

Proyecto PLN

Grupo: 5BM1

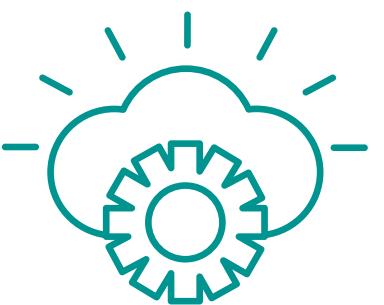
Integrantes :

- Góngora Hernández Jorge Alan
- Miranda Chavez Victor Ulises
- Pérez Sánchez Ives Lancelote



Tabla de contenido

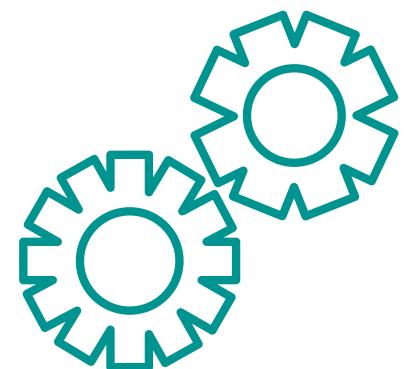
01 Introducción



02 Enfoque basado en diccionarios de términos afectivos



03 Enfoque basado en aprendizaje automático



04 Pruebas y Resultados



INTRODUCCIÓN

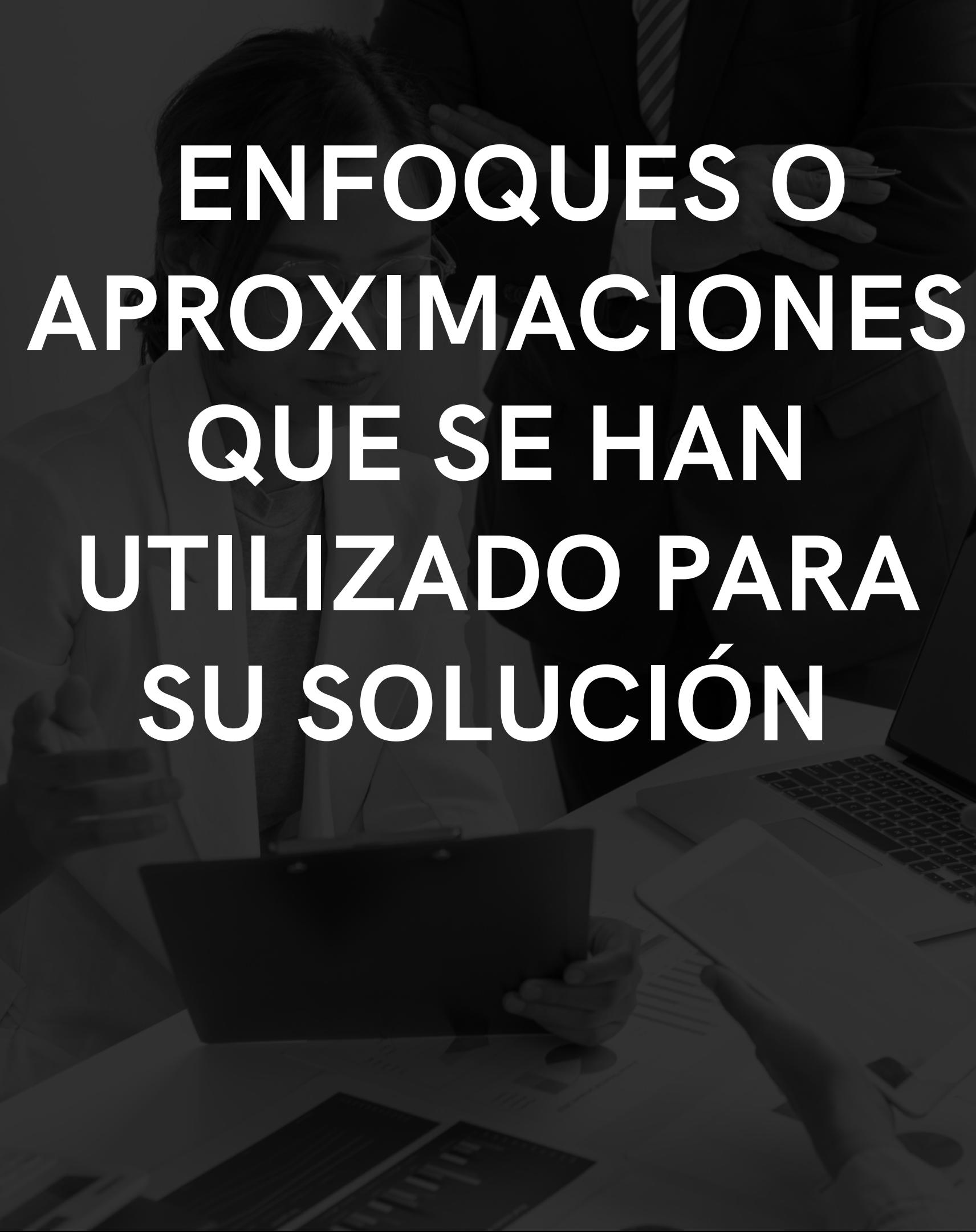
DESCRIPCIÓN

La tarea de polaridad de opinión en Aprendizaje Automático se refiere a la clasificación de texto en las categorías que se requieran: Esto se utiliza a menudo para analizar opiniones o reseñas de productos, servicios o eventos, con el objetivo de determinar el sentimiento general de los clientes o usuarios. Los modelos de Aprendizaje Automático se entran con un conjunto de datos etiquetado, donde cada ejemplo de texto está asociado con una etiqueta. Luego, los modelos se utilizan para clasificar nuevos ejemplos de texto no etiquetado.

MOTIVACIÓN

La motivación para llevar a cabo tareas de polaridad de opinión en Aprendizaje Automático proviene de la necesidad de analizar y comprender cómo las personas sienten y opinan sobre un tema específico. Esto es especialmente importante en campos como el marketing, la investigación de mercado y la atención al cliente, ya que permite a las empresas y organizaciones conocer las opiniones de sus clientes y adaptar sus estrategias en consecuencia.

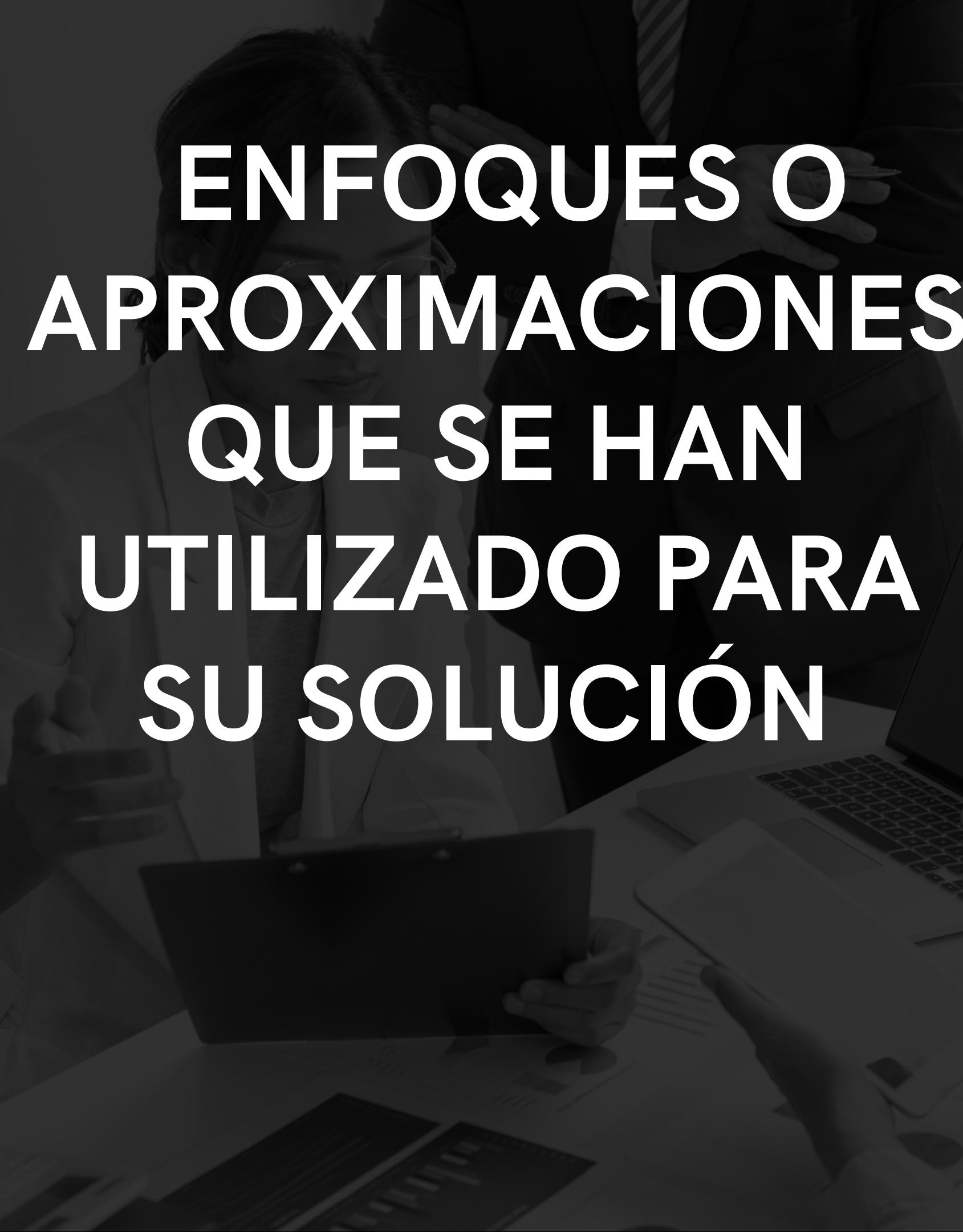
Además, el análisis de sentimientos también es útil en áreas como las redes sociales, la política y la noticias, donde se utiliza para entender cómo la gente reacciona a un evento o tema en particular.



