	Text Classification Using Transformer Networks (BERT) Some initialization:
In [1]:	<pre>from transformers import AutoTokenizer, AutoModel from huggingface_hub import snapshot_download # Define el nombre del modelo que quieres descargar model_name = "bert-base-cased"</pre>
	<pre># Descarga el modelo completo a una carpeta local local_folder = snapshot_download(repo_id=model_name) print(f"Modelo descargado en: {local_folder}") Fetching 10 files: 0% </pre>
	<pre>.gitattributes: 0% </pre>
In [8]:	vocab.txt: 0% 0.00/213k [00:00 , ?B/s] tokenizer_config.json: 0% 0.00/49.0 [00:00<?, ?B/s] Modelo descargado en: /Users/jacobohirsch/.cache/huggingface/hub/models—bert-base—cased/snapshots/cd5ef92a9fb2f889e972770a36d4ed042daf221e # Importamos las bibliotecas necesarias import random # Biblioteca para trabajar con números aleatorios. import torch # Framework de machine learning para cálculos en GPU y CPU.</td
	<pre>import numpy as np # Biblioteca para operaciones matemáticas y matrices. import pandas as pd # Biblioteca para manipulación de datos en estructuras como DataFrames. from tqdm.notebook import tqdm # Proporciona barras de progreso visuales en Jupyter Notebooks. # Habilitamos el uso de barras de progreso en las operaciones de pandas tqdm.pandas()</pre>
	# Variable para habilitar el uso de GPU, si está disponible use_gpu = True # Seleccionamos el dispositivo a usar: 'cuda' (GPU) si está habilitada y disponible, de lo contrario 'cpu' device = torch.device('cuda' if use_gpu and torch.cuda.is_available() else 'cpu') # Mostramos qué dispositivo se utilizará print(f'device: {device.type}')
	<pre># Definimos una semilla aleatoria para asegurar reproducibilidad seed = 1122 # Configuramos la semilla aleatoria si está definida (diferente de None) if seed is not None: print(f'random seed: {seed}') # Mostramos la semilla configurada.</pre>
	random.seed(seed) # Establecemos la semilla para el módulo random. np.random.seed(seed) # Establecemos la semilla para NumPy. torch.manual_seed(seed) # Establecemos la semilla para PyTorch. device: cpu random seed: 1122 Read the train/dev/test datasets and create a HuggingFace Dataset object:
In [9]:	<pre>def read_data(filename): # Lee un archivo CSV y lo carga en un DataFrame de pandas. df = pd.read_csv(filename, header=None) # Asigna nombres a las columnas: 'label', 'title', y 'description'. df.columns = ['label', 'title', 'description'] # Ajusta las etiquetas para que sean basadas en cero (resta 1 a todas las etiquetas).</pre>
	<pre>df['label'] -= 1 # Combina las columnas 'title' y 'description' en una nueva columna llamada 'text'. # Luego, elimina cualquier barra invertida ('\') en el texto. df['text'] = df['title'] + " " + df['description'] df['text'] = df['text'].str.replace('\\', ' ', regex=False) # Devuelve el DataFrame procesado. return df</pre>
<pre>In [10]: Out[10]:</pre>	<pre>labels = open('/datasets/ag_news_csv/classes.txt').read().splitlines() train_df = read_data('/datasets/ag_news_csv/train.csv') test_df = read_data('/datasets/ag_news_csv/test.csv') train_df</pre> label title description text
	 Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers, Wall Street's dwindli Wall St. Bears Claw Back Into the Black (Reute Carlyle Looks Toward Commercial Aerospace (Reu Reuters - Private investment firm Carlyle Grou Carlyle Looks Toward Commercial Aerospace (Reu Oil and Economy Cloud Stocks' Outlook (Reuters) Reuters - Soaring crude prices plus worries\ab Oil and Economy Cloud Stocks' Outlook (Reuters Iraq Halts Oil Exports from Main Southern Pipe
	4 2 Oil prices soar to all-time record, posing new AFP - Tearaway world oil prices, toppling reco Oil prices soar to all-time record, posing new
	119997 1 Saban not going to Dolphins yet The Miami Dolphins will put their courtship of Saban not going to Dolphins yet The Miami Dolp 119998 1 Today's NFL games PITTSBURGH at NY GIANTS Time: 1:30 p.m. Line: Today's NFL games PITTSBURGH at NY GIANTS Time 119999 1 Nets get Carter from Raptors INDIANAPOLIS All-Star Vince Carter was trad Nets get Carter from Raptors INDIANAPOLIS A
In [11]:	# Importamos la función train_test_split de scikit-learn para dividir el dataset en subconjuntos. from sklearn.model_selection import train_test_split # Dividimos el DataFrame de entrenamiento (train_df) en dos partes: # - train_df (90% de los datos originales) para entrenar el modelo.
