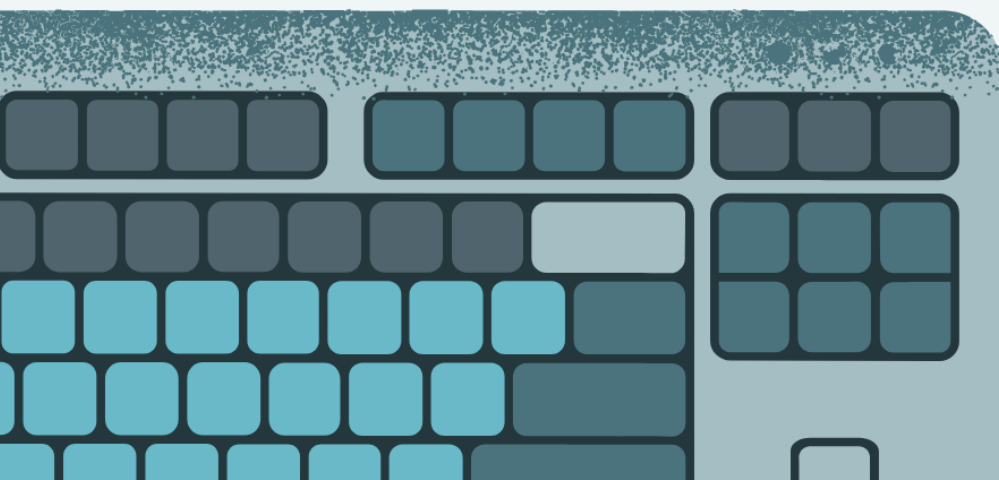


Guillermo Bárcena López

IDENTIFICADOR DE RESEÑAS

- Desarrollo de Aplicaciones Multiplataforma
- Acceso a datos
- Fernando Usero
- I.E.S Ramón del Valle Inclán



1. ¿Qué es la clasificación de texto?	5
2. Principales modelos utilizados en clasificación de texto:	5
2.1 Modelos tradicionales	5
2.1.1 Naive Bayes	5
2.1.2 Regresión Logística	5
2.1.3 SVM	6
2.2 Modelos basados en redes neuronales	6
2.2.1 LSTM	6
2.2.2 CNNs	6
2.3 Modelos preentrenados	7
2.3.1 BERT	7
2.3.2 DistilBERT	7
2.3.3 GPT	7
3. Técnicas de procesamiento de texto	8
3.1 Tokenización	8
3.2 Eliminación de Stopwords	9
3.3 Lematización	9
3.4 Stemming	10
4. Representación del texto	11
4.1 TF-IDF (Term Frequency - Inverse Document Frequency)	11
4.2 Word Embeddings (Word2Vec, GloVe, FastText)	12
4.3 Representaciones Preentrenadas (BERT, GPT, etc.)	12
5. Elección del dataset	13
Reseña Positiva (label = 2)	13
Reseña Negativa (label = 1)	13
6. Explicación del Script	14
6.1 Carga del dataset	14
6.2 Exploración de los datos	14
6.3 Procesamiento del texto	15
6.4 Representación Numérica del Texto con TF-IDF	16
6.5 División de Datos en Entrenamiento y Prueba	16
6.6 Entrenamiento del Modelo Naive Bayes	16
6.7 Entrenamiento del Modelo Naive Bayes	16
6.8 Predicción con Nuevas Reseñas	17
6.9 Resultado esperado	18
Anexo 1: Código	21

1. ¿Qué es la clasificación de texto?

La clasificación de textos es una de las aplicaciones más importantes del machine learning y consiste en catalogar textos en función de lo que se hable en él, es decir, analizar palabra a palabra para decidir qué características presenta el texto.

Un ejemplo sería para decidir qué tipo de noticia nos da un texto:

- Deportiva
- Económica
- Prensa rosa

Este tipo de predicción se usa en muchísimas empresas en la vida real, por ejemplo para clasificar el tipo de solicitudes de atención al cliente y diferenciarlo sobre qué departamento debería resolver esta incidencia.

También es usado para diferenciar los correos de Spam y mandarlos a la carpeta que le corresponde.

Otro de los usos principales es el análisis de sentimientos, el análisis de sentimientos es una técnica de Procesamiento de Lenguaje Natural (NLP) que permite determinar la opinión o emoción expresada en un texto. Su objetivo es clasificar textos en categorías como positivas, negativas o neutrales. En nuestro caso vamos a usar este análisis de sentimientos para calificar las reseñas.

