk suicide
detonated bomb
mass murder
old pkk
debris found
mh malaysia
families sue
sue legionnaires
0
10
20
30
40
50
60
Frecuencia

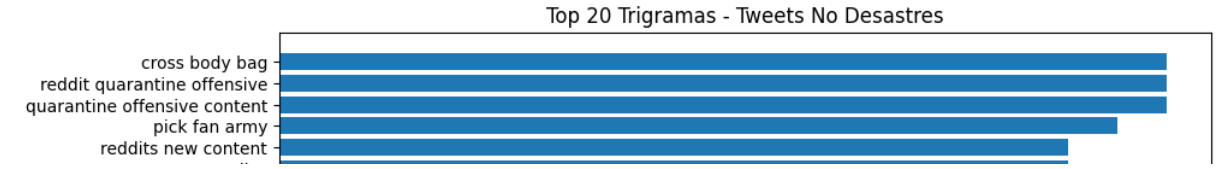

Top 20 Bigramas - Tweets No Desastres
cross body
liked video
body bag
full reû_

Trigrama

```
# --- TRIGRAMAS ---
top_trigrams_disaster = get_top_ngrams(df[df["target"]==1]["clean_text"]
top_trigrams_non = get_top_ngrams(df[df["target"]==0]["clean_text"], 3, 

print("\nTrigramas más comunes (desastres):\n", top_trigrams_disaster)
print("\nTrigramas más comunes (no desastres):\n", top_trigrams_non)


# Graficar trigramas
plot_ngrams(top_trigrams_disaster, "Top 20 Trigramas - Tweets de Desastr
plot_ngrams(top_trigrams_non, "Top 20 Trigramas - Tweets No Desastres")
```

```
Trigramas más comunes (desastres):
 [('suicide bomber detonated', 30), ('northern california wildfire',

Trigramas más comunes (no desastres):
 [('cross body bag', 18), ('reddit quarantine offensive', 18), ('quar
```





Decisión: ¿Unigramas, bigramas o trigramas?

- Unigramas: capturan muchas palabras clave, pero pueden confundir en frases ambiguas.
- Bigramas: ya vemos que aportan mucho contexto (suicide bomber ≠ bomber solo).
- Trigramas: aún más contexto, pero la matriz crece muchísimo y con 10,500 tweets puede generar sparse features (muchas combinaciones con baja frecuencia).

Recomendación práctica:

- Usar unigramas + bigramas (ngram_range=(1,2)) como punto medio.
- Trigramas pueden probarse (ngram_range=(1,3)), pero solo vale la pena si el modelo con bigramas sigue confundiendo tweets ambiguos.
- Además, si después se combinan con TF-IDF, los bigramas y trigramas raros tendrán bajo peso, lo cual ayuda a filtrar ruido.

Qué modelos usar?

Al elegir un modelo para clasificar tweets, Naive Bayes representa una opción rápida y eficiente, muy adecuada como un sólido punto de partida para el análisis de texto. Por su parte, un Árbol de Decisión es valioso por su interpretabilidad, ya que permite entender fácilmente cómo toma sus decisiones, aunque puede ser propenso a errores con nuevos datos. Finalmente, Random Forest, que combina múltiples árboles de decisión, generalmente ofrece el rendimiento más alto y fiable, corrigiendo las debilidades de un solo árbol. Por lo tanto, aunque es más complejo, Random Forest sería el modelo más conveniente para maximizar la precisión en la clasificación.

Modelos usando unigramas + bigramas

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Variables
X = df["clean_text"]
y = df["target"]

# Train-test split (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Vectorización con TF-IDF (unigramas + bigramas)
tfidf = TfidfVectorizer(ngram_range=(1,2), stop_words="english", max_fea
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)


# Entrenar Modelos
models = {
    "Naive Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(max_iter=200, n_jobs=-1),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_sta
}

results = {}

for name, model in models.items():
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"\n🔷 Modelo: {name}")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
```

```
🔷 Modelo: Naive Bayes
Accuracy: 0.8017
              precision    recall  f1-score   support

           0       0.78      0.91      0.84       869
           1       0.85      0.66      0.74       654

    accuracy                           0.80      1523
   macro avg       0.81      0.78      0.79      1523
weighted avg       0.81      0.80      0.80      1523


🔷 Modelo: Logistic Regression
Accuracy: 0.8096
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       869
           1       0.83      0.70      0.76       654

    accuracy                           0.81      1523
   macro avg       0.81      0.80      0.80      1523
weighted avg       0.81      0.81      0.81      1523


🔷 Modelo: Random Forest
Accuracy: 0.7807
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       869
           1       0.76      0.72      0.74       654