	# - eval_df (10% de los datos originales) para validar el modelo durante el entrenamiento. train_df, eval_df = train_test_split(train_df, train_size=0.9) # Reseteamos los índices del DataFrame de entrenamiento después de la división. # Esto asegura que los índices comiencen desde 0 y elimina los índices anteriores. train_df.reset_index(inplace=True, drop=True) # Reseteamos los índices del DataFrame de evaluación (eval_df) por las mismas razones.
	<pre># Mostramos el número de filas en cada subconjunto para verificar la división. # len(train_df.index) cuenta las filas en el DataFrame de entrenamiento. print(f'train rows: {len(train_df.index):,}') # Cantidad de filas en el conjunto de entrenamiento. print(f'eval rows: {len(eval_df.index):,}') # Cantidad de filas en el conjunto de evaluación. print(f'test rows: {len(test_df.index):,}') # Cantidad de filas en el conjunto de prueba.</pre>
	train rows: 108,000 eval rows: 12,000 test rows: 7,600 # Importamos Dataset y DatasetDict de la librería datasets. # Dataset: representa un conjunto de datos tabular.
	# DatasetDict: estructura que organiza múltiples conjuntos de datos (e.g., train, validation, test). from datasets import Dataset, DatasetDict # Creamos un DatasetDict para almacenar nuestros conjuntos de datos procesados. ds = DatasetDict() # Convertimos el DataFrame de entrenamiento (train_df) en un objeto Dataset y lo asignamos a la clave 'train'.
	<pre>ds['train'] = Dataset.from_pandas(train_df) # Convertimos el DataFrame de validación (eval_df) en un objeto Dataset y lo asignamos a la clave 'validation'. ds['validation'] = Dataset.from_pandas(eval_df) # Convertimos el DataFrame de prueba (test_df) en un objeto Dataset y lo asignamos a la clave 'test'. ds['test'] = Dataset.from_pandas(test_df)</pre>
Out[12]:	<pre># Mostramos el DatasetDict para inspeccionar sus contenidos. ds DatasetDict({ train: Dataset({ features: ['label', 'title', 'description', 'text'], num_rows: 108000</pre>
	<pre> }) validation: Dataset({ features: ['label', 'title', 'description', 'text'], num_rows: 12000 }) test: Dataset({ features: ['label', 'title', 'description', 'text'], </pre>
Tn ^r	<pre>features: ['label', 'title', 'description', 'text'],</pre>
ın [13]:	# Importamos el AutoTokenizer desde la librería transformers. from transformers import AutoTokenizer # Definimos el nombre del modelo preentrenado que queremos usar. transformer_name = 'bert-base-cased' # Cargamos el tokenizador preentrenado correspondiente al modelo definido. tokenizer = AutoTokenizer.from_pretrained(transformer_name)
In [14]:	
	# Tokeniza y transforma el conjunto de datos de entrenamiento train_ds = ds['train'].map(tokenize, # Aplica la función de tokenización a cada ejemplo batched=True, # Procesa múltiples ejemplos al mismo tiempo para mayor eficiencia remove_columns=['title', 'description', 'text'], # Elimina estas columnas después de tokenizar)
	<pre># Tokeniza y transforma el conjunto de datos de validación eval_ds = ds['validation'].map(tokenize, # Aplica la función de tokenización batched=True, # Procesa en lotes para mayor eficiencia remove_columns=['title', 'description', 'text'], # Elimina las columnas originales)</pre>