2. Principales modelos utilizados en clasificación de texto:

2.1 Modelos tradicionales

2.1.1 Naive Bayes

Naive Bayes es un algoritmo basado en el Teorema de Bayes, que calcula la probabilidad de que un documento pertenezca a una clase determinada. Se llama "Naive" porque asume que todas las palabras en el texto son independientes entre sí, lo cual no es del todo cierto en el lenguaje natural.

Un sistema de filtrado de correos electrónicos como Gmail usa Naive Bayes para detectar si un correo es spam o no spam.

2.1.2 Regresión Logística

La regresión logística es un modelo de clasificación que estima la probabilidad de pertenencia de una instancia a una clase utilizando una función logística o sigmoide.

Un sistema de análisis de reseñas como Amazon Reviews puede usar Regresión Logística para predecir si una reseña es positiva o negativa.

2.1.3 SVM

SVM es un algoritmo que encuentra un hiperplano óptimo que separa los datos en clases con el mayor margen posible.

Un agregador de noticias como Google News puede usar SVM para clasificar artículos en categorías como "Deportes", "Política" o "Tecnología".

2.2 Modelos basados en redes neuronales

2.2.1 LSTM

LSTM es un tipo de red neuronal recurrente (RNN) diseñado para manejar secuencias largas de texto. A diferencia de una RNN tradicional, LSTM tiene una memoria más efectiva para recordar palabras clave a largo plazo y evitar el problema del desvanecimiento del gradiente.

Los chatbots y asistentes virtuales utilizan LSTM para entender el contexto de una conversación.

Por qué funciona bien:

- Puede recordar información de mensajes anteriores en una conversación.
- Maneja frases largas con coherencia.
- Entiende secuencias temporales, como seguir el hilo de una pregunta/respuesta.

2.2.2 CNNs

Las Redes Neuronales Convolucionales (CNNs), aunque son más conocidas en visión por computadora, también se pueden usar en clasificación de texto. En lugar de imágenes, aplican filtros a secuencias de palabras o caracteres.

Las redes sociales utilizan CNNs para detectar discursos de odio, acoso y spam en comentarios.

Por qué funciona bien:

- Captura patrones de palabras en una ventana pequeña sin importar su posición en la oración.
- Es eficiente y rápido en comparación con LSTM.
- Funciona bien en clasificación de oraciones cortas.

2.3 Modelos preentrenados

2.3.1 BERT

BERT es un modelo basado en Transformers que entiende el contexto de una oración considerando las palabras antes y después de una palabra clave. Esto lo hace muy útil para tareas de comprensión del lenguaje natural.

Google usa BERT para mejorar los resultados de búsqueda interpretando el significado completo de la consulta en lugar de solo palabras clave.

Por qué funciona bien:

- Analiza el contexto bidireccionalmente (palabras antes y después).
- Mejora la precisión en preguntas complejas y frases ambiguas.
- Es útil en clasificación de texto, resúmenes y respuestas a preguntas.

2.3.2 DistilBERT

DistilBERT es una versión más ligera y rápida de BERT, con casi el mismo rendimiento pero usando menos recursos. Es ideal para aplicaciones en tiempo real.

Empresas como Netflix, McDonald's o Tesla analizan tweets de los usuarios para medir la opinión sobre sus productos o servicios.

Por qué funciona bien:

- Es más rápido que BERT, ideal para grandes volúmenes de datos.
- Detecta sarcasmo, doble sentido y contexto.
- Puede ejecutarse en dispositivos con menos capacidad.

2.3.3 GPT

GPT es un modelo de lenguaje autoregresivo que predice la siguiente palabra en función del contexto anterior, lo que lo hace excelente para generación de texto.

GPT se usa para generar resúmenes de noticias, redacción de blogs y automatización de contenido.

Por qué funciona bien:

- Genera texto fluido y coherente.
- Puede responder preguntas, escribir resúmenes o incluso programar código.
- Aprende de grandes volúmenes de texto sin necesidad de etiquetado.

3. Técnicas de procesamiento de texto

El preprocesamiento de texto es un paso esencial en el procesamiento del lenguaje natural (NLP), ya que permite limpiar y estructurar el texto para que los modelos de Machine Learning lo interpreten de manera más eficiente

3.1 Tokenización

La tokenización consiste en dividir un texto en unidades más pequeñas llamadas tokens. Un token puede ser una palabra, una oración o un carácter, dependiendo del tipo de tokenización utilizada.