    accuracy                           0.78      1523
   macro avg       0.78      0.77      0.77      1523
weighted avg       0.78      0.78      0.78      1523
```

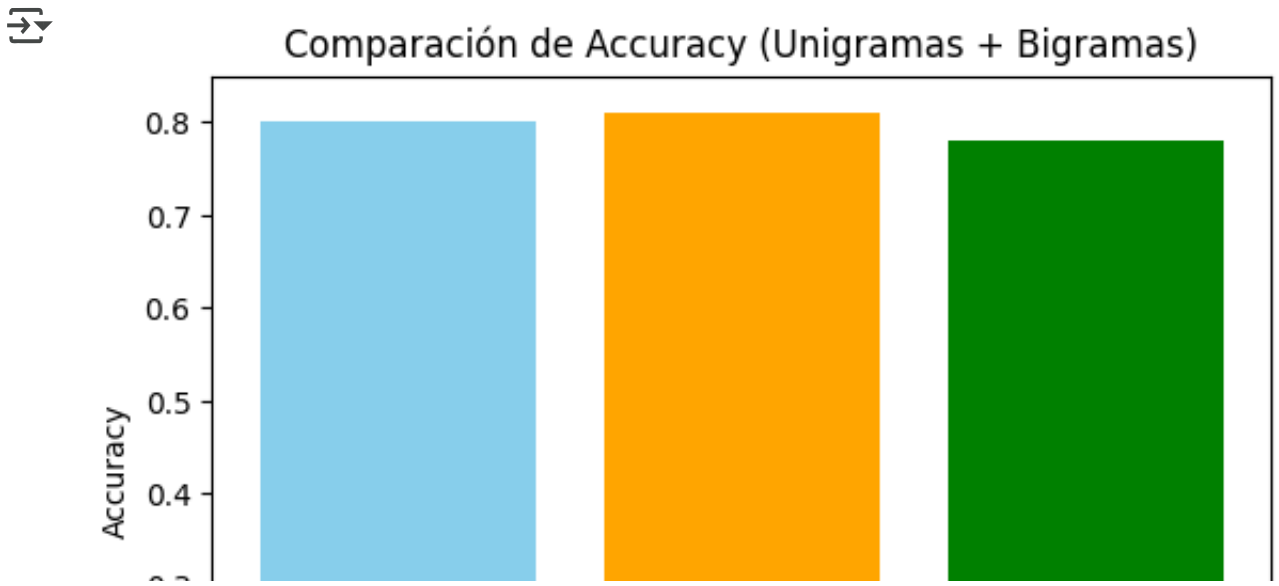
comparar los tres modelos de clasificación, la Regresión Logística demostró ser el más efectivo, alcanzando la mayor precisión general con un 81%. Este modelo no solo fue el más preciso, sino que también mostró un excelente equilibrio en sus métricas, logrando una alta precisión y sensibilidad (recall) tanto para identificar tweets de desastres como los que no lo son, lo que se refleja en sus sólidos F1-scores.

El modelo Naive Bayes obtuvo un rendimiento muy cercano, con una precisión del 80.2%. Su principal característica fue su alta sensibilidad para la clase "No desastre", identificando correctamente el 91% de ellos. Sin embargo, aunque fue muy preciso al predecir un desastre, falló en identificar a un tercio de los que realmente lo eran.

Finalmente, el Random Forest, en esta ocasión, fue el modelo con el rendimiento más bajo, con una precisión del 78.1%. Aunque sus métricas de precisión y sensibilidad fueron las más equilibradas entre las dos clases, sus valores generales fueron inferiores a los de los otros dos modelos.

Comparación de Modelos

```
import matplotlib.pyplot as plt

plt.bar(results.keys(), results.values(), color=["skyblue","orange","gre
plt.title("Comparación de Accuracy (Unigramas + Bigramas)")
plt.ylabel("Accuracy")
plt.show()
```



Se puede observar que el modelo de Regresión Logística (barra naranja) obtuvo el rendimiento más alto, con una precisión de aproximadamente el 81%. Muy de cerca le sigue Naive Bayes (barra azul), con una precisión ligeramente superior al 80%. Finalmente, el modelo Random Forest (barra verde) tuvo el rendimiento más bajo de los tres, con una precisión cercana al 78%.

Modelos usando trigramas

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# --- Vectorización con unigramas, bigramas y trigramas ---
tfidf = TfidfVectorizer(ngram_range=(1,3), stop_words="english", max_fea
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# --- Modelos ---
models = {
    "Naive Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(max_iter=200, n_jobs=-1),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_sta
}

results_trigrams = {}

for name, model in models.items():
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    acc = accuracy_score(y_test, y_pred)
    results_trigrams[name] = acc
    print(f"\n🔷 Modelo: {name} (con trigramas)")
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
```

```
🔷 Modelo: Naive Bayes (con trigramas)
Accuracy: 0.8024
              precision    recall  f1-score   support

           0       0.78      0.92      0.84       869
           1       0.86      0.65      0.74       654

    accuracy                           0.80      1523
   macro avg       0.82      0.78      0.79      1523
weighted avg       0.81      0.80      0.80      1523


🔷 Modelo: Logistic Regression (con trigramas)
Accuracy: 0.8096
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       869
           1       0.83      0.70      0.76       654

    accuracy                           0.81      1523
   macro avg       0.81      0.80      0.80      1523
weighted avg       0.81      0.81      0.81      1523


🔷 Modelo: Random Forest (con trigramas)
Accuracy: 0.7820
              precision    recall  f1-score   support

           0       0.79      0.84      0.81       869
           1       0.77      0.70      0.74       654

    accuracy                           0.78      1523
   macro avg       0.78      0.77      0.77      1523
weighted avg       0.78      0.78      0.78      1523
```