Out[14]:	# Convierte el conjunto de datos de entrenamiento tokenizado a un DataFrame de pandas train_ds.to_pandas() Map: 0%
	 2 [101, 16752, 13335, 1186, 2101, 6690, 9717, 11 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
	4 3 [101, 142, 10044, 27302, 4317, 1584, 3273, 111 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
	107997 2 [101, 13832, 3484, 11300, 4060, 5058, 112, 188 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
In [15]:	# Importamos módulos necesarios de PyTorch y la biblioteca Hugging Face Transformers. from torch import nn # Submódulo para construir redes neuronales. from transformers.modeling_outputs import SequenceClassifierOutput # Clase que encapsula las salidas del modelo. from transformers.models.bert.modeling_bert import BertModel, BertPreTrainedModel # Modelos base de BERT.
	<pre># Clase para clasificación de secuencias basada en un modelo preentrenado de BERT. class BertForSequenceClassification(BertPreTrainedModel): # Inicialización de la clase definit(self, config): super()init(config) # Inicializa la clase base con la configuración del modelo. self.num_labels = config.num_labels # Número de etiquetas de clasificación.</pre>
	<pre>self.bert = BertModel(config) # Modelo base de BERT. self.dropout = nn.Dropout(config.hidden_dropout_prob) # Capa de dropout para regularización. self.classifier = nn.Linear(config.hidden_size, config.num_labels) # Capa totalmente conectada para clasificación. self.init_weights() # Inicializa los pesos del modelo. # Método que define el comportamiento del modelo en la fase de forward (inferencia/entrenamiento). def forward(self, input_ids=None, attention_mask=None, token_type_ids=None, tabels=None, **kwargs):</pre>
	<pre># Ejecuta el modelo BERT base. outputs = self.bert(input_ids, # IDs de los tokens de entrada. attention_mask=attention_mask, # Máscara de atención para indicar posiciones válidas en la secuencia. token_type_ids=token_type_ids, # IDs para diferenciar tipos de segmentos (opcional). **kwargs, # Argumentos adicionales (si los hay).)</pre>
	<pre># Extrae las representaciones del token [CLS] (primer token de cada secuencia). cls_outputs = outputs.last_hidden_state[:, 0, :] # Dimensión `[batch_size, hidden_size]`. # Aplica dropout para prevenir overfitting. cls_outputs = self.dropout(cls_outputs) # Calcula los logits (valores sin pormalizar para cada clase)</pre>
	# Calcula los logits (valores sin normalizar para cada clase). logits = self.classifier(cls_outputs) # Dimensión `[batch_size, num_labels]`. # Calcula la pérdida si las etiquetas están presentes (entrenamiento). loss = None if labels is not None: loss_fn = nn.CrossEntropyLoss() # Función de pérdida de entropía cruzada para clasificación.
	loss = loss_fn(logits, labels) # Calcula la pérdida entre los logits y las etiquetas. # Devuelve una estructura de salida con la pérdida, los logits, y otras representaciones. return SequenceClassifierOutput(loss=loss, # Pérdida calculada (None si no hay etiquetas). logits=logits, # Logits de salida para cada clase. hidden_states=outputs.hidden_states, # Representaciones ocultas de BERT (opcional). attentions=outputs.attentions, # Mapas de atención de BERT (opcional).
