Luis Maximiliano López Ramírez - A00833321

Pipeline de Clasificación de Texto Usando Redes Transformer (BERT)

Este pipeline implementa un proceso de clasificación de texto utilizando un modelo BERT. A continuación se describe cada etapa en detalle.

1. Configuración inicial y preparación del entorno

- **Bibliotecas y dispositivos**: Se importan bibliotecas como `torch`, `pandas`, `tqdm`, y otras necesarias para el procesamiento de texto y el entrenamiento de modelos. Se configura el dispositivo de ejecución (`GPU` si está disponible, o `CPU` en su defecto) y se fija una semilla para garantizar la reproducibilidad de los resultados.

2. Carga y procesamiento de datos

- **Función `read_data`**: Carga los datos de un archivo CSV y asigna nombres a las columnas (`label`, `title`, `description`). Luego:
  - Los labels se ajustan para que empiecen desde cero.
  - Se combina el título y la descripción en una columna de texto única (`text`), eliminando los caracteres de barra invertida (`\`).
- **División del conjunto de datos**:
  - Se cargan los datos de entrenamiento y prueba desde archivos separados.
  - El conjunto de datos de entrenamiento se divide en una partición de entrenamiento (`train_df`) y una partición de validación (`eval_df`) usando `train_test_split` de `scikit-learn`, con un 90% de los datos en el conjunto de entrenamiento.
- **Creación de objetos `Dataset`**:
  - Se usa la biblioteca `datasets` de Hugging Face para convertir los DataFrames en un objeto `DatasetDict` con tres particiones: `train`, `validation`, y `test`.

3. Tokenización del texto

- **Inicialización del tokenizador**:

    - Se carga el tokenizador correspondiente al modelo BERT seleccionado (`bert-base-cased`) desde Hugging Face.
- **Función `tokenize`**:
    - La función toma el texto de entrada (`text`) y lo convierte en tokens compatibles con el modelo BERT, truncando los textos largos que excedan la longitud máxima permitida.
- **Aplicación de la tokenización**:
    - Se tokenizan las particiones `train` , `validation` , y `test` usando la función `tokenize` con la opción `batched=True` para tokenizar en lotes.
    - Se eliminan las columnas originales (`title` , `description` , `text`) después de la tokenización para reducir el uso de memoria.

4. Definición del modelo de clasificación

- **Subclase `BertForSequenceClassification`**:
    - Se define una clase personalizada de modelo llamada `BertForSequenceClassification` , que hereda de `BertPreTrainedModel` .
    - El modelo se compone de:
        - Un modelo base BERT para obtener representaciones.
        - Un clasificador (`Linear`) que toma la representación de la primera posición del último estado oculto (`cls_outputs`) y genera las predicciones de clase.
    - En el método `forward` , se calcula la pérdida de entropía cruzada si se proporcionan labels durante el entrenamiento.

5. Configuración del modelo y argumentos de entrenamiento

- **Configuración del modelo**:
    - Se cargan los parámetros de configuración (`config`) específicos de `bert-base-cased` y se especifica el número de clases para el modelo de clasificación.
- **Definición de `TrainingArguments`**:
    - Se establecen los argumentos de entrenamiento (`TrainingArguments`), como el número de épocas, tamaño de batch, estrategia de evaluación, y tasa de decaimiento del peso (`weight_decay`).
    - Estos argumentos permiten controlar aspectos importantes del entrenamiento, como la frecuencia de evaluación y el número de épocas.

6. Definición de métricas de evaluación

- **Función `compute_metrics`**:
  - Para evaluar el rendimiento del modelo, se define una función que calcula la precisión (`accuracy_score`), comparando las etiquetas reales (`y_true`) con las predicciones (`y_pred`).