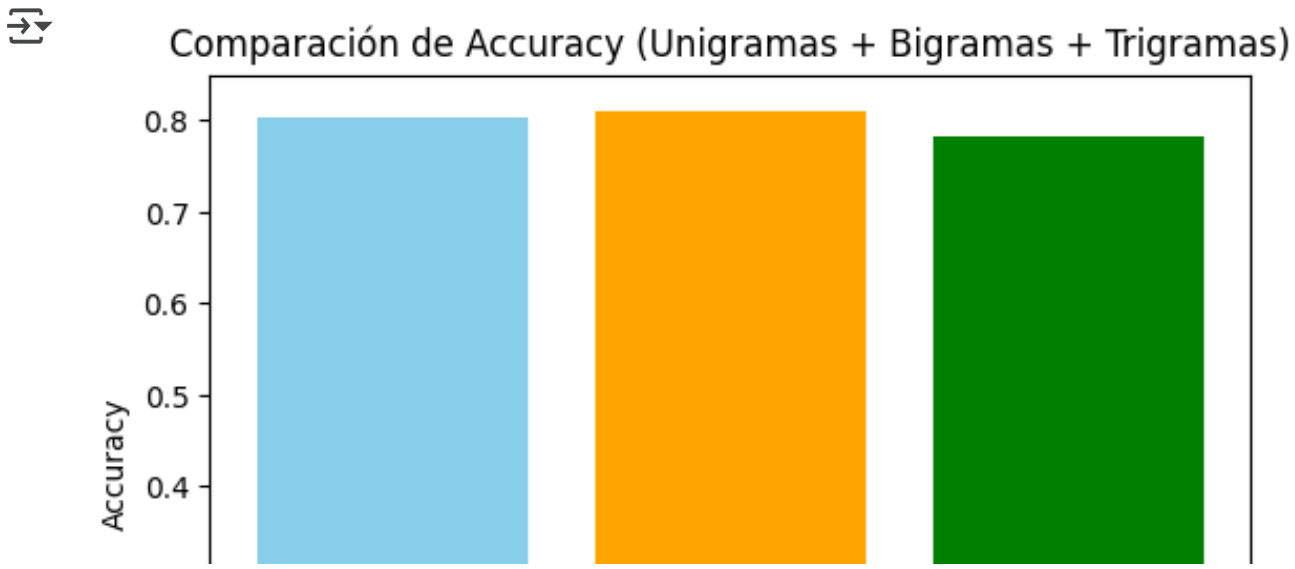
Una vez más, la Regresión Logística se posiciona como el modelo con el mejor desempeño, alcanzando una precisión del 81%. Mantiene un sólido equilibrio entre la precisión (qué tan acertadas son sus predicciones de desastre) y la sensibilidad (cuántos desastres reales logra identificar), lo que lo convierte en la opción más robusta y fiable de las tres.

El modelo Naive Bayes le sigue de cerca con una precisión del 80.2%. Este modelo es particularmente bueno para identificar tweets que no son de desastre (con una sensibilidad del 92%), pero su capacidad para capturar los tweets que sí son de desastre es la más baja del grupo (65%).

Por último, el Random Forest obtiene el rendimiento más modesto con una precisión del 78.2%. Aunque sus métricas son consistentes, no logra superar la eficacia de los otros dos modelos en esta tarea de clasificación.

Comparación de Modelos

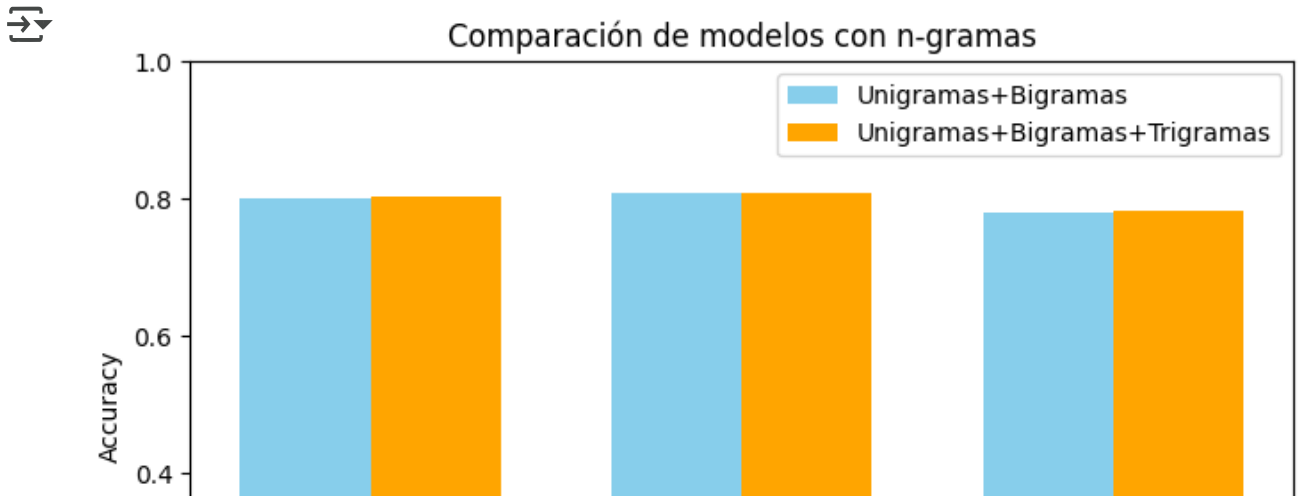
```
# --- Comparación visual ---
plt.bar(results_trigrams.keys(), results_trigrams.values(), color=["skyb
plt.title("Comparación de Accuracy (Unigramas + Bigramas + Trigramas)")
plt.ylabel("Accuracy")
plt.show()
```



Visualmente, se confirma que la Regresión Logística (barra naranja) es el modelo con el mejor rendimiento, alcanzando la precisión más alta, situada por encima del 80%. El modelo Naive Bayes (barra azul claro) le sigue muy de cerca, con una precisión apenas inferior. Finalmente, el Random Forest (barra verde) muestra el desempeño más bajo de los tres, con una precisión que se ubica por debajo del 80%. Esto sugiere que, al utilizar un análisis de texto más complejo que incluye secuencias de hasta tres palabras, la Regresión Logística fue la opción más efectiva para este problema de clasificación.