ENFOQUES O APROXIMACIONES QUE SE HAN UTILIZADO PARA SU SOLUCIÓN

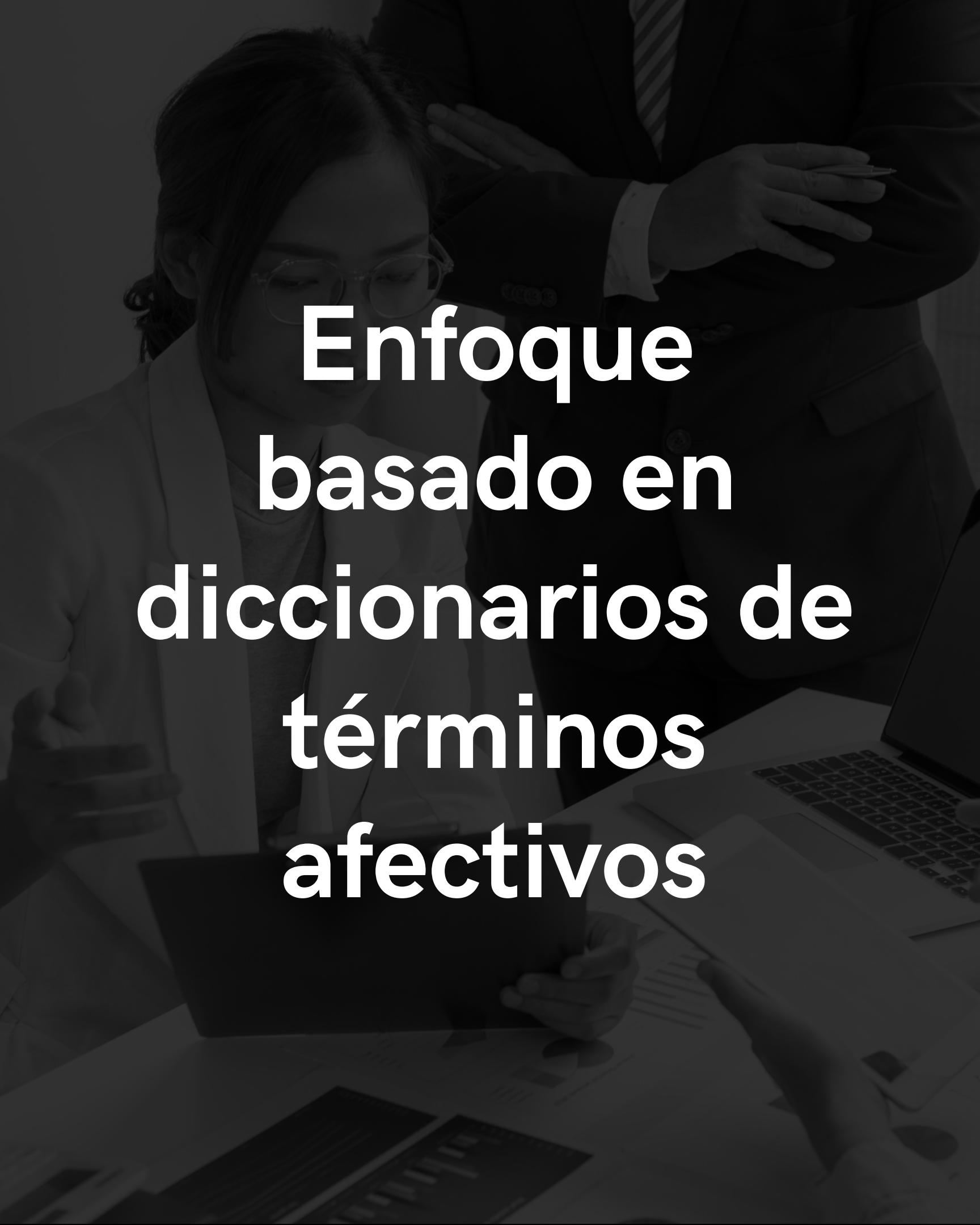
Hay varios enfoques o aproximaciones que se han utilizado para resolver la tarea de polaridad de opinión en Aprendizaje Automático para nuestra solución utilizamos los siguientes:

Análisis de sentimientos basados en lexicones: Este enfoque utiliza una lista previamente elaborada de palabras conocidas como "lexicones", que se asocian con una polaridad positiva o negativa. Los textos son analizados para contar las palabras positivas y negativas presentes, y la polaridad se determina en función de la cantidad de palabras de cada tipo.



ENFOQUES O APROXIMACIONES QUE SE HAN UTILIZADO PARA SU SOLUCIÓN

- Aprendizaje automático supervisado: Este enfoque utiliza algoritmos de aprendizaje automático supervisado, como regresión logística, redes neuronales y algoritmos de árboles de decisión, para aprender las características que indican polaridad positiva o negativa a partir de un conjunto de datos etiquetado previamente.

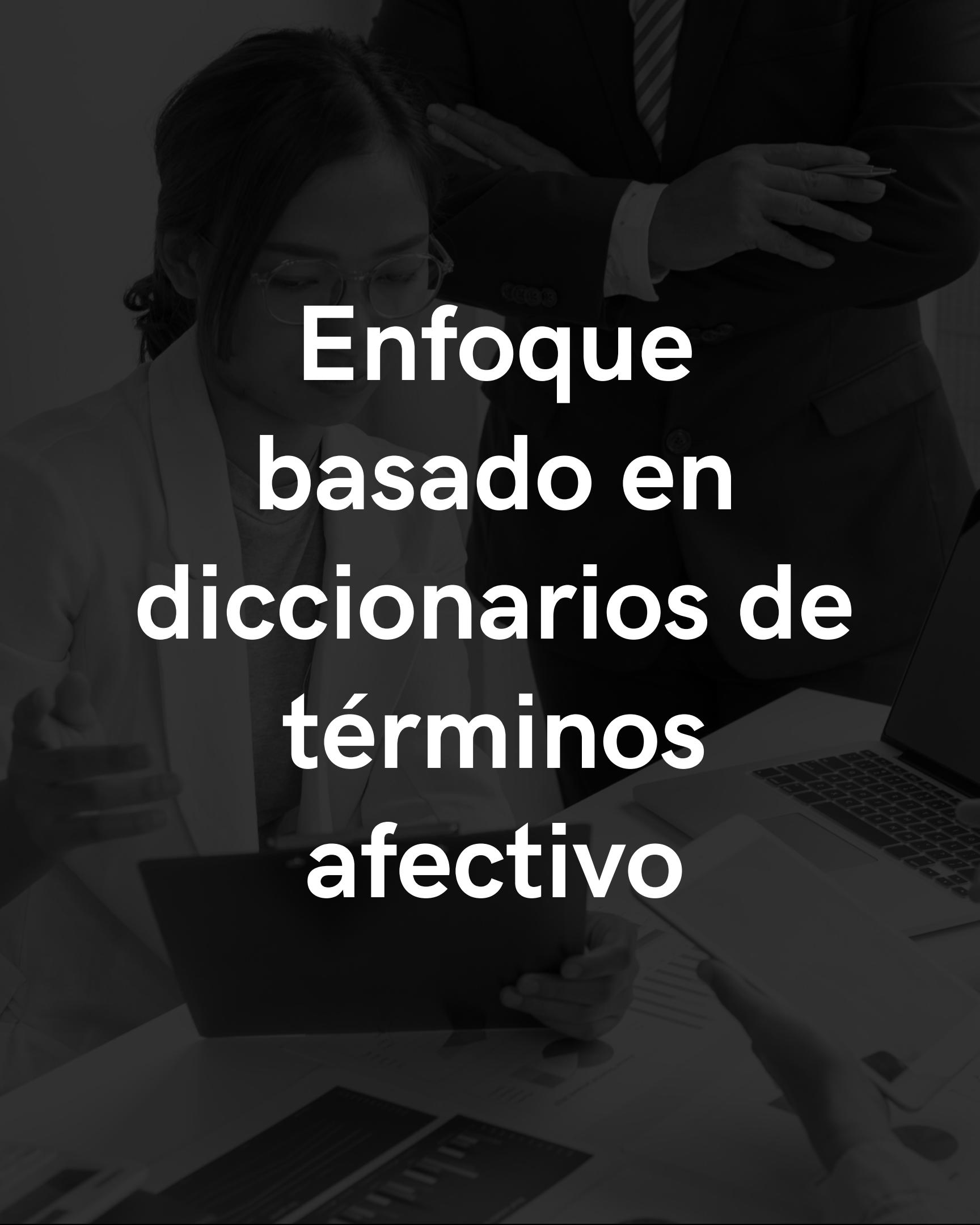


Enfoque basado en diccionarios de términos afectivos

Descripción del enfoque

La idea es crear un diccionario de términos previamente seleccionados que se consideran como afectivamente cargados, ya sea positivamente o negativamente. Luego, se utiliza este diccionario para analizar el texto y asignar una polaridad, es decir, si el texto tiene un tono positivo, negativo o neutral.