Vamos a poner un ejemplo

Texto original:

- "El perro corre rápido."

Tokenización por palabras:

- ["El", "perro", "corre", "rápido", "."]

Tokenización por oraciones:

- ["El perro corre rápido."]

Librerías para tokenización en Python:

- `nltk.word_tokenize(texto)`
- `spacy.nlp(texto)`
- `text.split()`

3.2 Eliminación de Stopwords

Las stopwords son palabras comunes en un idioma que no aportan mucho significado en el análisis, como "el", "la", "de", "en".

Vamos a poner un ejemplo

Texto original:

- "El perro corre rápido."

Después de eliminar stopwords:

- ["perro", "corre", "rápido", "parque"]

Librerías para eliminar stopwords:

- *nltk.corpus.stopwords*
- *spacy.lang.es.stop_words*

3.3 Lematización

La lematización convierte una palabra en su forma base o lema, teniendo en cuenta su significado en el contexto.

Vamos a poner un ejemplo

Texto original:

- "Los niños corrieron por el parque."

Después de lematización:

- ["niño", "correr", "parque"]

Librerías para eliminar lematización:

- *spacy.nlp().lemma_*
- *nltk.WordNetLemmatizer()*

3.4 Stemming

El stemming reduce una palabra a su raíz, sin considerar el contexto o significado.

Vamos a poner un ejemplo

Texto original:

- "Jugando, jugamos, jugaron."

Después de stemming:

- ["Jug"]

El algoritmo simplemente recorta la palabra, lo que puede generar palabras sin sentido.

Librerías para eliminar stemming:

- *nltk.PorterStemmer()*
- *nltk.SnowballStemmer()*

4. Representación del texto

Después del preprocesamiento, es necesario convertir el texto en una representación numérica que pueda ser utilizada por los modelos de aprendizaje automático.

4.1 TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF mide la importancia de una palabra en un documento dentro de un conjunto de documentos.

Fórmula:

$$TF - IDF = TF \times IDF$$

- TF (Frecuencia de Término): Número de veces que aparece una palabra en el documento.
- IDF (Frecuencia Inversa del Documento): Penaliza palabras muy comunes en todo el corpus.

Documento	Palabra "perro"	Palabra "parque"
Doc1	3 veces	1 vez
Doc2	0 veces	5 veces

Aquí, "perro" tendrá un TF-IDF mayor en Doc1, mientras que "parque" será más importante en Doc2.

Librerías en Python:

- `sklearn.feature_extraction.text.TfidfVectorizer()`

4.2 Word Embeddings (Word2Vec, GloVe, FastText)

Los Word Embeddings representan palabras en un espacio de varias dimensiones, donde palabras con significado similar tienen vectores cercanos.

Si entrenamos un modelo con Word2Vec, podemos ver relaciones entre palabras como:

$$- \text{ "Rey" - "Hombre" + "Mujer" = "Reina" }$$

Esto significa que "rey" y "reina" tienen una relación semántica similar a "hombre" y "mujer".

Librerías en Python:

- `gensim.models.Word2Vec`
- `spacy.load("es_core_news_md")`

4.3 Representaciones Preentrenadas (BERT, GPT, etc.)

Los modelos como BERT y GPT utilizan embeddings preentrenados que consideran el contexto completo de una palabra en una oración.

En la frase:

$$- \text{ "El banco de peces nadaba cerca del banco de arena." }$$

BERT entiende que "banco" tiene dos significados distintos según el contexto.

Librerías en Python:

- `transformers.BertTokenizer`
- `transformers.BertModel`

5. Elección del dataset

El Amazon Reviews Dataset es un conjunto de datos que recopila reseñas de productos realizadas por clientes en Amazon a lo largo de 18 años, hasta marzo de 2013, sumando aproximadamente 35 millones de reseñas. Este extenso dataset es ampliamente utilizado en proyectos de análisis de sentimientos, recomendaciones de productos y otras aplicaciones relacionadas con el procesamiento del lenguaje natural.

Columnas del dataset:

- **label:** Indica si la reseña es positiva (2) o negativa (1).
- **title:** Es el título de la reseña, un breve resumen proporcionado por el usuario.
- **content:** Es el texto completo de la reseña, donde el usuario describe su experiencia con el producto.