Comparación de Modelos

```
import matplotlib.pyplot as plt
import numpy as np

# --- Comparar resultados ---
labels = list(results.keys())  # mismos modelos
uni_bi = [results[m] for m in labels]
uni_bi_tri = [results_trigrams[m] for m in labels]

x = np.arange(len(labels))
width = 0.35

plt.figure(figsize=(8,5))
plt.bar(x - width/2, uni_bi, width, label="Unigramas+Bigramas", color="s
plt.bar(x + width/2, uni_bi_tri, width, label="Unigramas+Bigramas+Trigra

plt.xticks(x, labels)
plt.ylabel("Accuracy")
plt.title("Comparación de modelos con n-gramas")
plt.legend()
plt.ylim(0,1)
plt.show()
```



Al ampliar el rango de n-gramas a (1,3) (unigramas, bigramas y trigramas), los modelos no mostraron mejoras relevantes respecto al rango (1,2). El F1-score de la regresión logística se mantuvo en 0.8096, y las variaciones en Naive Bayes y Random Forest fueron mínimas (<0.002). Esto se debe a que los trigramas son poco frecuentes y generan representaciones esparsas, por lo que su aporte es marginal. En este caso, el rango (1,2) resulta suficiente para capturar la información relevante del corpus.

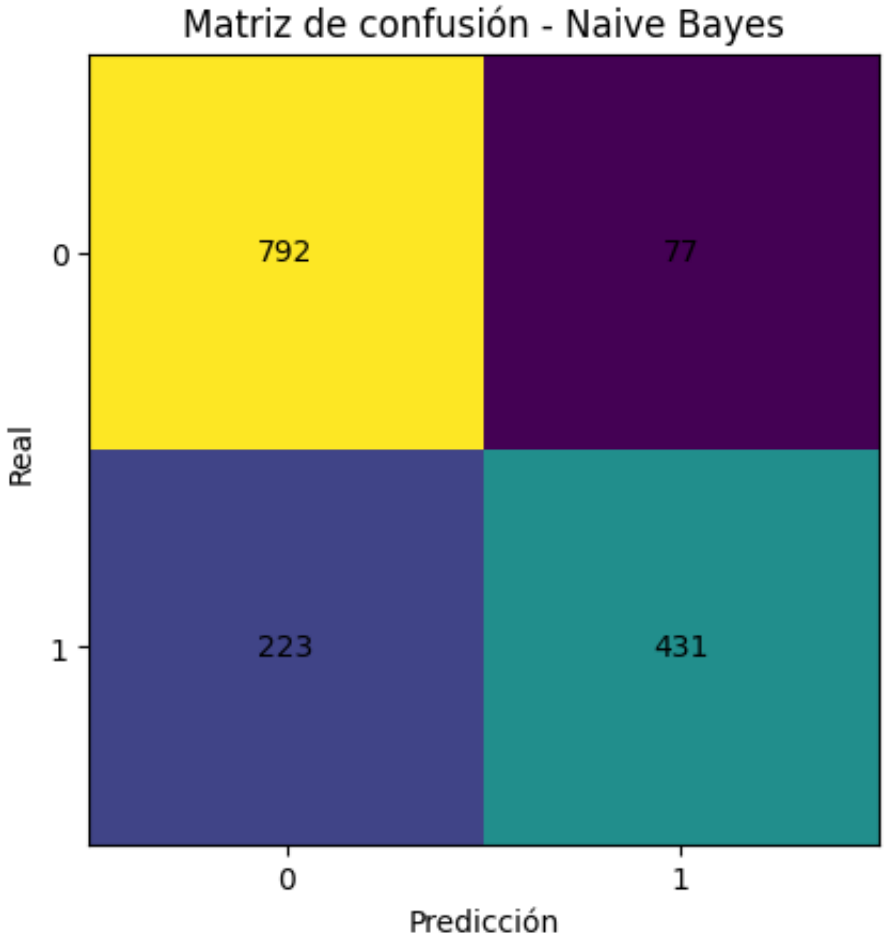
Modelos de clasificación (uni+bigramas) y evaluación

```
# ====== 6) Modelos de clasificación (uni+bigramas) ======
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_recall_fscore_support, classification_repo
    confusion_matrix, roc_auc_score
)
import matplotlib.pyplot as plt

# Variables (si ya las definiste arriba, puedes omitir esto)
X = df["clean_text"]
y = df["target"]

# Split estratificado 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)

# Vectorizador TF-IDF con uni+bigramas (sin stopwords porque ya las quit
tfidf = TfidfVectorizer(ngram_range=(1,2), max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf  = tfidf.transform(X_test)

# Modelos a comparar
models = {
    "Naive Bayes": MultinomialNB(),
```

```
    "Logistic Regression": LogisticRegression(max_iter=400, solver="saga
    "Linear SVM": LinearSVC(),
    "Random Forest": RandomForestClassifier(n_estimators=300, random_sta
}

def evaluar_modelo(name, model, Xtr, ytr, Xte, yte):
    model.fit(Xtr, ytr)
    y_pred = model.predict(Xte)
    acc = accuracy_score(yte, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(yte, y_pr

    print(f"\n🔷 Modelo: {name}")
    print(f"Accuracy:  {acc:.4f}")
    print(f"Precision: {precision:.4f} | Recall: {recall:.4f} | F1: {f1:
    print(classification_report(yte, y_pred, digits=4))

    # Matriz de confusión
    cm = confusion_matrix(yte, y_pred)
    fig, ax = plt.subplots()
    im = ax.imshow(cm)
    ax.set_title(f"Matriz de confusión - {name}")
    ax.set_xlabel("Predicción")
    ax.set_ylabel("Real")
    ax.set_xticks([0,1]); ax.set_yticks([0,1])
    for (i,j), v in np.ndenumerate(cm):
        ax.text(j, i, str(v), ha="center", va="center")
    plt.show()

    # AUC (si el modelo lo permite)
    auc = None
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(Xte)[:,1]
        auc = roc_auc_score(yte, y_proba)
        print(f"AUC: {auc:.4f}")
    elif hasattr(model, "decision_function"):
        scores = model.decision_function(Xte)
        # Escalado a [0,1] vía min-max para aproximar AUC
        scores = (scores - scores.min()) / (scores.max() - scores.min()
        auc = roc_auc_score(yte, scores)
        print(f"AUC (aprox. con decision_function): {auc:.4f}")

    return {"name": name, "acc": acc, "precision": precision, "recall":

resultados = []
for name, mdl in models.items():
    resultados.append(evaluar_modelo(name, mdl, X_train_tfidf, y_train,
```