Una vez construido el diccionario, utilizar un *sistema de ponderación*, donde cada palabra tiene un peso específico asignado de acuerdo a su carga emocional y se suman los pesos para determinar la polaridad general.



Enfoque basado en diccionarios de términos afectivo

Recursos utilizados

- Gambino O.& Calvo H. (2016). A Comparison Between Two Spanish SentimentLexicons in the Twitter Sentiment Analysis Task. Instituto Politécnico Nacional, ESCOM. Ciudad de México, México.
- Diccionario de polaridad SELL_full
- Data-set:
Rest_Mex_2022_Sentiment_Analysis_Track_Train
- Bibliotecas:
 - spacy
 - sklearn
 - matplotlib
 - pandas
 - numpy
 - re
 - os

PREPROCESAMIENTO

1. Cargamos el dataset

Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx

2. Obtenemos los valores de las columnas Tittle, Opinion

3. Obtenemos los valores de las columnas Polarity y Attraction

4. Concatenamos las columnas de Tittle y Opinion

5. Lematizamos, tokenizamos y removemos stop-words del resultado de la concatenación a través de SPACY

6. Construimos un nuevo dataset

```
def lematizacion(texto):
    doc = nlp(texto)
    palabras = ""

    for token in doc:
        if not (token.is_punct | token.is_stop) and token.orth_.isalpha():
            lemma = token.lemma_.lower()
            palabras += lemma + " "

    return palabras

df = pd.read_excel("Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx")

# Obtenemos los valores columnas de tittle y opinion
X = df.iloc[:, :2].values
# Obtenemos los valores columnas de Polarity y attraction
Y = df.iloc[:, 2:3].values

#Categorizamos la columna atraccion
# LaEncoder = preprocessing.LabelEncoder()
# Y[:, -1] = LaEncoder.fit_transform(Y[:, -1])

# Lematizamos las columnas de title y opinion
titulosOpinionesLematizados = []
for i in range(len(X)):
    tituloOpinion_Unida = str(X[i][0]) + "." + str(X[i][1])
    textoLematizado = lematizacion(tituloOpinion_Unida)
    titulosOpinionesLematizados.append(textoLematizado)

#Guarda el dataset en csv
data = {
    "Titulo y opinion": titulosOpinionesLematizados,
    "Polaridad": Y[:, -2],
    "Atraccion": Y[:, -1],
}
dfLemTitulosOpiniones = pd.DataFrame(data)
dfLemTitulosOpiniones.to_csv("restMexLematizado-parte2.csv")
```

REPRESENTACIÓN DEL TEXTO

Hablamos de un texto plano

ALGORITMO PARA DETERMINAR LA POLARIDAD

1. Obtenemos el diccionario de términos afectivos de SEL_full
2. Evaluamos cada opinión
3. Evaluamos cada palabra en la opinión
4. Buscamos esa palabra en el diccionario, obtenemos su puntuación y sumamos su categoría
5. Realizamos una sumatoria completa de emociones positivas y negativas
6. Calculamos la diferencia
7. Clasificamos la opinión por el nivel obtenido de acuerdo con los umbrales que definimos

```
obtenerPolaridades(x_train, y_train, lexiconDic, boundStart, salto):  
  
    features = []  
    for opinion in x_train:  
  
        valor_alegria = 0.0000  
        valor_enojo = 0.0000  
        valor_miedo = 0.0000  
        valor_repulsion = 0.0  
        valor_sorpresa = 0.0  
        valor_tristeza = 0.0  
  
        op_separada = re.split('\s+', opinion)  
  
        dicEmociones = {}  
        for palabra in op_separada:  
  
            if palabra in lexiconDic:  
                puntuaciones = lexiconDic[palabra]  
  
                for emocion, valor in puntuaciones:  
                    if emocion == 'Alegría':  
                        valor_alegria += round(float(valor),4)  
                    elif emocion == 'Tristeza':  
                        valor_tristeza += round(float(valor),4)  
                    elif emocion == 'Enojo':  
                        valor_enojo += round(float(valor),4)  
                    elif emocion == 'Repulsión':  
                        valor_repulsion += round(float(valor),4)  
                    elif emocion == 'Miedo':  
                        valor_miedo += round(float(valor),4)  
                    elif emocion == 'Sorpresa':  
                        valor_sorpresa += round(float(valor),4)  
  
            dicEmociones['Alegria'] = valor_alegria  
            dicEmociones['Tristeza'] = valor_tristeza  
            dicEmociones['Enojo'] = valor_enojo  
            dicEmociones['Repulsion'] = valor_repulsion  
            dicEmociones['Miedo'] = valor_miedo  
            dicEmociones['Sorpresa'] = valor_sorpresa  
  
            emocionPositivaAcum = dicEmociones['Alegria'] + dicEmociones['Sorpresa']  
            emocionNegativaAcum = dicEmociones['Enojo'] + dicEmociones['Miedo'] + dicEmociones['Repulsion'] + dicEmociones['Tristeza']  
  
            difPosNeg = emocionPositivaAcum - emocionNegativaAcum  
  
            if difPosNeg > boundStart + (salto*3):  
                features.append(5)  
            elif boundStart + (salto*2) <= difPosNeg and difPosNeg <= boundStart + (salto*3):  
                features.append(4)  
            elif boundStart + salto <= difPosNeg and difPosNeg < boundStart + (salto*2):  
                features.append(3)  
            elif boundStart <= difPosNeg and difPosNeg < boundStart + salto:  
                features.append(2)  
            elif difPosNeg < boundstart:  
                features.append(1)
```

REGLAS O HEURÍSTICAS UTILIZADAS

Para poder clasificar la opinión de acuerdo con los umbrales que definimos, tenemos la opción de que a partir de un umbral de inicio, este realizará saltos automáticos de un tamaño que nosotros proveemos.

La otra opción es definirlos manualmente.

MÉTRICAS UTILIZADAS

Para evaluar el rendimiento de nuestro modelo, nos concentraremos en la métrica de "accuracy"

```
if difPosNeg > boundStart + (salto*3):
    features.append(5)
elif boundStart + (salto*2) <= difPosNeg and difPosNeg <= boundStart + (salto*3):
    features.append(4)
elif boundStart + salto <= difPosNeg and difPosNeg < boundStart + (salto*2):
    features.append(3)
elif boundStart <= difPosNeg and difPosNeg < boundStart + salto:
    features.append(2)
elif difPosNeg < boundStart:
    features.append(1)
```

```
# if difPosNeg > -0.0003:
#     features.append(5)
# elif -0.3 <= difPosNeg and difPosNeg <= -0.0003:
#     features.append(4)
# elif -0.6 <= difPosNeg and difPosNeg < -0.3:
#     features.append(3)
# elif -1 <= difPosNeg and difPosNeg < -0.6:
#     features.append(2)
# elif difPosNeg <-1:
#     features.append(1)
```

```
accuracy = accuracy_score(y_true, y_pred)

target_names = ['Muy Negativo', 'Negativo', 'Neutro', 'Positivo', 'MuyPositivo']
reporte = classification_report(y_true, y_pred, target_names=target_names, zero_division=0)

file.write(f"-----UMBRAL: {boundStart} hasta {boundStart + salto*3} con saltos de {salto}\n")
file.write("Reporte de clasificación:\n")
file.write(reporte)
file.write("\n")
```

