Pasos previos:

activar ambiente tuenvironment\Scripts\activate

pip install --pre torch --index-url https://download.pytorch.org/whl/nightly/cu128

pip install transformers==4.30.0

In [32]:

```
import torch

# Verificar si CUDA está disponible
if torch.cuda.is_available():
    device = torch.device("cuda")
    print("✅ CUDA disponible. Usando GPU:", torch.cuda.get_device_name(0))
else:
    device = torch.device("cpu")
    print("⚠️ CUDA no disponible. Usando CPU.")
```

Out[32]:

```
✅ CUDA disponible. Usando GPU: NVIDIA GeForce RTX 3070
```

In [7]:

```
import kagglehub
import os

# Descargar dataset
path = kagglehub.dataset_download("luthfim/steam-reviews-dataset")
print("Archivos disponibles:", os.listdir(path))
```

Out[7]:

```
Warning: Looks like you're using an outdated `kagglehub` version (installed:
0.3.11), please consider upgrading to the latest version (0.3.12).
Archivos disponibles: ['steam_reviews.csv']
```

In [8]:

```
import pandas as pd

# Cargar CSV con encoding y limpiar nombres de columnas
csv_path = os.path.join(path, "steam_reviews.csv")
```

```
df = pd.read_csv(csv_path, encoding="utf-8")
df.columns = df.columns.str.strip()

# Ver columnas
print("Columnas disponibles:", df.columns.tolist())
```

Out[8]:

```
Columnas disponibles: ['date_posted', 'funny', 'helpful', 'hour_played',
'is_early_access_review', 'recommendation', 'review', 'title']
```

In [9]:

```
# Seleccionar y renombrar columnas
df = df[['review', 'recommendation']].copy()
df.rename(columns={'review': 'review_text', 'recommendation': 'recommended'}, inpla

# Convertir a etiquetas binarias
df['label'] = df['recommended'].apply(lambda x: 1 if x == 'Recommended' else 0)

# Filtrar textos nulos o vacíos
df = df[df['review_text'].notnull() & (df['review_text'].str.strip() != '')]

# Balancear clases
min_class = df['label'].value_counts().min()
df_balanced = pd.concat([
    df[df['label'] == 0].sample(min_class, random_state=42),
    df[df['label'] == 1].sample(min_class, random_state=42)
])

print("Distribución balanceada:\n", df_balanced['label'].value_counts())
```

Out[9]:

```
Distribución balanceada:
 label
0    130624
1    130624
Name: count, dtype: int64
```

In [10]:

```
from sklearn.model_selection import train_test_split
```

```
train_df, test_df = train_test_split(df_balanced, test_size=0.2, stratify=df_balanc
```

In [12]:

```
from datasets import Dataset
from transformers import AutoTokenizer

model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize(example):
    return tokenizer(example["review_text"], truncation=True, padding="max_length",

# Convertir a datasets
train_dataset = Dataset.from_pandas(train_df[['review_text', 'label']])
test_dataset = Dataset.from_pandas(test_df[['review_text', 'label']])

# Aplicar tokenización
train_dataset = train_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)
```

Out[12]:

```
Map:   0%|          | 0/208998 [00:00<?, ? examples/s]
```

Out[12]:

```
Map:   0%|          | 0/52250 [00:00<?, ? examples/s]
```

In [13]:

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2
```

Out[13]:

```
Some weights of DistilBertForSequenceClassification were not initialized from the
model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

In [14]:

```
import evaluate
import numpy as np

accuracy_metric = evaluate.load("accuracy")
f1_metric = evaluate.load("f1")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return {
        "accuracy": accuracy_metric.compute(predictions=preds, references=labels)["
        "f1": f1_metric.compute(predictions=preds, references=labels, average="weig
    }
```

In [15]:

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    do_train=True,
    do_eval=True,
    per_device_train_batch_size=32,  # Tamaño de lote para entrenamiento
    per_device_eval_batch_size=64,   # Tamaño de lote para evaluación
    num_train_epochs=5,  # Aumentar el número de épocas para entrenar más
    weight_decay=0.01,  # Regularización para evitar sobreajuste
    logging_dir='./logs',
    logging_steps=200,
    save_steps=None,  # No es necesario save_steps cuando no hay evaluación
    save_total_limit=2,
    save_strategy="epoch",  # Guardar después de cada época
    metric_for_best_model="accuracy",  # Métrica para determinar el mejor modelo
    greater_is_better=True,
    no_cuda=not torch.cuda.is_available(),
    fp16=True,  # Usar precisión de 16 bits para acelerar el entrenamiento
    seed=42,  # Semilla para reproducibilidad
    learning_rate=3e-5,  # Ajustar la tasa de aprendizaje para un entrenamiento más
    warmup_steps=500,  # Calentamiento de pasos para evitar inestabilidad al inicio
)
```

In [16]:

```
from transformers import Trainer
import time

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics
)

start = time.time()
trainer.train()
print("⏱️ Tiempo total de entrenamiento:", round(time.time() - start, 2), "segundos"
```

Out[16]:

[32660/32660 43:02, Epoch 5/5]

Step	Training Loss
200	0.594400
400	0.396000
600	0.354000
800	0.345400
1000	0.329300
1200	0.314700
1400	0.313900
1600	0.300400
1800	0.286800
2000	0.301800
2200	0.309800
2400	0.289000
2600	0.284600
2800	0.277900
3000	0.289100
3200	0.277900
3400	0.280800
3600	0.280500

Step	Training Loss
3800	0.274300
4000	0.272000
4200	0.275900
4400	0.285000
4600	0.269100
4800	0.269200
5000	0.287100
5200	0.268600
5400	0.261700
5600	0.273400
5800	0.262000
6000	0.265000
6200	0.273100
6400	0.265500
6600	0.235100
6800	0.217200
7000	0.217100
7200	0.205100
7400	0.219500
7600	0.209900
7800	0.194600
8000	0.222000
8200	0.203900
8400	0.204100
8600	0.225700
8800	0.221600
9000	0.216500
9200	0.215900
9400	0.218500
9600	0.223100
9800	0.208200
10000	0.210900

Step	Training Loss
10200	0.219600
10400	0.211900
10600	0.212500
10800	0.216500
11000	0.217600
11200	0.223000
11400	0.212500
11600	0.225000
11800	0.207500
12000	0.213500
12200	0.210200
12400	0.221500
12600	0.218400
12800	0.218500
13000	0.220800
13200	0.178500
13400	0.153400
13600	0.161200
13800	0.147900
14000	0.160700
14200	0.146500
14400	0.150400
14600	0.156400
14800	0.162400
15000	0.164300
15200	0.152000
15400	0.174900
15600	0.154100
15800	0.160000
16000	0.166000
16200	0.160600
16400	0.166200

Step	Training Loss
16600	0.155100
16800	0.163500
17000	0.159200
17200	0.168800
17400	0.156900
17600	0.152500
17800	0.147900
18000	0.158800
18200	0.157900
18400	0.148900
18600	0.169600
18800	0.154200
19000	0.155800
19200	0.155900
19400	0.151500
19600	0.165000
19800	0.109000
20000	0.110100
20200	0.112300
20400	0.101200
20600	0.119100
20800	0.123500
21000	0.104800
21200	0.118500
21400	0.116400
21600	0.118000
21800	0.116000
22000	0.108100
22200	0.118800
22400	0.117100
22600	0.122000
22800	0.120300

Step	Training Loss
23000	0.117500
23200	0.108300
23400	0.120400
23600	0.103000
23800	0.123600
24000	0.116500
24200	0.119700
24400	0.110600
24600	0.125100
24800	0.118100
25000	0.104500
25200	0.119500
25400	0.122500
25600	0.103500
25800	0.111600
26000	0.110100
26200	0.101400
26400	0.093800
26600	0.090300
26800	0.081100
27000	0.086600
27200	0.089100
27400	0.076500
27600	0.085000
27800	0.089600
28000	0.096400
28200	0.092200
28400	0.087500
28600	0.085500
28800	0.087100
29000	0.077800
29200	0.088400

Step	Training Loss
29400	0.082900
29600	0.090200
29800	0.089300
30000	0.086700
30200	0.091800
30400	0.077400
30600	0.088700
30800	0.077800
31000	0.092200
31200	0.082900
31400	0.092200
31600	0.079500
31800	0.082600
32000	0.089200
32200	0.092900
32400	0.082600
32600	0.079200

Out[16]:

```
⏱️ Tiempo total de entrenamiento: 2583.01 segundos
```

In [17]:

```
results = trainer.evaluate()
print("📊 Resultados finales:", results)
```

Out[17]:

Out[17]:

```
📊 Resultados finales: {'eval_loss': 0.45394834876060486, 'eval_accuracy':
0.8936076555023923, 'eval_f1': 0.89360626341314, 'eval_runtime': 31.5706,
'eval_samples_per_second': 1655.019, 'eval_steps_per_second': 25.878, 'epoch':
5.0}
```

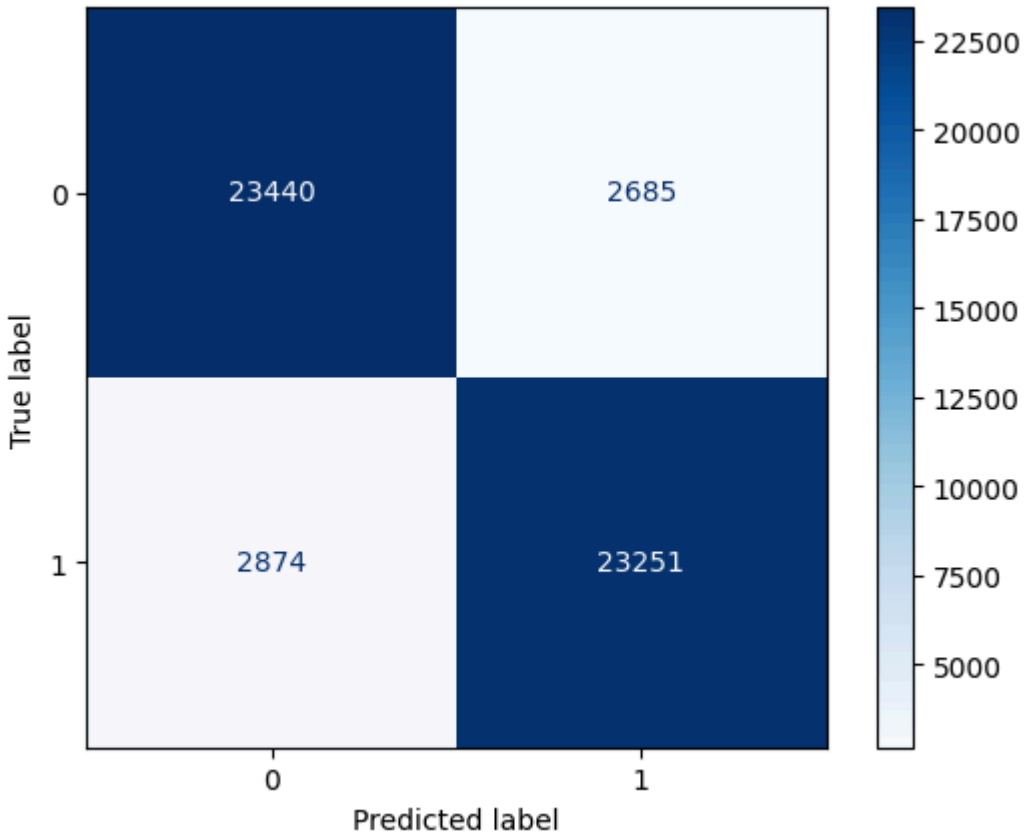
In [18]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Primero, hacemos predicciones sobre el set de evaluación
predictions = trainer.predict(trainer.eval_dataset)
preds = np.argmax(predictions.predictions, axis=1)  # Tomamos la clase con mayor pr
labels = predictions.label_ids  # Las etiquetas verdaderas

# Ahora calculamos la matriz de confusión
cm = confusion_matrix(labels, preds)

# Mostramos la matriz de confusión
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues', values_format='d')  # 'd' para que muestre enteros
```

Out[18]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x276c4448d70>
```

Out[18]:



In [27]:

```
predictions = trainer.predict(test_dataset)
preds = predictions.predictions.argmax(-1)
labels = predictions.label_ids
```

Out[27]:

In [28]:

```
# Si usaste pandas para armar tu dataset:
df_test = test_dataset.to_pandas()

# Agregamos predicciones
df_test['prediction'] = preds
df_test['label'] = labels
```

In [29]:

```
# Aciertos
correct = df_test[df_test['prediction'] == df_test['label']]

# Errores
wrong = df_test[df_test['prediction'] != df_test['label']]
```

In [31]:

```
# Mostrar 2 aciertos y 2 errores
ejemplos = pd.concat([
    correct.sample(2, random_state=41),
    wrong.sample(2, random_state=41)
])[['review_text', 'label', 'prediction']]
print(ejemplos)
```

Out[31]:

	review_text	label	prediction
6693	One of the worst games you can play right now....	0	0
5710	36$ so i can stare at failed connection screen?	0	0
30194	toxic	1	0
28487	This game rocks but overrun with hackers just ...	1	0