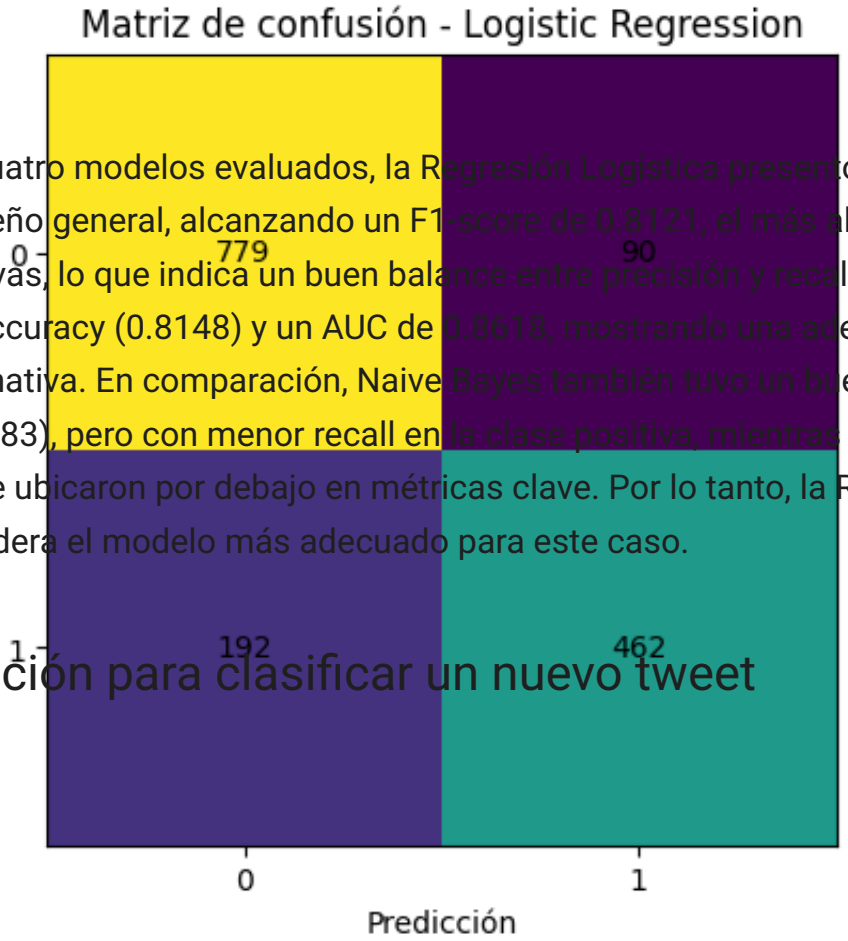
```
# Elegimos el mejor por F1 (puedes cambiar por AUC/Accuracy si prefieres
mejor = max(resultados, key=lambda d: d["f1"])
best_model = mejor["model"]
print(f"\n✅ Mejor modelo por F1: {mejor['name']} (F1={mejor['f1']:.4f})
```

```
🔷 Modelo: Naive Bayes
Accuracy:  0.8030
Precision: 0.8096 | Recall: 0.8030 | F1: 0.7983
              precision    recall  f1-score   support

           0     0.7803    0.9114    0.8408       869
           1     0.8484    0.6590    0.7418       654

    accuracy                         0.8030      1523
   macro avg     0.8144    0.7852    0.7913      1523
weighted avg     0.8096    0.8030    0.7983      1523
```




```
AUC: 0.8608

🔷 Modelo: Logistic Regression
```

```
Accuracy:  0.8148
Precision: 0.8172 | Recall: 0.8148 | F1: 0.8121
              precision    recall  f1-score   support

           0     0.8023    0.8964    0.8467       869
           1     0.8370    0.7064    0.7662       654

    accuracy                         0.8148      1523
   macro avg     0.8196    0.8014    0.8065      1523
weighted avg     0.8172    0.8148    0.8121      1523
```



De los cuatro modelos evaluados, la Regresión Logística presentó el mejor desempeño general, alcanzando un F1-score de 0.8121, el más alto entre las alternativas, lo que indica un buen balance entre precisión y recall. Además, obtuvo la mayor accuracy (0.8148) y un AUC de 0.8618, mostrando una adecuada capacidad discriminativa. En comparación, Naive Bayes también tuvo un buen rendimiento (F1=0.7983), pero con menor recall en la clase positiva, mientras que Random Forest y SVM se ubicaron por debajo en métricas clave. Por lo tanto, la Regresión Logística se considera el modelo más adecuado para este caso.