Reseña Positiva (label = 2)

label	title	content
2	"Excelente producto"	"Me encantó este libro, la historia es atrapante y no pude dejar de leerlo. Lo recomiendo totalmente."

Reseña Negativa (label = 1)

label	title	content
1	"Mala calidad"	"El producto llegó roto y no funcionó como esperaba. No volvería a comprarlo."

Este conjunto de datos se puede usar para entrenar un modelo de clasificación de texto, donde el objetivo es predecir si una reseña es positiva o negativa a partir del título y el contenido de la misma.

6. Explicación del Script

Este script implementa un modelo de Análisis de Sentimientos utilizando el algoritmo de Naive Bayes Multinomial. Se basa en el procesamiento de reseñas de productos de Amazon para predecir si una reseña es positiva o negativa.

El modelo se entrena con un conjunto de reseñas y luego se evalúa con datos de prueba. Finalmente, se realizan predicciones sobre nuevas reseñas ingresadas manualmente.

6.1 Carga del dataset

```
columnas = ["label", "title", "content"]  
df = pd.read_csv("amazon_reviews.csv", names=columnas, header=None)
```

El script carga el dataset desde un archivo CSV, donde las columnas son:

- label: Indica si la reseña es positiva (2) o negativa (1).
- title: Título de la reseña (no se usa en el modelo).
- content: Contenido completo de la reseña.

6.2 Exploración de los datos

```
print(df.head())  
print(df.isnull().sum())
```

Se imprimen las primeras filas del dataset y se verifica si existen valores nulos.

Si hay valores nulos en la columna content, se eliminan:

```
df.dropna(subset=['content'], inplace=True)
```

6.3 Procesamiento del texto

Se verifica si las stopwords de NLTK están instaladas y, en caso contrario, se descargan.

```
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')
```

Las stopwords son palabras comunes como "the", "is", "and", que no aportan valor en el análisis de sentimientos y deben eliminarse.

```
stop_words = set(stopwords.words('english'))
```

Esta función realiza varias transformaciones en el texto:

- Convierte todo a minúsculas para normalizar las palabras.
- Elimina caracteres especiales para dejar solo palabras.
- Elimina espacios en blanco adicionales.
- Divide el texto en palabras sin usar word_tokenize(), lo que lo hace más rápido.
- Elimina stopwords para reducir ruido en los datos.

```
def limpiar_texto(texto):
    if isinstance(texto, str):
        texto = texto.lower() # Convertir a minúsculas
        texto = re.sub(r'\W', ' ', texto) # Eliminar caracteres
especiales
        texto = re.sub(r'\s+', ' ', texto).strip() # Eliminar espacios
extra
        palabras = texto.split() # Separar palabras sin usar
word_tokenize()
        palabras = [palabra for palabra in palabras if palabra not in
stop_words] # Eliminar stopwords
        return " ".join(palabras)
    return ""
```

6.4 Representación Numérica del Texto con TF-IDF

```
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df["content_clean"])
y = df["label"]
```

TF-IDF (Term Frequency - Inverse Document Frequency) convierte los textos en vectores numéricos para que puedan ser utilizados en el modelo.

- `max_features=5000` selecciona solo las 5000 palabras más relevantes.

6.5 División de Datos en Entrenamiento y Prueba

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)
```

Esta función significa:

- 70% de los datos se usan para entrenar el modelo.
- 30% de los datos se usa para evaluar el modelo.
- `stratify=y` asegura que ambas clases (positiva y negativa) tengan representación equitativa en los datos de prueba.

6.6 Entrenamiento del Modelo Naive Bayes

```
modelo_nb = MultinomialNB()
modelo_nb.fit(X_train, y_train)
```

Se utiliza el modelo **Multinomial Naive Bayes**, que es ideal para clasificación de texto basada en **frecuencia de palabras**.

6.7 Entrenamiento del Modelo Naive Bayes

```
y_pred = modelo_nb.predict(X_test)
print("\nResultados del modelo:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Se evalúa el modelo con:

- `accuracy_score()` → Porcentaje de predicciones correctas.
- `classification_report()` → Incluye precision, recall y F1-score.