ETAPA DE ENTRENAMIENTO

```
num_folders = 5
validation_set = cross_validation([X_train, y_train], num_folders)

accuracyList_1 = []
accuracyList_2 = []
accuracyList_3 = []
accuracyList_4 = []

umbralesInicio = [-4, -3, -3.5, -2.5]
salto = 0.35
# umbralesInicio = [-1, -1, -1, -1]
# salto = 0.5
# umbralesInicio = [-2.5, -2, -1.5, -1]
# salto = 0.6
|
print("salto", salto)

file = open("ReportesClasificacion.txt", 'a')

for i in range(num_folders):
    file.write(f"-----Folder {i}-----\n")
    print("Folder ", i)
    accuracyList_1.append(obtenerPolaridades(validation_set.val_set[i].x_train, validation_set.val_set[i].y_train, lexiconDic, umbralesInicio[0], salto))
    accuracyList_2.append(obtenerPolaridades(validation_set.val_set[i].x_train, validation_set.val_set[i].y_train, lexiconDic, umbralesInicio[1], salto))
    accuracyList_3.append(obtenerPolaridades(validation_set.val_set[i].x_train, validation_set.val_set[i].y_train, lexiconDic, umbralesInicio[2], salto))
    accuracyList_4.append(obtenerPolaridades(validation_set.val_set[i].x_train, validation_set.val_set[i].y_train, lexiconDic, umbralesInicio[3], salto))

accuracyList_1 = np.array(accuracyList_1, dtype=object)
accuracyList_2 = np.array(accuracyList_2, dtype=object)
accuracyList_3 = np.array(accuracyList_3, dtype=object)
accuracyList_4 = np.array(accuracyList_4, dtype=object)

accuracy_1_promedio = statistics.mean(accuracyList_1[:, 0])
accuracy_2_promedio = statistics.mean(accuracyList_2[:, 0])
accuracy_3_promedio = statistics.mean(accuracyList_3[:, 0])
accuracy_4_promedio = statistics.mean(accuracyList_4[:, 0])

accuracyPromedios = {
    umbralesInicio[0]: accuracy_1_promedio,
    umbralesInicio[1]: accuracy_2_promedio,
    umbralesInicio[2]: accuracy_3_promedio,
    umbralesInicio[3]: accuracy_4_promedio
}

print("Promedios accuracy: ", accuracyPromedios)

mejorPromedio = max(accuracyPromedios.items(), key=lambda x: x[1])
print("Mejor promedio: ", mejorPromedio)
```

ETAPA DE PRUEBA

```
mejorPromedio = max(accuracyPromedios.items(), key=lambda x: x[1])
print("Mejor promedio: ", mejorPromedio)

accuracyFinal = 0

if(mejorPromedio[0] == umbralesInicio[0]):
    accuracyFinal = obtenerPolaridades(X_test, y_test, lexiconDic, umbralesInicio[0], salto)
elif(mejorPromedio[0] == umbralesInicio[1]):
    accuracyFinal = obtenerPolaridades(X_test, y_test, lexiconDic, umbralesInicio[1], salto)
elif(mejorPromedio[0] == umbralesInicio[2]):
    accuracyFinal = obtenerPolaridades(X_test, y_test, lexiconDic, umbralesInicio[2], salto)
elif(mejorPromedio[0] == umbralesInicio[3]):
    accuracyFinal = obtenerPolaridades(X_test, y_test, lexiconDic, umbralesInicio[3], salto)
file.close()

print("Accuracy final: ", accuracyFinal[0])
y_true = accuracyFinal[2]
y_pred = accuracyFinal[3]

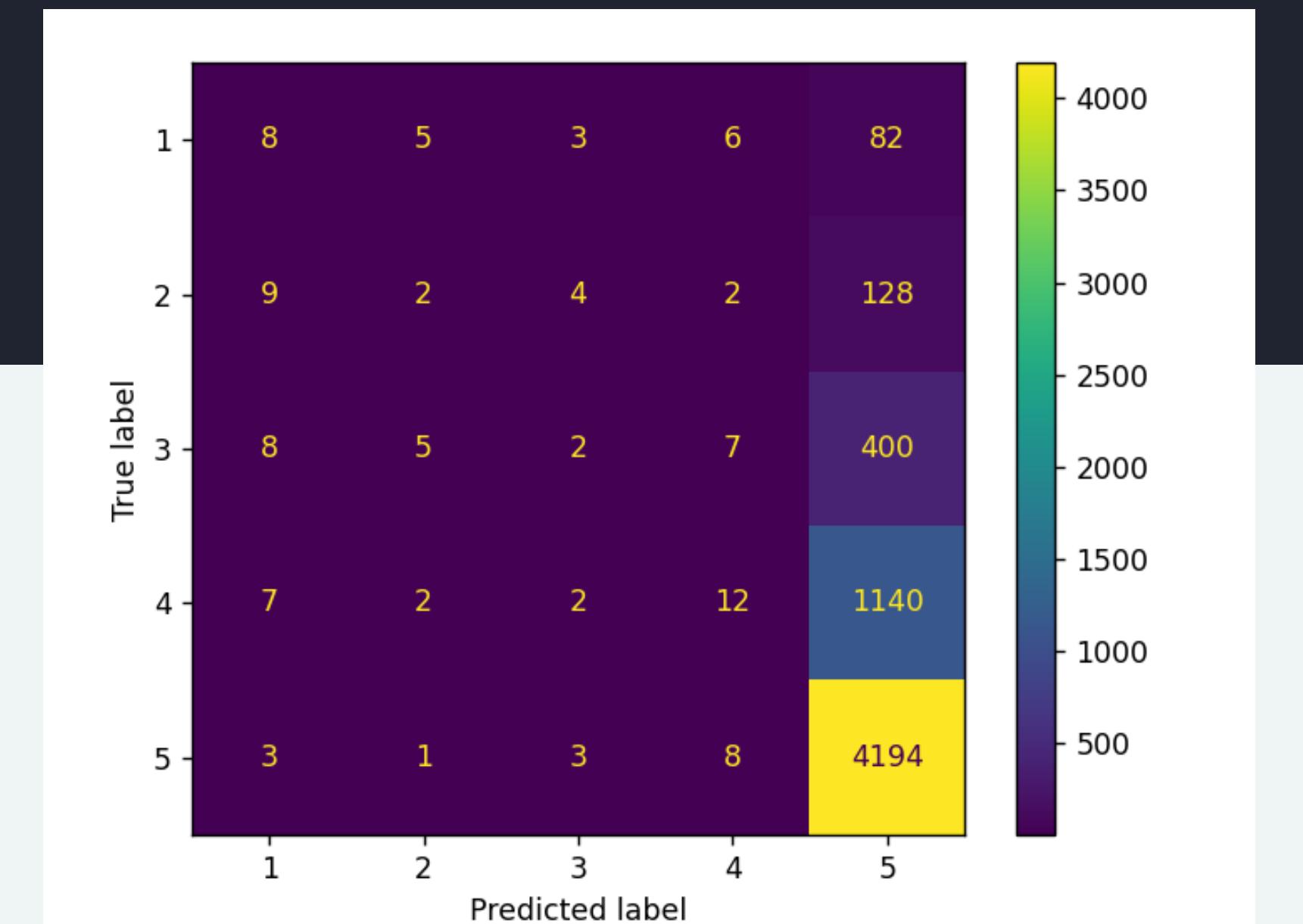
print (confusion_matrix(y_true, y_pred))
ConfusionMatrixDisplay.from_predictions(y_true, y_pred)

plt.show()
```

RESULTADOS

```
salto 0.35
Folder 0
Folder 1
Folder 2
Folder 3
Folder 4
Promedios accuracy: {-4: 0.6929537767937415, -3: 0.6941950357239187, -3.5: 0.6932847806016751, -2.5: 0.6945674152753228}
Mejor promedio: (-2.5, 0.6945674152753228)
Accuracy final: 0.6979976832698991
[[ 8   5   3   6   82]
 [ 9   2   4   2  128]
 [ 8   5   2   7  400]
 [ 7   2   2  12 1140]
 [ 3   1   3   8 4194]]
```