In [16]:	from transformers import AutoConfig # Crea una configuración para el modelo, cargándola desde un modelo preentrenado. config = AutoConfig.from_pretrained(transformer_name, # Nombre o ruta del modelo preentrenado (e.g., 'bert-base-uncased').
	num_labels=len(labels), # Especifica el número de etiquetas para la tarea de clasificación. # Inicializa el modelo de clasificación basado en BERT con la configuración definida. model = (BertForSequenceClassification .from_pretrained(transformer_name, config=config) # Carga pesos preentrenados y usa la configuración personalizada.
	Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-cased and are newly initialized: ['classifier.bias', 'classifier.weight'] You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference. Create the trainer object and train:
	<pre>import os os.environ["TOKENIZERS_PARALLELISM"] = "false" import accelerate print(f"Accelerate version: {accelerateversion}") Accelerate version: 1.1.1</pre>
	<pre>import accelerate import accelerate print(f"Accelerate version: {accelerateversion}") Accelerate version: 1.1.1</pre>
In [6]:	!pip install torch torchvision torchaudioindex-url https://download.pytorch.org/whl/cpu !pip install 'transformers[torch]' Defaulting to user installation because normal site-packages is not writeable Looking in indexes: https://download.pytorch.org/whl/cpu Requirement already satisfied: torch in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (2.5.1) Requirement already satisfied: torchvision in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (0.20.1)
	Requirement already satisfied: torchaudio in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (2.5.1) Requirement already satisfied: filelock in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torch) (3.12.2) Requirement already satisfied: typing-extensions>=4.8.0 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torch) (4.8.0) Requirement already satisfied: networkx in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torch) (3.1) Requirement already satisfied: fsspec in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from torch) (2024.6.1) Requirement already satisfied: sympy==1.13.1 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from torch) (1.13.1)
	Requirement already satisfied: mpmath<1.4,>=1.1.0 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from sympy==1.13.1->torch) (1.3.0) Requirement already satisfied: numpy in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from torchvision) (2.1.3) Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torchvision) (10.0.0) Requirement already satisfied: MarkupSafe>=2.0 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from jinja2->torch) (2.1.3) Defaulting to user installation because normal site-packages is not writeable Requirement already satisfied: transformers[torch] in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (4.46.3) Requirement already satisfied: filelock in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from transformers[torch]) (3.12.2)
	Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (0.26.2) Requirement already satisfied: numpy>=1.17 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (2.1.3) Requirement already satisfied: packaging>=20.0 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from transformers[torch]) (23.1) Requirement already satisfied: pyyaml>=5.1 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python/site-packages (from transformers[torch]) (6.0.1) Requirement already satisfied: regex!=2019.12.17 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (2.32.3) Requirement already satisfied: tokenizers<0.21,>=0.20 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (0.20.3)
	Requirement already satisfied: safetensors>=0.4.1 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (0.4.5) Requirement already satisfied: tqdm>=4.27 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (4.67.0) Requirement already satisfied: torch in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (2.5.1) Requirement already satisfied: accelerate>=0.26.0 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from transformers[torch]) (1.1.1) Requirement already satisfied: psutil in /Library/Frameworks/Python.framework/Versions/3.10/lib/python/site-packages (from huggingface-hub<1.0,>=0.23.2->transformers[torch]) (2024.6.1) Requirement already satisfied: typing-extensions>=3.7.4.3 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from huggingface-hub<1.0,>=0.23.2->transformers[torch]) (4.8.
	Requirement already satisfied: networkx in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torch->transformers[torch]) (3.1) Requirement already satisfied: jinja2 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from torch->transformers[torch]) (3.1.2) Requirement already satisfied: sympy==1.13.1 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from torch->transformers[torch]) (1.13.1) Requirement already satisfied: mpmath<1.4,>=1.1.0 in /Users/jacobohirsch/Library/Python/3.10/lib/python/site-packages (from sympy==1.13.1->torch->transformers[torch]) (1.3.0) Requirement already satisfied: charset-normalizer<4,>=2 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from requests->transformers[torch]) (3.2.0) Requirement already satisfied: urlibia<3,>=1.21.1 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from requests->transformers[torch]) (2.0.4)
In [18]:	Requirement already satisfied: certifi>=2017.4.17 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from requests->transformers[torch]) (2022.12.7) Requirement already satisfied: MarkupSafe>=2.0 in /Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages (from jinja2->torch->transformers[torch]) (2.1.3) from transformers import TrainingArguments # Parámetros para el entrenamiento num_epochs = 2 # Número de épocas (cuántas veces se verá todo el dataset durante el entrenamiento).