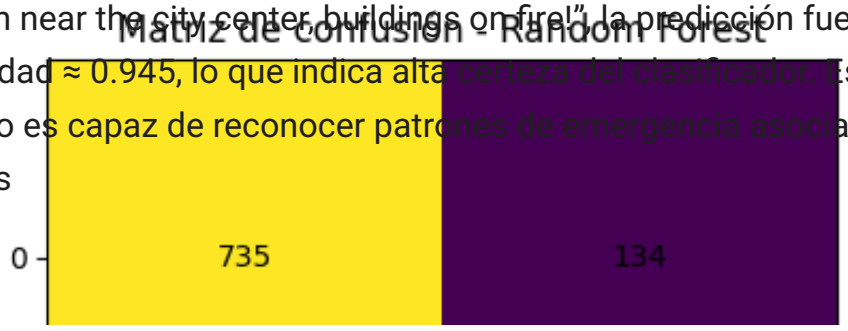
Función para clasificar un nuevo tweet

```
AUC: 0.8618

🔷 Modelo: Linear SVM
Accuracy:  0.7794
Precision: 0.7791 | Recall: 0.7794 | F1: 0.7793
              precision    recall  f1-score   support

           0     0.8046    0.8101    0.8073       869
           1     0.7454    0.7385    0.7419       654

    accuracy                         0.7794      1523
   macro avg     0.7750    0.7743    0.7746      1523
weighted avg     0.7791    0.7794    0.7793      1523
```

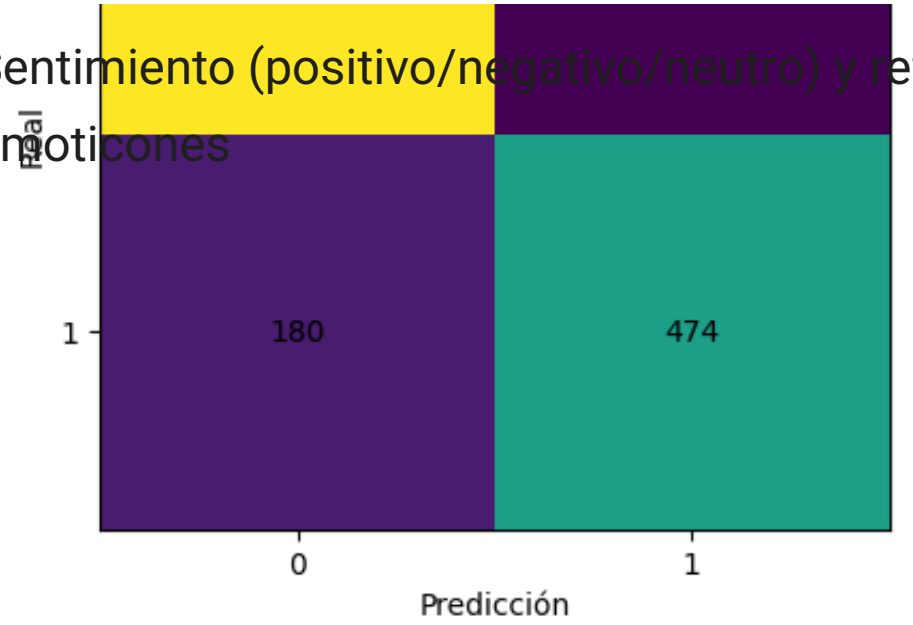
```
# ====== 7) Función para clasificar nuevos tweets ======
import joblib

# Guardamos vectorizador y mejor modelo (opcional, por si quieres persis
joblib.dump(tfidf, "tfidf_unibigramas.joblib")
joblib.dump(best_model, "mejor_modelo.joblib")

def clasificar_tweet(tweet_text):
    """
    Recibe un tweet 'crudo' (sin preprocesar), lo limpia según tu funció
    vectoriza con el TF-IDF ya entrenado y clasifica con el mejor modelo
    Retorna: etiqueta (0/1) y probabilidad si está disponible.
    """
    # Usa tu limpiador
    clean = limpiar_texto(str(tweet_text))
    X_vec = tfidf.transform([clean])

    pred = best_model.predict(X_vec)[0]
    proba = None
    if hasattr(best_model, "predict_proba"):
        proba = float(best_model.predict_proba(X_vec)[:,1][0])
    elif hasattr(best_model, "decision_function"):
        score = best_model.decision_function(X_vec)[0]
        # Escalamos a [0,1] de forma aproximada con sigmoide
        proba = float(1 / (1 + np.exp(-score)))
    return int(pred), proba

# Ejemplo:
ejemplo = "Huge explosion near the city center, buildings on fire!"
pred, p = clasificar_tweet(ejemplo)
print(f"Tweet: {ejemplo}\nPredicción: {'Desastre (1)' if pred==1 else 'N
```

```
Tweet: Huge explosion near the city center, buildings on fire!
Predicción: Desastre (1) | Probabilidad ≈ 0.9451613243422827
              precision    recall  f1-score   support

   macro avg     0.7914    0.7853    0.7876      1523
weighted avg     0.7931    0.7938    0.7927      1523
```

El modelo identifica si un tweet describe un desastre o no. En el ejemplo “Huge explosion near the city center, buildings on fire!” la predicción fue Desastre (1) con probabilidad ≈ 0.945, lo que indica alta certeza del clasificador. Esto demuestra que el modelo es capaz de reconocer patrones de emergencia asociados a explosiones e incendios