Caso: SALTOS
AUTOMÁTICOS



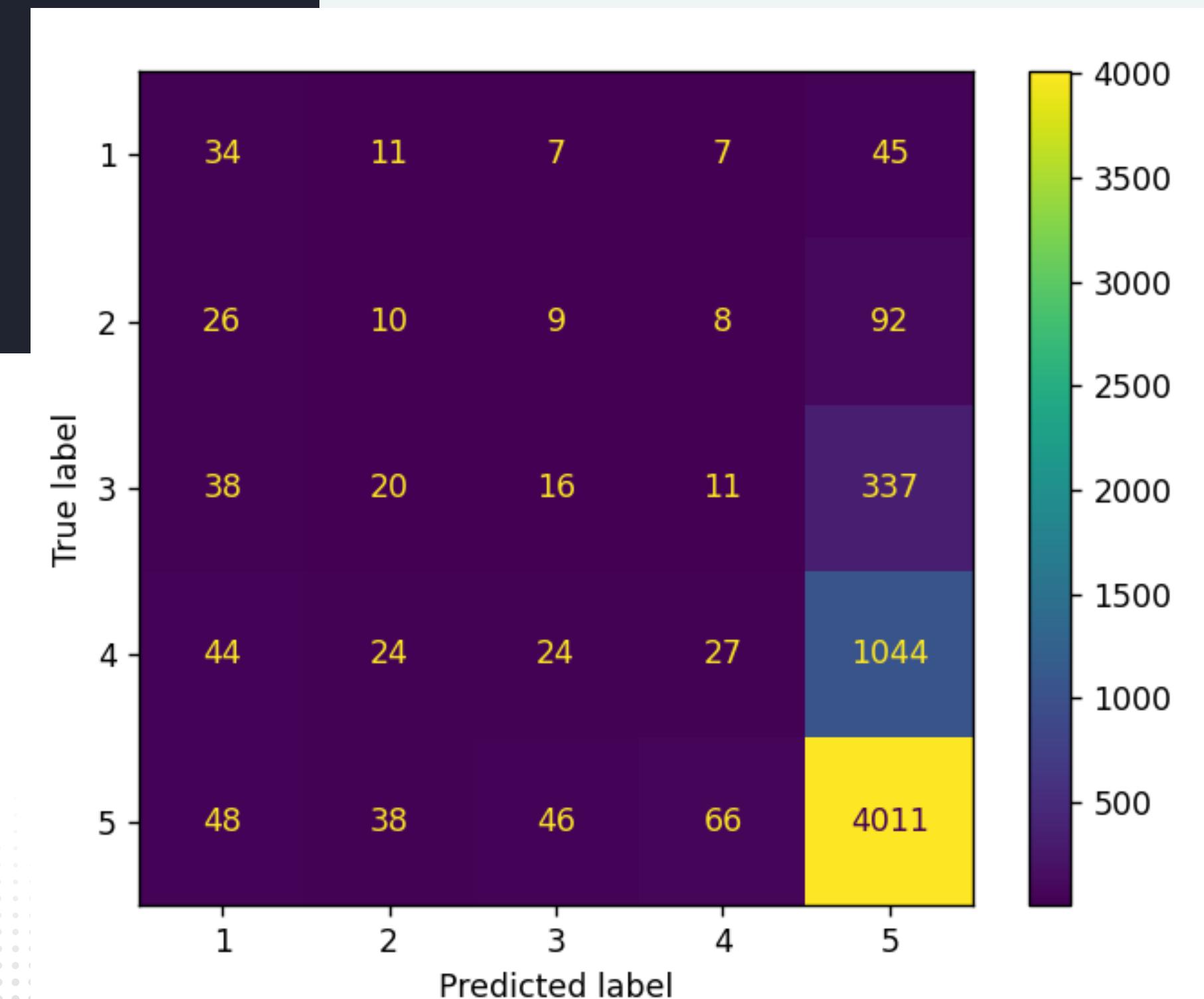
RESULTADOS

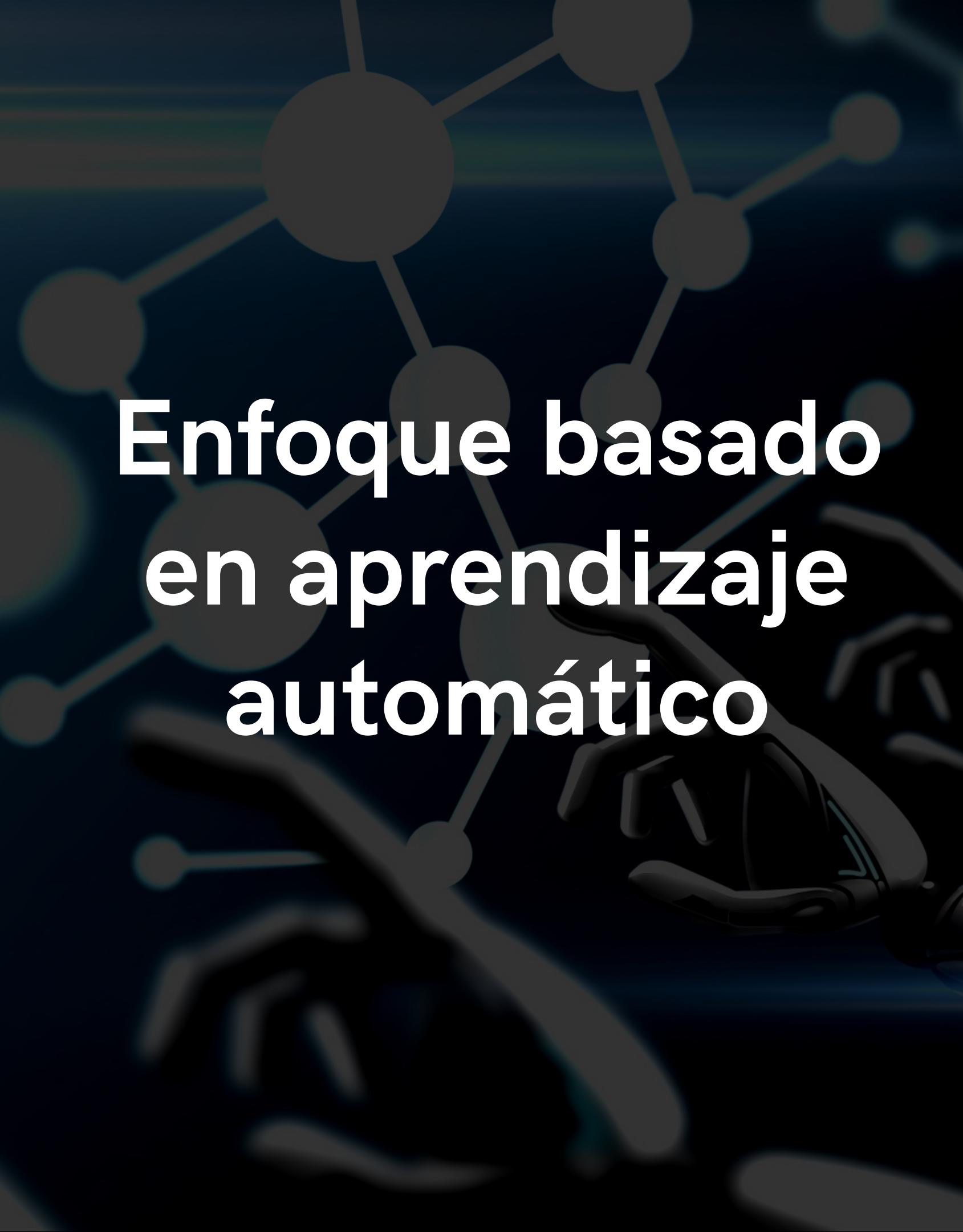
Mejor promedio: (-4, 0.672845370890765)

Accuracy final: 0.6781399966903856

```
[[ 34  11   7   7  45]
 [ 26  10   9   8  92]
 [ 38  20  16  11 337]
 [ 44  24  24  27 1044]
 [ 48  38  46  66 4011]]
```