	<pre>batch_size = 24 # Tamaño del batch (número de muestras procesadas en cada paso). weight_decay = 0.01 # Coeficiente de regularización L2 para prevenir overfitting. model_name = f'{transformer_name}-sequence-classification' # Nombre del modelo con propósito de identificación. # Configuración del entrenamiento training_args = TrainingArguments(output_dir=model_name, # Directorio donde se guardarán los resultados y checkpoints.</pre>
	log_level='error', # Nivel de registro (para reducir el ruido en la salida). num_train_epochs=num_epochs, # Número de épocas definidas previamente. per_device_train_batch_size=batch_size, # Tamaño del batch por dispositivo (GPU o CPU). per_device_eval_batch_size=batch_size, # Tamaño del batch para evaluación (igualmente por dispositivo). evaluation_strategy='epoch', # Realiza evaluación al final de cada época. weight_decay=weight_decay, # Tasa de decaimiento de peso (regularización).)
In [19]:	<pre>from sklearn.metrics import accuracy_score # Función para calcular métricas de evaluación. def compute_metrics(eval_pred): # Extrae las etiquetas reales (ground truth) del objeto de predicciones. y_true = eval_pred.label_ids</pre>
	# Extrae las predicciones y obtiene las clases con mayor probabilidad (argmax). y_pred = np.argmax(eval_pred.predictions, axis=-1) # Calcula la precisión (accuracy) comparando las etiquetas reales y las predicciones. return {'accuracy': accuracy_score(y_true, y_pred)}
In [20]:	# Inicializa un objeto Trainer para gestionar el entrenamiento y evaluación del modelo. trainer = Trainer(model=model, # Modelo de clasificación (por ejemplo, basado en BERT). args=training_args, # Argumentos de entrenamiento configurados con `TrainingArguments`. compute_metrics=compute_metrics, # Función para calcular métricas de evaluación.
	train_dataset=train_ds, # Conjunto de datos de entrenamiento. eval_dataset=eval_ds, # Conjunto de datos de evaluación. tokenizer=tokenizer, # Tokenizador utilizado para procesar los datos. /var/folders/d4/4sl4p51s0h73_0dwmqbxcsr80000gn/T/ipykernel_65182/353304582.py:4: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainerinit`. Use `processing_class` instead.