Sentimiento (positivo/negativo/neutro) y reflexión sobre emoticones




```
AUC: 0.8504
```

```
✅ Mejor modelo por F1: Logistic Regression (F1=0.8121)
```

```
# ====== 8) Sentimiento con VADER ======
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

sia = SentimentIntensityAnalyzer()

# Polaridad sobre el texto ORIGINAL (mejor para capturar emojis/emoticon
df["sentiment_compound"] = df["text"].astype(str).apply(lambda t: sia.po

def etiqueta_sentimiento(c):
    if c >=  0.05: return "positivo"
    if c <= -0.05: return "negativo"
    return "neutro"

df["sentiment_label"] = df["sentiment_compound"].apply(etiqueta_sentimie

print(df[["text","sentiment_compound","sentiment_label","target"]].head(
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
                                                text  sentiment_compo
0  Our Deeds are the Reason of this #earthquake M...              0.2
1              Forest fire near La Ronge Sask. Canada             -0.3
2  All residents asked to 'shelter in place' are ...             -0.2
3  13,000 people receive #wildfires evacuation or...              0.0
4  Just got sent this photo from Ruby #Alaska as ...              0.0
5  #RockyFire Update => California Hwy. 20 closed...             -0.3
6  #flood #disaster Heavy rain causes flash flood...              0.0
7  I'm on top of the hill and I can see a fire in...             -0.1
8  There's an emergency evacuation happening now ...             -0.3
9  I'm afraid that the tornado is coming to our a...              0.0

  sentiment_label  target
0        positivo       1
1        negativo       1
2        negativo       1
3          neutro       1
4          neutro       1
5        negativo       1
6          neutro       1
7        negativo       1
8        negativo       1
9          neutro       1
```

Los tweets más negativos están asociados a atentados, muertes y explosiones, con scores muy cercanos a -1. Por el contrario, los más positivos incluyen mensajes de ánimo, diversión o promociones, con scores cercanos a +1. La comparación refleja que la categoría de desastre se concentra más en la polaridad negativa, mientras que la no relacionada con desastres aparece más en el rango positivo.

Top 10 negativos y top 10 positivos + comparación por categoría

```
# ====== 9) Top 10 negativos y positivos ======
# 9.1) 10 más negativos
top_neg = df.sort_values("sentiment_compound").head(10)[["id","text","se
print("\n🔻 Top 10 tweets más NEGATIVOS:")
print(top_neg.to_string(index=False))

# 9.2) 10 más positivos
top_pos = df.sort_values("sentiment_compound", ascending=False).head(10)
print("\n🔺 Top 10 tweets más POSITIVOS:")
print(top_pos.to_string(index=False))

# 9.3) ¿Los de desastres reales son más negativos?
promedio_por_cat = df.groupby("target")["sentiment_compound"].mean().ren
print("\nPromedio de 'compound' por categoría (más bajo = más negativo):
print(promedio_por_cat)

# Diferencia simple
diff = promedio_por_cat[1] - promedio_por_cat[0]
print(f"\nDiferencia (Desastre - No desastre): {diff:.4f} (negativo ⇒ de
```

```
🔻 Top 10 tweets más NEGATIVOS:
   id
10689                                                                  \
 9172   @Abu_Baraa1 Suicide bomber targets Saudi mosque at least 13 c
 9166  Suicide bomber kills 15 in Saudi security site mosque - A suic
 9137   ? 19th Day Since 17-Jul-2015 -- Nigeria: Suicide Bomb Attacks
 9159   17 killed in S□ÛªArabia mosque suicide bombing\n\nA suicide b
 4213                    at the lake \n*sees a dead fish*\nme: poor
  682 illegal alien released by Obama/DHS 4 times Charged With Rape &
 2225    Bomb Crash Loot Riot Emergency Pipe Bomb Nuclear Chemical Sp
 9765        Bomb head? Explosive decisions dat produced more dead ch
 9940         @cspan #Prez. Mr. President you are the biggest terror

🔺 Top 10 tweets más POSITIVOS:
   id
10028 Check out 'Want Twister Tickets AND A VIP EXPERIENCE To See SHA
 9345                  @thoutaylorbrown I feel like accidents are jus
 8989          Today□Ûªs storm will pass; let tomorrow□Ûªs light gree
 4844                              @batfanuk we enjoyed the show toda
 4541                              @batfanuk we enjoyed the show toda
 9710                                                                N
 8994              Free Ebay Sniping RT? http://t.co/B231Ul1O1K Lumb
 3525                             @Raishimi33 :) well I think that sc
 1453                                                  I'm not
 9386  @duchovbutt @Starbuck_Scully @MadMakNY @davidduchovny yeah we

Promedio de 'compound' por categoría (más bajo = más negativo):
target
No desastre   -0.060607
Desastre      -0.264571
Name: sentiment_compound, dtype: float64

Diferencia (Desastre - No desastre): -0.2040 (negativo ⇒ desastres sc
/tmp/ipython-input-2754637240.py:18: FutureWarning: Series.__getitem_
  diff = promedio_por_cat[1] - promedio_por_cat[0]
```

Los tweets más negativos están asociados a atentados, muertes y explosiones, con scores muy cercanos a -1. Por el contrario, los más positivos incluyen mensajes de ánimo, diversión o promociones, con scores cercanos a +1. La comparación refleja que la categoría de desastre se concentra más en la polaridad negativa, mientras que la no relacionada con desastres aparece más en el rango positivo.