Caso: MANUALMENTE

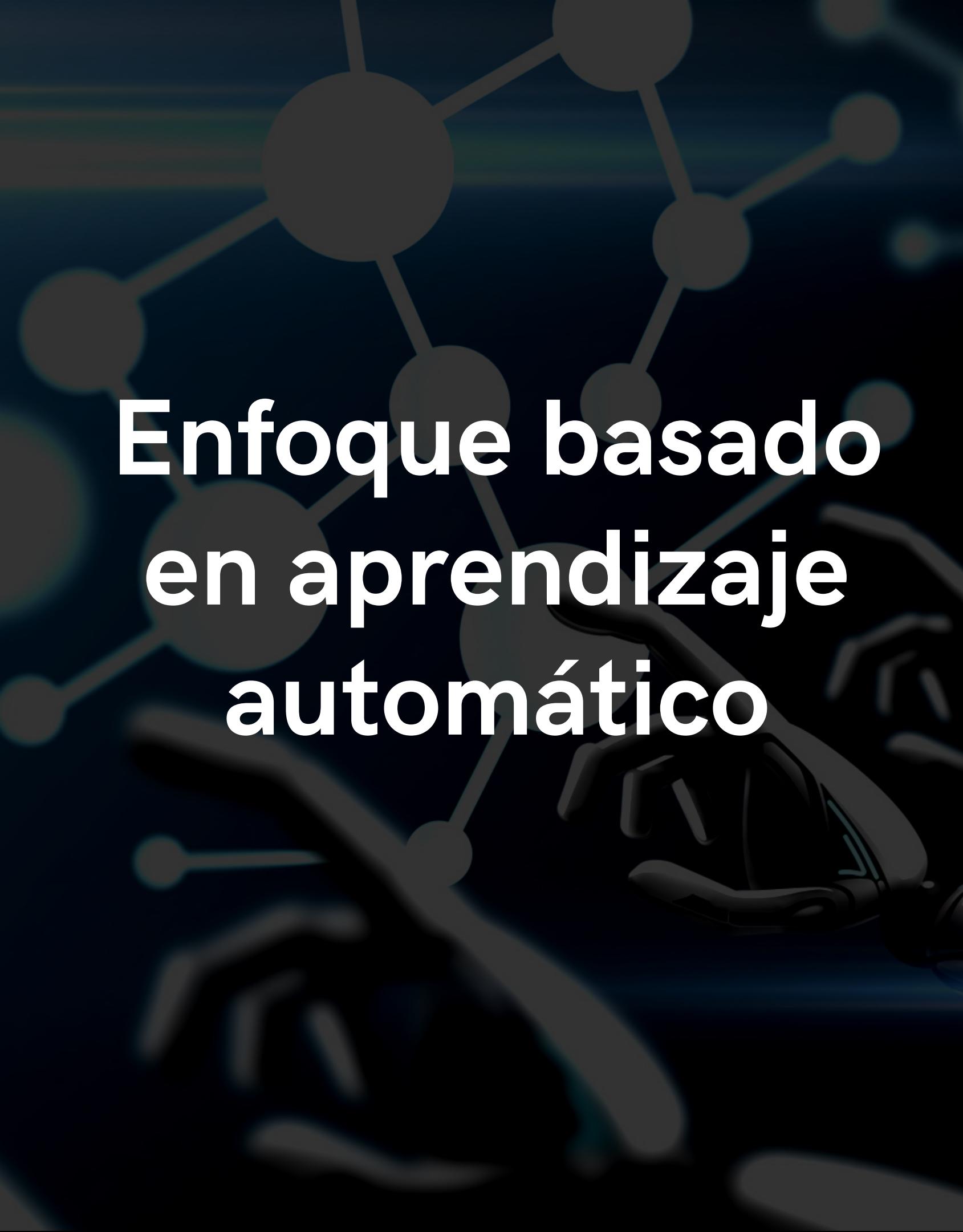




Enfoque basado en aprendizaje automático

Descripción del enfoque

- Uso de algoritmos de Aprendizaje de Maquina
- Modelo supervisado
- Etapas de entrenamiento y prueba
- Clasificar nuevos conjuntos de datos como positivo, negativo o neutral
- 5 categorias de polaridad



Enfoque basado en aprendizaje automático

Recursos utilizados

- Gambino O.& Calvo H. (2016). A Comparison Between Two Spanish SentimentLexicons in the Twitter Sentiment Analysis Task. Instituto Politécnico Nacional, ESCOM.
- Dataset:
Rest_Mex_2022_Sentiment_Analysis_Track_Train
- Librerias:
NLTK
Pandas
SkLearn

PREPROCESAMIENTO

1. Cargamos el dataset

Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx

2. Se carga el archivo xlsx en un data set con ayuda de pandas

3. Se aplica el proceso de tokenización y lematización para las columnas de "Tittle" y "Opinion" con NLTK

4. Se realiza una copia de las columnas de
'Titulo_lemmatized', 'Opinion_lemmatized',
'Polarity' y 'Attraction'

5. Se exporta el dataset como
"RestMex_proyectoParte2.csv"

```
1 import pandas as pd
2 import nltk
3 from nltk.tokenize import word_tokenize
4 from nltk.stem import WordNetLemmatizer
5
6 # Inicializa el lematizador
7 lemmatizer = WordNetLemmatizer()
8
9 # Define una función para aplicar el proceso de tokenización y lematización a una cadena
10 def tokenize_lemmatize(text):
11     # Tokeniza el texto
12     tokens = word_tokenize(str(text))
13
14     # Lematiza cada token y devuelve la lista de lemas
15     lemmas = [lemmatizer.lemmatize(token) for token in tokens]
16     return lemmas
17
18 # Carga el archivo xlsx en un DataFrame
19 df = pd.read_excel('Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx')
20
21 # Aplica el proceso de tokenización y lematización a las columnas 'Titulo' y 'Opinion'
22 df['Titulo_lemmatized'] = df['Title'].apply(tokenize_lemmatize)
23 df['Opinion_lemmatized'] = df['Opinion'].apply(tokenize_lemmatize)
24
25 # Crea una copia del DataFrame original con las columnas 'Titulo_lemmatized', 'Opinion_lemmatized', 'Polarity', 'Attraction'
26 df_processed = df[['Titulo_lemmatized', 'Opinion_lemmatized', 'Polarity', 'Attraction']]
27
28 df_processed.to_csv("RestMex_proyectoParte2.csv")
```

REPRESENTACIÓN DEL TEXTO

Utilizamos una representación de texto en espacio vectorial, la cual nos permite asociar a cada característica un valor

1-Representación Binaria: Asigna un valor binario a cada palabra del documento para indicar si existe o no.

2- Representacion Frecuencial: Indica la cantidad de aparición de una palabra, asigna un valor numérico a cada palabra en un documento

FEATURES

En el caso de un vectorizador binario, cada feature es un término (palabra) del vocabulario y su valor es 1 si el término aparece en el documento y 0 si no aparece.

En el caso de un vectorizador frecuencial, cada feature es un término (palabra) del vocabulario y su valor es el número de veces que el término aparece en el documento.

```
# Inicializa el vectorizador DE BINARIO
vectorizer = CountVectorizer(binary=True)

# Vectorizamos
X_train_val = vectorizer.fit_transform(X_train_val)
X_test_val = vectorizer.transform(X_test_val)
```

```
# Inicializa el vectorizador DE FRECUENCIAS
vectorizer = CountVectorizer(binary=False)

# Vectorizamos
X_train_val = vectorizer.fit_transform(X_train_val)
X_test_val = vectorizer.transform(X_test_val)
```

ALGORITMOS DE ML: REGRESION LOGISTICA

1. Cargamos el dataset en un DataFrame Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx
2. Asignamos X y Y, 'Titulo_lemmatized' y 'Opinion_lemmatized' como datos de entrada, y 'Polarity' y 'Attraction' como variables objetivo.
3. Se inicia el modelo regresión y el conjunto de validación
4. Separa el dataset en conjunto de entrenamiento y prueba para la columna de Polaridad y Attracction
5. Itera sobre los pliegues con el 80% de los datos donde obtiene los conjuntos de entrenamiento y validación
6. Se hace la representación vectorial binaria si es necesaria
7. Se entrena el modelo en entrenamiento y validación

```
# Carga el archivo xlsx en un DataFrame
df_processed = pd.read_csv('RestMex_proyectoParte2.csv')

X = df_processed[['Titulo_lemmatized', 'Opinion_lemmatized']] # Datos
X = X['Titulo_lemmatized'].values + X['Opinion_lemmatized'].values
y = df_processed[['Polarity', 'Attraction']] # Clases

# Inicializa el modelo
model = LogisticRegression(max_iter=5000, C=0.5)

# Inicializa el conjunto de validación
kf = KFold(n_splits=5, shuffle=True, random_state=0)

# -----PARA ATRACCION
# Separa el dataset en conjunto de entrenamiento y prueba, utilizando el 80% y el 20% de los datos
x_train, x_test, y_train, y_test = train_test_split(X, y['Attraction'].values, test_size=0.2, random_state=0)

# Inicializa las variables para almacenar las métricas
acc_list_Attraction = []
prec_list_Attraction = []
rec_list_Attraction = []
f1_list_Attraction = []

# Itera sobre los pliegues con el 80% de los datos
for train_index, val_index in kf.split(x_train):
    print("Pliegue de attraction")
    # Obtiene los conjuntos de entrenamiento y validación para el pliegue actual
    x_train_val = x_train[train_index]
    x_test_val = x_train[val_index]

    y_train_val = y_train[train_index]
    y_test_val = y_train[val_index]

    # Inicializa el vectorizador DE BINARIO
    vectorizer = CountVectorizer(binary=True)
```

ALGORITMOS DE ML: NAIVE BAYES

1. Cargamos el dataset en un DataFrame Rest_Mex_2022_Sentiment_Analysis_Track_Train.xlsx
2. Asignamos X y Y, 'Titulo_lemmatized' y 'Opinion_lemmatized' como datos de entrada, y 'Polarity' y 'Attraction' como variables objetivo.
3. Se inicia el modelo de Naive Bayes y el conjunto de validación
4. Separa el dataset en conjunto de entrenamiento y prueba para la columna de Polaridad y Attracction
5. Itera sobre los pliegues con el 80% de los datos donde obtiene los conjuntos de entrenamiento y validación
6. Se hace la representación vectorial binaria si es necesaria
7. Se entrena el modelo en entrenamiento y validación

```
# Carga el archivo xlsx en un DataFrame
df_processed = pd.read_csv('RestMex_proyectoParte2.csv')

X = df_processed[['Titulo_lemmatized', 'Opinion_lemmatized']] # Datos
X = X['Titulo_lemmatized'].values + X['Opinion_lemmatized'].values
y = df_processed[['Polarity', 'Attraction']] # Clases

# Inicializa el modelo
model = MultinomialNB()

# Inicializa el conjunto de validación
kf = KFold(n_splits=5, shuffle=True, random_state=0)

# -----PARA ATRACCION
# Separa el dataset en conjunto de entrenamiento y prueba, utilizando el 80% y el 20% de los
X_train, X_test, y_train, y_test = train_test_split(x, y['Attraction'].values, test_size=0.2)

# Inicializa las variables para almacenar las métricas
acc_list_Attraction = []
prec_list_Attraction = []
rec_list_Attraction = []
f1_list_Attraction = []

# Itera sobre los pliegues con el 80% de los datos
for train_index, val_index in kf.split(X_train):
    print("Pliegue de attraction")
    # Obtiene los conjuntos de entrenamiento y validación para el pliegue actual
    X_train_val = X_train[train_index]
    X_test_val = X_train[val_index]

    y_train_val = y_train[train_index]
    y_test_val = y_train[val_index]

    # Inicializa el vectorizador BINARIO
    vectorizer = CountVectorizer(binary=True)
```