In [21]:	trainer = Trainer(trainer.train() 0% 0/9000 [00:00 , ?it/s] {'loss': 0.3383, 'grad_norm': 5.228844165802002, 'learning_rate': 4.7222222222222222222222222222222222222</td
	{'loss': 0.2156, 'grad_norm': 1.524783968925476, 'learning_rate': 3.8888888888889e-05, 'epoch': 0.44} {'loss': 0.2137, 'grad_norm': 1.386098861694336, 'learning_rate': 3.61111111111111e-05, 'epoch': 0.56} {'loss': 0.198, 'grad_norm': 6.009778022766113, 'learning_rate': 3.33333333333335e-05, 'epoch': 0.67} {'loss': 0.2019, 'grad_norm': 5.193398475646973, 'learning_rate': 3.0555555555556e-05, 'epoch': 0.78} {'loss': 0.1915, 'grad_norm': 6.458729267120361, 'learning_rate': 2.7777777777778e-05, 'epoch': 0.89} {'loss': 0.1805, 'grad_norm': 2.4253339767456055, 'learning_rate': 2.5e-05, 'epoch': 1.0} 0% 0/500 [00:00 , ?it/s]</td
	{'eval_loss': 0.18264496326446533, 'eval_accuracy': 0.942583333333333, 'eval_runtime': 39.9468, 'eval_samples_per_second': 300.399, 'eval_steps_per_second': 12.517, 'epoch': 1.0} {'loss': 0.1326, 'grad_norm': 0.41072121262550354, 'learning_rate': 2.22222222222222222222222222222222222
Out[21 ¹ ·	{'loss': 0.1132, 'grad_norm': 4.58664608001709, 'learning_rate': 5.55555555555555555555555555555555555
	<pre>e+16, 'train_loss': 0.17396008724636502, 'epoch': 2.0}) Evaluate on the test partition: test_ds = ds['test'].map(tokenize, batched=True,</pre>
Out[22]:	remove_columns=['title', 'description', 'text'],) test_ds.to_pandas() Map: 0%
	 2 [101, 11284, 1116, 1111, 157, 151, 12966, 1170 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
	4 3 [101, 11917, 8914, 119, 19294, 4206, 1106, 215 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
	7597 1 [101, 16247, 2972, 9178, 2409, 4271, 140, 1418 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
<pre>In [23]: Out[23]:</pre>	output 0% 0/317 [00:00 , ?it/s]</td
	- , , , , , , , , , , , , , , , , , , ,
In [25]:	<pre>from sklearn.metrics import classification_report y_true = output.label_ids y_pred = np.argmax(output.predictions, axis=-1) target_names = labels print(classification_report(y_true, y_pred, target_names=target_names))</pre>
	precision recall f1-score support World 0.96 0.96 0.96 1900 Sports 0.99 0.99 0.99 1900 Business 0.93 0.91 0.92 1900 Sci/Tech 0.91 0.94 0.93 1900
	accuracy
	El pipeline con BERT para clasificación de texto sigue un flujo claro y bien organizado que abarca desde la preparación de los datos hasta la evaluación final del modelo. Primero, configura el entorno para asegurar que se utilice la GPU (si está disponible) y garantiza la reproducibilidad estableciendo una semilla aleatoria. Luego, los datos son procesados: se cargan desde archivos CSV, se limpian y se transforman en un formato compatible con modelos de aprendizaje profundo. Esto incluye tareas como tokenización, donde se convierte el texto en números que el modelo puede interpretar, y la organización en conjuntos de entrenamiento, validación y prueba. El modelo utilizado es BERT, adaptado específicamente para tareas de clasificación de texto. A través de AutoConfig y BertForSequenceClassification, se personaliza el modelo para manejar el número correcto de etiquetas. El entrenamiento del modelo se realiza utilizando la clase Trainer, que automatiza tareas como dividir los datos en lotes, calcular pérdidas, optimizar los parámetros y evaluar el modelo después de cada época. Además, se incluyen técnicas como la regularización

(weight decay) para evitar que el modelo se sobreajuste a los datos de entrenamiento.

!jupyter nbconvert --to html chap13_classification_bert.ipynb

[NbConvertApp] Converting notebook chap13_classification_bert.ipynb to html

[NbConvertApp] Writing 324884 bytes to chap13_classification_bert.html

In [28]: import os
 os.environ["TOKENIZERS_PARALLELISM"] = "false"

In [30]: import os
 os.environ["TOKENIZERS_PARALLELISM"] = "false"

Al finalizar el entrenamiento, el modelo se prueba en datos no vistos. Este proceso utiliza métricas como precisión, recall y F1-score para evaluar su rendimiento. Herramientas como classification_report ayudan a identificar las fortalezas y

debilidades del modelo en cada categoría. En general, este pipeline está diseñado para ser eficiente y robusto, maximizando el uso de modelos preentrenados como BERT y simplificando el trabajo gracias a las herramientas de Hugging Face.