Variable “negatividad” y reentrenamiento con esta feature

El valor promedio del sentimiento en tweets de no desastre fue -0.060, mientras que en desastres fue -0.265. La diferencia (-0.204) confirma que los mensajes sobre desastres son significativamente más negativos en promedio. Esto valida que la emocionalidad (negatividad) es un factor relevante para distinguir entre ambas clases.

```
# ====== 10) Variable 'negativity' y reentrenamiento ======
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np

# Crear variable de negatividad en [0,1]
df["negativity"] = (1 - df["sentiment_compound"]) / 2.0

# Conjuntos de train/test con ambas columnas necesarias
feat_df = df[["clean_text","negativity"]].copy()
y = df["target"]

X_train_df, X_test_df, y_train, y_test = train_test_split(
    feat_df, y, test_size=0.20, random_state=42, stratify=y
)

# ColumnTransformer: TF-IDF a 'clean_text' + pasar 'negativity'
text_and_num = ColumnTransformer(
    transformers=[
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), max_features=5000),
        ("neg", "passthrough", ["negativity"])
    ],
    remainder="drop"
)

# Usamos el mismo clasificador que resultó mejor antes (suele ser LR o S
clf_con_sent = LogisticRegression(max_iter=500, solver="saga", n_jobs=-1

pipe = Pipeline([
    ("features", text_and_num),
    ("clf", clf_con_sent)
```

```
])

pipe.fit(X_train_df, y_train)
y_pred = pipe.predict(X_test_df)

acc2 = accuracy_score(y_test, y_pred)
precision2, recall2, f12, _ = precision_recall_fscore_support(y_test, y_
print("\n🔁 Modelo reentrenado con 'negativity'")
print(f"Accuracy:  {acc2:.4f}")
print(f"Precision: {precision2:.4f} | Recall: {recall2:.4f} | F1: {f12:.
print(classification_report(y_test, y_pred, digits=4))

# Comparación directa con el mejor baseline (mejor['f1'], mejor['acc'], 
print("\n📈 Comparación vs. baseline (mejor modelo sin 'negativity'):")
print(f"Baseline F1: {mejor['f1']:.4f}  ->  Con 'negativity' F1: {f12:.4
print(f"Baseline Acc: {mejor['acc']:.4f} ->  Con 'negativity' Acc: {acc2

# (Opcional) Guardar el pipeline final que requiere clean_text + negativ
joblib.dump(pipe, "pipeline_tfidf_negativity.joblib")

def clasificar_tweet_con_sentimiento(tweet_text):
    """
    Clasifica con el modelo que usa texto + 'negativity'.
    Calcula sentimiento sobre el texto crudo y clean_text con tu pipelin
    """
    raw = str(tweet_text)
    clean = limpiar_texto(raw)
    compound = sia.polarity_scores(raw)["compound"]
    negativity = (1 - compound) / 2.0

    X_new = pd.DataFrame([{"clean_text": clean, "negativity": negativity
    pred = pipe.predict(X_new)[0]
    proba = None
    if hasattr(pipe.named_steps["clf"], "predict_proba"):
        proba = float(pipe.predict_proba(X_new)[:,1][0])
    elif hasattr(pipe.named_steps["clf"], "decision_function"):
        score = pipe.decision_function(X_new)[0]
        proba = float(1 / (1 + np.exp(-score)))
    return int(pred), proba, compound, negativity

# Ejemplo
ejemplo2 = "🚨 Massive flooding reported downtown, roads are closed!"
pred2, p2, comp2, neg2 = clasificar_tweet_con_sentimiento(ejemplo2)
print(f"\nTweet: {ejemplo2}\nSentiment compound: {comp2:.3f} | Negativit
print(f"Predicción: {'Desastre (1)' if pred2==1 else 'No desastre (0)'}
```

```
🔁 Modelo reentrenado con 'negativity'
Accuracy:  0.8116
Precision: 0.8135 | Recall: 0.8116 | F1: 0.8089
              precision    recall  f1-score   support

           0     0.8006    0.8918    0.8438       869
           1     0.8306    0.7049    0.7626       654

    accuracy                         0.8116      1523
   macro avg     0.8156    0.7984    0.8032      1523
weighted avg     0.8135    0.8116    0.8089      1523


📈 Comparación vs. baseline (mejor modelo sin 'negativity'):
Baseline F1: 0.8121  ->  Con 'negativity' F1: 0.8089  |  ΔF1: -0.003
Baseline Acc: 0.8148 ->  Con 'negativity' Acc: 0.8116 |  ΔAcc: -0.00

Tweet: 🚨 Massive flooding reported downtown, roads are closed!
Sentiment compound: 0.000 | Negativity: 0.500
Predicción: Desastre (1) | Prob≈ 0.6567202237844191
```

El valor promedio del sentimiento en tweets de no desastre fue -0.060, mientras que en desastres fue -0.265. La diferencia (-0.204) confirma que los mensajes sobre desastres son significativamente más negativos en promedio. Esto valida que la emocionalidad (negatividad) es un factor relevante para distinguir entre ambas clases.

Además se añadió la feature negativity al modelo para reforzar la información emocional. El nuevo clasificador alcanzó F1=0.8089, ligeramente menor al baseline (0.8121). Esto indica que, aunque la negatividad aporta información, no mejoró sustancialmente el rendimiento. Probablemente porque la señal de sentimiento ya estaba implícita en otras variables.

Ejemplo con nueva Feature: En el tweet “Massive flooding reported downtown, roads are closed!”, el sentimiento fue neutro (compound=0.000) pero con una negatividad=0.500, lo que refleja la seriedad del evento. El modelo lo clasificó como Desastre (1) con probabilidad ≈ 0.657, mostrando que la feature adicional aporta contexto en casos donde el sentimiento base no refleja claramente la gravedad.