MÉTRICAS UTILIZADAS

Son calculadas 4 métricas las cuales son el accuracy, precision, recall y la mas importante el measure(f1) la cual nos va a dar la efectividad del modelo, se calculan para cada una de las iteraciones pero con ayuda del promedio podemos obtener los resultados correspondientes.

```
# Calcula las métricas
acc = accuracy_score(y_test_val, y_pred)
prec = precision_score(y_test_val, y_pred, average='micro')
rec = recall_score(y_test_val, y_pred, average='micro')
f1 = f1_score(y_test_val, y_pred, average='micro')

# Agrega las métricas a las listas
acc_list_Attraction.append(acc)
prec_list_Attraction.append(prec)
rec_list_Attraction.append(rec)
f1_list_Attraction.append(f1)

# Calcula el promedio de las métricas
acc_mean_Attraction = sum(acc_list_Attraction) / len(acc_list_Attraction)
prec_mean_Attraction = sum(prec_list_Attraction) / len(prec_list_Attraction)
rec_mean_Attraction = sum(rec_list_Attraction) / len(rec_list_Attraction)
f1_mean_Attraction = sum(f1_list_Attraction) / len(f1_list_Attraction)
```

MEJORAS PROPUESTAS

Como "mejoras" tenemos:

1. Realizar una nueva limpieza de nuestro data set utilizando en esta ocasión NLTK para únicamente lematizar y tokenizar, a diferencia de la primera parte del proyecto donde utilizábamos SPACY.
2. Otra mejora propuesta fue modificar los hiperparámetros de nuestro modelo de regresión logística, específicamente el parámetro C que se encarga del inverso de la fuerza de la regularización, el cual reducimos a 0.5 para una regularización mas fuerte.

Otras mejoras que resultaron poco beneficiosas y que no se incluyeron en la versión final fue el balanceo y otros hiperparámetros de la regresión logística.

```
# Inicializa el modelo
model = LogisticRegression(max_iter=5000, C=0.5)
```

RESULTADOS

Predicción de polaridad					
Representación	Clasificador	Accuracy promedio	Precisión promedio	Recall Promedio	F-measure promedio
Binarizado	Regresión logística	0.724812	0.724812	0.724812	0.724812
	Naive Bayes	0.701228	0.701228	0.701228	0.701228
Frecuencia	Regresión logística	0.727005	0.727005	0.727005	0.727005
	Naive Bayes	0.704124	0.704124	0.704124	0.704124

Predicción de polaridad					
Representación	Clasificador	Accuracy promedio	Precisión promedio	Recall Promedio	F-measure promedio
Frecuencia	Regresión logística	0.7356	0.7356	0.7356	0.7356

RESULTADOS

Predicción de Lugar					
Representación	Clasificador	Accuracy promedio	Precisión promedio	Recall Promedio	F-measure promedio
Binarizado	Regresión logística	0.977822	0.977822	0.977822	0.977822
	Naive Bayes	0.971037	0.971037	0.971037	0.971037
Frecuencia	Regresión logística	0.977822	0.977822	0.977822	0.977822
	Naive Bayes	0.966940	0.966940	0.966940	0.966940

Predicción de Lugar					
Representación	Clasificador	Accuracy promedio	Precisión promedio	Recall Promedio	F-measure promedio
Frecuencias	Regresión logística	0.9820	0.9820	0.9820	0.9820

RESULTADOS REGRESION LOGISTICA FRECUENCIAL PROBANDO EL 20% DE LOS DATOS DE LUGAR

REGRESION LOGISTICA FRECUENCIAL PROBANDO EL 20% DE LOS ATTRACTION resultados:

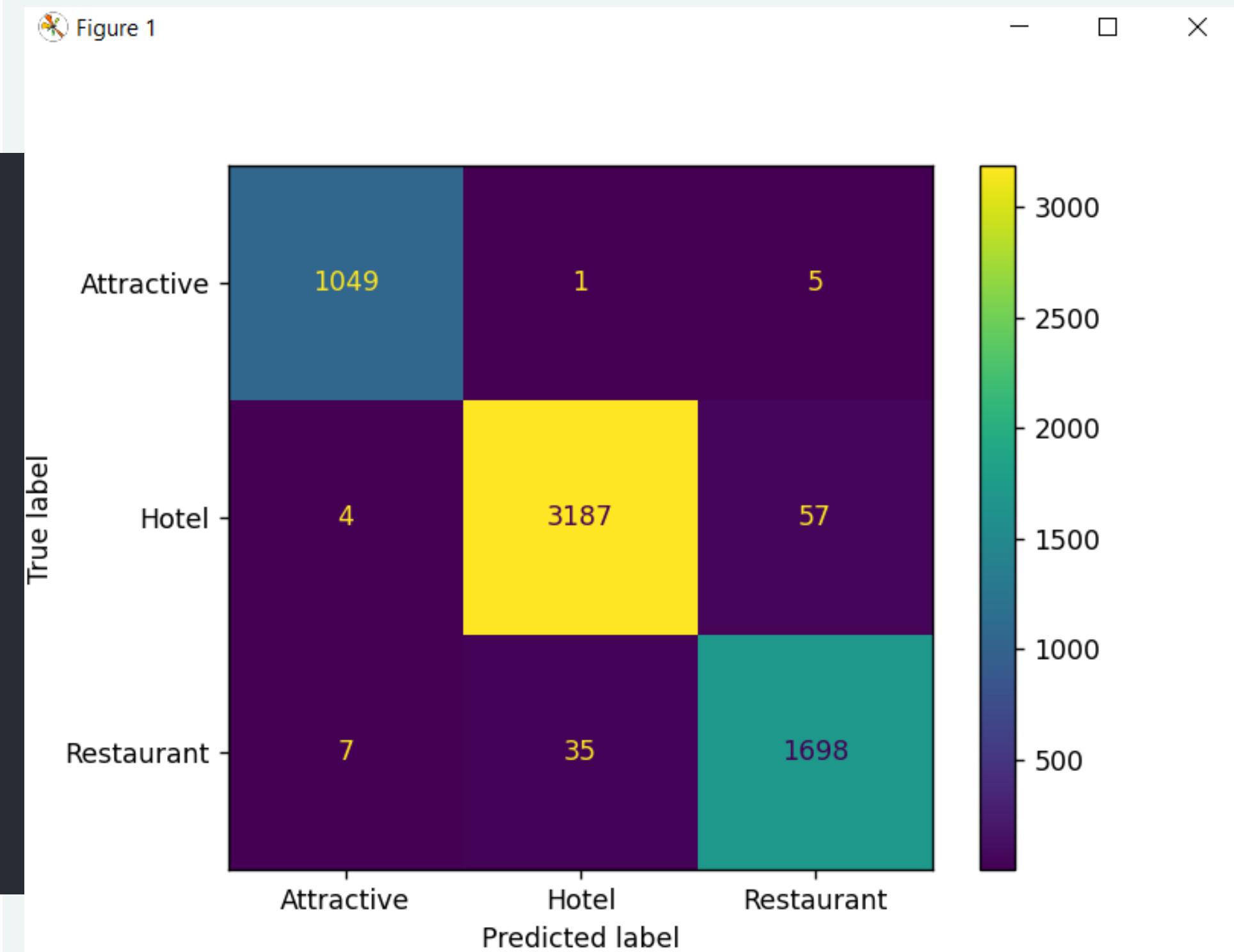
Accuracy: 0.9820

Precision: 0.9820

Recall: 0.9820

F-measure: 0.9820

	precision	recall	f1-score	support
Attractive	0.99	0.99	0.99	1055
Hotel	0.99	0.98	0.99	3248
Restaurant	0.96	0.98	0.97	1740
accuracy			0.98	6043
macro avg	0.98	0.98	0.98	6043
weighted avg	0.98	0.98	0.98	6043