6.8 Predicción con Nuevas Reseñas

```
nuevas_reseñas_clean = [limpiar_texto(texto) for texto in nuevas_reseñas]
X_nuevas = vectorizer.transform(nuevas_reseñas_clean)

predicciones = modelo_nb.predict(X_nuevas)

print("\nPredicciones para nuevas reseñas:")
for review, label in zip(nuevas_reseñas, predicciones):
    print(f"Reseña: {review} --> Predicción: {'Positiva' if label == 2 else 'Negativa'}")
```






Se ingresan tres nuevas reseñas para probar el modelo.

Cada reseña se limpia y se convierte en vector numérico, luego el modelo predice si es positiva o negativa.

Las reseñas que vamos a utilizar van a ser del libro de Clean Code, el cual es bastante famosos entre programadores.

Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series) > Opiniones de clientes

Opiniones de clientes
★★★★★ 4,8 de 5
6.275 valoraciones globales


5 estrellas  84%
4 estrellas  12%
3 estrellas  3%
2 estrellas  0%
1 estrella  1%

[Escribir una opinión](#)

[Cómo funcionan las opiniones y las valoraciones de los clientes](#) ▼

Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)
por Martin Robert

1ª Reseña: Positiva



Jose Luis Rodriguez Villapecellin

★★★★★ **Best practices book that ever exist**
Revisado en España el 15 de febrero de 2014
Compra verificada

This book describe the way to get the very best practices in software development.
I recommend it for every one who want to develop better programs.

[Útil](#) | [Denunciar](#)

2ª Reseña: Positiva



Davit-A

★★★★★ **This is a must be**

Revisado en España el 24 de octubre de 2015

Compra verificada

This is a kind of programming bible.

Every programmer must read this book to get good practices and to refresh the idea of how beauty it is to write beauty, clean and perfect code.

Útil

Denunciar

3ª Reseña: Negativa



Lofty Snowman

★☆☆☆☆ **Laughably poor print quality**

Revisado en los Estados Unidos el 28 de agosto de 2016

Compra verificada

As mentioned in other reviews, the book itself is low quality. The cover image is blurry, there is no print on the binding, and there are pointlessly large margins surrounding all of the text. On top of that, the printed font is grainy and difficult to read.

I would rather track down a proper printing of this product, than try to trudge through reading this. As a comparison, Code Complete is three times longer than this book. However, I was able to easily hold and read that book. This version of Clean Code feels like you are holding a ream of printer paper, with every page printed in portrait mode.

It is unwieldy to say the least.



A 85 personas les ha parecido esto útil

Denunciar

[Traducir reseña a Español](#)

6.9 Resultado esperado

Resultados del modelo:				
Accuracy: 0.8208138888888888				
	precision	recall	f1-score	support
1	0.82	0.82	0.82	540000
2	0.82	0.82	0.82	540000
accuracy			0.82	1080000
macro avg	0.82	0.82	0.82	1080000
weighted avg	0.82	0.82	0.82	1080000


El modelo tiene un 82.08% de precisión global:

- Esto significa que el modelo clasificó correctamente el 82.08% de las reseñas en el conjunto de prueba.
- En este caso, el 82% de las reseñas que el modelo clasificó como positivas (2) realmente eran positivas.

- Igualmente, el 82% de las reseñas que el modelo clasificó como **negativas (1)** realmente eran negativas.

```
Predicciones para nuevas reseñas:  
Reseña: This book describe the way to get the very best practices in software development. I... --> Predicción: Positiva  
Reseña: This is a kind of programming bible. Every programmer must read this book to get... --> Predicción: Positiva  
Reseña: As mentioned in other reviews, the book itself is low quality. The cover image is... --> Predicción: Negativa
```

- Se muestran solo **las primeras 15 palabras** de cada reseña para hacer la salida más legible. Al final de cada línea se indica la **predicción del modelo**.

 Jose Luis Rodriguez Villapeccellin

★★★★★ **Best practices book that ever exist**

Revisado en España el 15 de febrero de 2014


Compra verificada

This book describe the way to get the very best practices in software development.
I recommend it for every one who want to develop better programs.

Útil | Denunciar

El modelo la clasificó como Positiva.

- Usa términos como *"best practices"*, *"recommend"*, *"better programs"*, lo que indica un tono positivo.

 Davit-A

★★★★★ **This is a must be**

Revisado en España el 24 de octubre de 2015

Compra verificada

This is a kind of programming bible.
Every programmer must read this book to get good practices and to refresh the idea of how beauty it is to write beauty, clean and perfect code.