RESULTADOS REGRESION LOGISTICA FRECUENCIAL PROBANDO EL 20% DE LOS DATOS DE POLARIDAD

POLARIDAD resultados

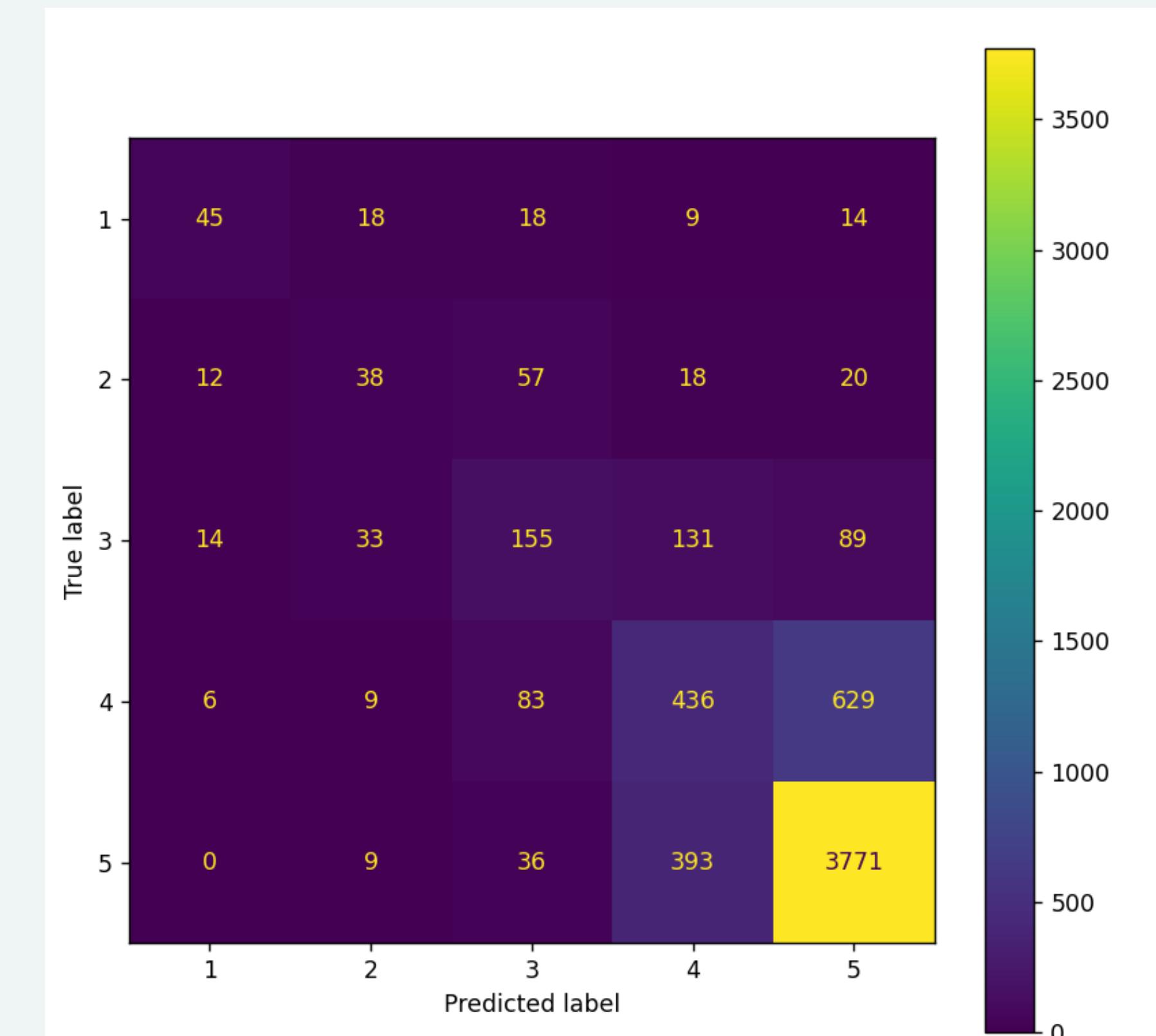
Accuracy: 0.7356

Precision: 0.7356

Recall: 0.7356

F-measure: 0.7356

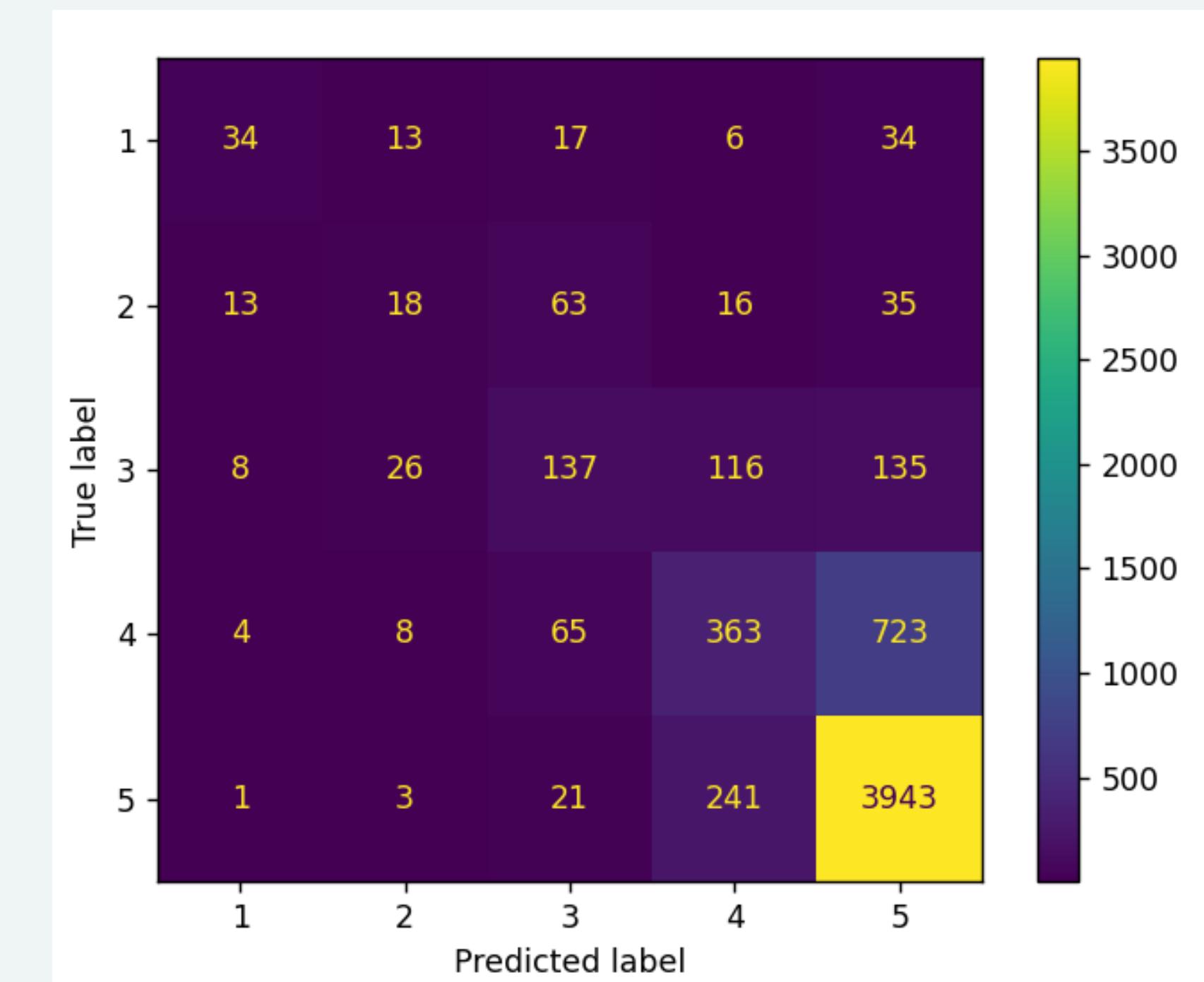
	precision	recall	f1-score	support
1	0.58	0.43	0.50	104
2	0.36	0.26	0.30	145
3	0.44	0.37	0.40	422
4	0.44	0.37	0.41	1163
5	0.83	0.90	0.86	4209
accuracy			0.74	6043
macro avg	0.53	0.47	0.49	6043
weighted avg	0.72	0.74	0.72	6043



RESULTADOS REGRESION LOGISTICA FRECUENCIAL PROBANDO EL 20% DE LOS DATOS DE POLARIDAD extra

Ajuste de C = 0.005:

POLARIDAD resultados				
Accuracy: 0.7438				
Precision: 0.7438				
Recall: 0.7438				
F-measure: 0.7438				
	precision	recall	f1-score	support
1	0.57	0.33	0.41	104
2	0.26	0.12	0.17	145
3	0.45	0.32	0.38	422
4	0.49	0.31	0.38	1163
5	0.81	0.94	0.87	4209
accuracy			0.74	6043
macro avg	0.52	0.40	0.44	6043
weighted avg	0.71	0.74	0.72	6043



The background of the image is a black and white aerial photograph of a city's skyline, featuring a high density of skyscrapers and buildings.

¡Muchas gracias!

Página de recursos

Usa estos iconos e ilustraciones en tu presentación de Canva. ¡Que lo pases bien diseñando!