Útil | Denunciar

El modelo la clasificó como Positiva.

- Contiene palabras como *"programming bible"*, *"must read"*, *"good practices"*, que sugieren una reseña favorable.



Lofty Snowman

★☆☆☆☆ **Laughably poor print quality**

Revisado en los Estados Unidos el 28 de agosto de 2016

Compra verificada

As mentioned in other reviews, the book itself is low quality. The cover image is blurry, there is no print on the binding, and there are pointlessly large margins surrounding all of the text. On top of that, the printed font is grainy and difficult to read.

I would rather track down a proper printing of this product, than try to trudge through reading this. As a comparison, Code Complete is three times longer than this book. However, I was able to easily hold and read that book. This version of Clean Code feels like you are holding a ream of printer paper, with every page printed in portrait mode.

It is unwieldy to say the least.



A 85 personas les ha parecido esto útil

[Denunciar](#)

[Traducir reseña a Español](#)

El modelo la clasificó como Negativa.

- Expresiones como "*low quality*", "*blurry*", "*difficult to read*" indican una experiencia negativa con el libro.

Anexo 1: Código

```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Cargar el dataset
columnas = ["label", "title", "content"]
df = pd.read_csv("amazon_reviews.csv", names=columnas, header=None)

# Explorar los datos
print("Primeras filas del dataset:")
print(df.head())

print("\nCantidad de valores nulos por columna:")
print(df.isnull().sum())

# Eliminar filas con valores nulos en 'content'
df.dropna(subset=['content'], inplace=True)

# Descargar los recursos de NLTK si no están disponibles
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

# Cargar stopwords en inglés
stop_words = set(stopwords.words('english'))

# Función de preprocesamiento de texto
def limpiar_texto(texto):
    if isinstance(texto, str):
        texto = texto.lower() # Convertir a minúsculas
```

```

        texto = re.sub(r'\W', ' ', texto) # Eliminar caracteres especiales
        texto = re.sub(r'\s+', ' ', texto).strip() # Eliminar espacios extra
        palabras = texto.split() # Separar palabras sin usar word_tokenize()
        palabras = [palabra for palabra in palabras if palabra not in
stop_words] # Eliminar stopwords
        return " ".join(palabras)
    return ""

# Aplicar limpieza de texto
print("\nIniciando limpieza de texto...")
df["content_clean"] = df["content"].apply(limpiar_texto)
print("Limpieza de texto completada.")

# Verificar si hay valores vacíos después de la limpieza
print("\nValores nulos después de limpieza:")
print(df["content_clean"].isnull().sum())

# Representación numérica del texto con TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df["content_clean"])
y = df["label"]

# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

# Entrenar el modelo Naive Bayes
modelo_nb = MultinomialNB()
modelo_nb.fit(X_train, y_train)

# Evaluar el modelo
y_pred = modelo_nb.predict(X_test)
print("\nResultados del modelo:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Predicción con nuevas reseñas
nuevas_reseñas = [
    "This book describe the way to get the very best practices in software
development. I recommend it for every one who want to develop better
programs.",

```

"This is a kind of programming bible. Every programmer must read this book to get good practices and to refresh the idea of how beauty it is to write beauty, clean and perfect code.",

"As mentioned in other reviews, the book itself is low quality. The cover image is blurry, there is no print on the binding, and there are pointlessly large margins surrounding all of the text. On top of that, the printed font is grainy and difficult to read. I would rather track down a proper printing of this product, than try to trudge through reading this. As a comparison, Code Complete is three times longer than this book. However, I was able to easily hold and read that book. This version of Clean Code feels like you are holding a ream of printer paper, with every page printed in portrait mode. It is unwieldy to say the least."

]

```
nuevas_reseñas_clean = [limpiar_texto(texto) for texto in nuevas_reseñas]
X_nuevas = vectorizer.transform(nuevas_reseñas_clean)
```

```
predicciones = modelo_nb.predict(X_nuevas)
```

```
print("\nPredicciones para nuevas reseñas:")
```

```
for review, label in zip(nuevas_reseñas, predicciones):
```

```
    # Tomar las primeras 15 palabras de la reseña
```

```
    resumen_resena = " ".join(review.split()[:15]) + "..."
```

```
    print(f"Reseña: {resumen_resena} --> Predicción: {'Positiva' if label == 2
else 'Negativa'}")
```

```
print("\nFin del script.")
```