7. Entrenamiento del modelo

- **Creación del objeto `Trainer`**:
  - Se usa la clase `Trainer` de Hugging Face, que facilita el proceso de entrenamiento. Los parámetros principales del `Trainer` incluyen:
    - `model` : el modelo BERT personalizado.
    - `args` : los argumentos de entrenamiento definidos en `TrainingArguments` .
    - `compute_metrics` : la función para calcular métricas.
    - `train_dataset` y `eval_dataset` : los conjuntos de datos de entrenamiento y validación.
    - `tokenizer` : el tokenizador BERT.
- **Entrenamiento**:
  - Se ejecuta el entrenamiento con `trainer.train()` , que entrenará el modelo usando los parámetros especificados y evaluará en el conjunto de validación en cada época.

8. Evaluación en el conjunto de prueba

- **Tokenización del conjunto de prueba**:
  - Se tokeniza el conjunto de prueba (`test_ds`) con la misma configuración de tokenización aplicada a los otros conjuntos de datos.
- **Predicciones en el conjunto de prueba**:
  - Se generan predicciones en el conjunto de prueba con `trainer.predict(test_ds)` .

9. Reporte de métricas de clasificación

- **Reporte de clasificación**:
  - Utilizando `classification_report` de `scikit-learn` , se calcula y muestra un reporte detallado de las métricas de clasificación, que incluye precisión, recall y F1-score para cada clase.

Resumen del pipeline:

1. Configuración del entorno.
2. Carga, procesamiento y tokenización de los datos.
3. Definición y configuración del modelo BERT.
4. Configuración del entrenamiento.
5. Entrenamiento del modelo.
6. Evaluación en el conjunto de prueba.
7. Generación de métricas de clasificación para evaluar el rendimiento del modelo.

Este pipeline permite entrenar y evaluar un modelo de clasificación de texto de manera eficiente usando redes Transformer.

Comparación de Resultados: BERT vs. Regresión Logística con GloVe

A continuación se comparan los resultados obtenidos con el modelo BERT (implementación en PyTorch) y los resultados previos usando un clasificador de regresión logística con embeddings GloVe.

Desempeño del modelo BERT (implementación en PyTorch)

- **Precisión general (accuracy)**: 0.95
- **Macro promedio (f1-score)**: 0.95
- **Rendimiento por clase**:
  - **World**: Precisión 0.96, Recall 0.95, F1-score 0.96
  - **Sports**: Precisión 0.99, Recall 0.99, F1-score 0.99
  - **Business**: Precisión 0.92, Recall 0.91, F1-score 0.92
  - **Sci/Tech**: Precisión 0.92, Recall 0.93, F1-score 0.93

Desempeño del clasificador de regresión logística con GloVe

- **Precisión general (accuracy)**: 0.90
- **Macro promedio (f1-score)**: 0.90
- **Rendimiento por clase**:
  - **World**: Precisión 0.92, Recall 0.88, F1-score 0.90
  - **Sports**: Precisión 0.95, Recall 0.97, F1-score 0.96
  - **Business**: Precisión 0.85, Recall 0.86, F1-score 0.85
  - **Sci/Tech**: Precisión 0.86, Recall 0.87, F1-score 0.87

Comparación y Conclusiones

- El modelo **BERT** supera al modelo de regresión logística con GloVe en todas las métricas y para cada clase.

- En particular, BERT muestra mejoras en las clases **World**, **Business** y **Sci/Tech**, donde su capacidad de capturar patrones complejos le permite superar las limitaciones del modelo basado en GloVe.
- **Macro y weighted promedio de F1-score**: BERT alcanza 0.95 frente a 0.90 con GloVe, indicando un mejor desempeño general en la clasificación de todas las clases.

Text Classification Using Transformer Networks (BERT)

Some initialization:

In [1]:
```
import random
import torch
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm

# enable tqdm in pandas
tqdm.pandas()

# set to True to use the gpu (if there is one available)
use_gpu = True

# select device
device = torch.device('cuda' if use_gpu and torch.cuda.is_available() else 'cpu')
print(f'device: {device.type}')

# random seed
seed = 1122

# set random seed
if seed is not None:
    print(f'random seed: {seed}')
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
```

```
device: cuda
random seed: 1122
```

Read the train/dev/test datasets and create a HuggingFace `Dataset` object:

In [2]:
```
def read_data(filename):
    # read csv file
    df = pd.read_csv(filename, header=None)
    # add column names
    df.columns = ['label', 'title', 'description']
    # make labels zero-based
    df['label'] -= 1
    # concatenate title and description, and remove backslashes
    df['text'] = df['title'] + " " + df['description']
```

```
    df['text'] = df['text'].str.replace('\\', ' ', regex=False)
    return df
```

In [3]:
```
labels = open('classes.txt').read().splitlines()
train_df = read_data('train (1).csv')
test_df = read_data('test.csv')
train_df
```

Out[3]:

	label	title	description	text
0	2	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...
1	2	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...	Carlyle Looks Toward Commercial Aerospace (Reu...
2	2	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...	Oil and Economy Cloud Stocks' Outlook (Reuters...
3	2	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...	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...
...	...	...	...	...
119995	0	Pakistan's Musharraf Says Won't Quit as Army C...	KARACHI (Reuters) - Pakistani President Perve...	Pakistan's Musharraf Says Won't Quit as Army C...
119996	1	Renteria signing a top-shelf deal	Red Sox general manager Theo Epstein acknowled...	Renteria signing a top-shelf deal Red Sox gene...
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...

120000 rows × 4 columns

In [4]:
```
from sklearn.model_selection import train_test_split

train_df, eval_df = train_test_split(train_df, train_size=0.9)
train_df.reset_index(inplace=True, drop=True)
```

```
eval_df.reset_index(inplace=True, drop=True)

print(f'train rows: {len(train_df.index):,}')
print(f'eval rows: {len(eval_df.index):,}')
print(f'test rows: {len(test_df.index):,}')
```

```
train rows: 108,000
eval rows: 12,000
test rows: 7,600
```

In [5]:
```
from datasets import Dataset, DatasetDict

ds = DatasetDict()
ds['train'] = Dataset.from_pandas(train_df)
ds['validation'] = Dataset.from_pandas(eval_df)
ds['test'] = Dataset.from_pandas(test_df)
ds
```

Out[5]:
```
DatasetDict({
    train: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 108000
    })
    validation: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 12000
    })
    test: Dataset({
        features: ['label', 'title', 'description', 'text'],
        num_rows: 7600
    })
})
```

Tokenize the texts:

In [6]:
```
from transformers import AutoTokenizer

transformer_name = 'bert-base-cased'
tokenizer = AutoTokenizer.from_pretrained(transformer_name)
```

In [7]:
```
def tokenize(examples):
    return tokenizer(examples['text'], truncation=True)

train_ds = ds['train'].map(
    tokenize, batched=True,
    remove_columns=['title', 'description', 'text'],
)
eval_ds = ds['validation'].map(
    tokenize,
    batched=True,
    remove_columns=['title', 'description', 'text'],
)
train_ds.to_pandas()
```

```
Map:   0%|          | 0/108000 [00:00<?, ? examples/s]
Map:   0%|          | 0/12000 [00:00<?, ? examples/s]
```

Out[7]:

	label	input_ids	token_type_ids	attention_mask
0	2	[101, 16752, 13335, 1186, 2101, 6690, 9717, 11...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1	1	[101, 145, 11680, 17308, 9741, 2428, 150, 1469...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2	2	[101, 1418, 14099, 27086, 1494, 1114, 4031, 11...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	1	[101, 2404, 117, 6734, 1996, 118, 1565, 5465, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	3	[101, 142, 10044, 27302, 4317, 1584, 3273, 111...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
...	...	...	...	...
107995	1	[101, 4922, 2274, 1654, 1112, 10503, 1505, 112...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107996	3	[101, 10605, 24632, 11252, 21285, 10221, 118, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107997	2	[101, 13832, 3484, 11300, 4060, 5058, 112, 188...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107998	3	[101, 142, 13675, 3756, 5795, 2445, 1104, 109,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
107999	2	[101, 157, 16450, 1658, 5302, 185, 7776, 11006...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

108000 rows × 4 columns

Create the transformer model:

In [8]:
```
from torch import nn
from transformers.modeling_outputs import SequenceClassifierOutput
from transformers.models.bert.modeling_bert import BertModel, BertPreTrainedModel

# https://github.com/huggingface/transformers/blob/65659a29cf5a079842e61a63d57fa244

class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        self.init_weights()

    def forward(self, input_ids=None, attention_mask=None, token_type_ids=None, lab
        outputs = self.bert(
            input_ids,
```

```
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            **kwargs,
        )
        cls_outputs = outputs.last_hidden_state[:, 0, :]
        cls_outputs = self.dropout(cls_outputs)
        logits = self.classifier(cls_outputs)
        loss = None
        if labels is not None:
            loss_fn = nn.CrossEntropyLoss()
            loss = loss_fn(logits, labels)
        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

In [9]:
```
from transformers import AutoConfig

config = AutoConfig.from_pretrained(
    transformer_name,
    num_labels=len(labels),
)

model = (
    BertForSequenceClassification
    .from_pretrained(transformer_name, config=config)
)
```

```
Some weights of BertForSequenceClassification were not initialized from the model ch
eckpoint at bert-base-cased and are newly initialized: ['classifier.bias', 'classifi
er.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:

In [10]:
```
from transformers import TrainingArguments

num_epochs = 2
batch_size = 8  # Reducir el batch size para disminuir el consumo de memoria
weight_decay = 0.01
model_name = f'{transformer_name}-sequence-classification'

training_args = TrainingArguments(
    output_dir=model_name,
    log_level='error',
    num_train_epochs=num_epochs,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    evaluation_strategy='epoch',
    weight_decay=weight_decay,
    fp16=True,  # Habilitar entrenamiento en media precisión si la GPU lo permite
```

```
    gradient_accumulation_steps=4,  # Acumular gradientes para mantener un tamaño d
)
```

```
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\transformers\training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
WARNING:tensorflow:From c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [11]:
```
from sklearn.metrics import accuracy_score

def compute_metrics(eval_pred):
    y_true = eval_pred.label_ids
    y_pred = np.argmax(eval_pred.predictions, axis=-1)
    return {'accuracy': accuracy_score(y_true, y_pred)}
```

In [12]:
```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_ds,
    eval_dataset=eval_ds,
    tokenizer=tokenizer,
)
```

```
C:\Users\luism\AppData\Local\Temp\ipykernel_18616\485909320.py:3: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

In [13]:
```
trainer.train()
```

```
  0%|          | 0/6750 [00:00<?, ?it/s]
c:\Users\luism\Escritorio\Documetos_2\Entornos Virtuales\RetoConcentracion\lib\site-packages\transformers\models\bert\modeling_bert.py:440: UserWarning: 1Torch was not compiled with flash attention. (Triggered internally at C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\aten\src\ATen\native\transformers\cuda\sdp_utils.cpp:555.)
  attn_output = torch.nn.functional.scaled_dot_product_attention(
```

```
{'loss': 0.3066, 'grad_norm': 11.77697467803955, 'learning_rate': 4.631851851851852e
-05, 'epoch': 0.15}
{'loss': 0.2371, 'grad_norm': 32.30866241455078, 'learning_rate': 4.261481481481482e
-05, 'epoch': 0.3}
{'loss': 0.2142, 'grad_norm': 6.398453235626221, 'learning_rate': 3.8911111111111117
e-05, 'epoch': 0.44}
{'loss': 0.1967, 'grad_norm': 15.257124900817871, 'learning_rate': 3.520740740740741
e-05, 'epoch': 0.59}
{'loss': 0.1948, 'grad_norm': 9.700437545776367, 'learning_rate': 3.1503703703703707
e-05, 'epoch': 0.74}
{'loss': 0.1859, 'grad_norm': 17.58692169189453, 'learning_rate': 2.7800000000000005
e-05, 'epoch': 0.89}
  0%|          | 0/1500 [00:00<?, ?it/s]
{'eval_loss': 0.17243844270706177, 'eval_accuracy': 0.9428333333333333, 'eval_runtim
e': 22.3133, 'eval_samples_per_second': 537.797, 'eval_steps_per_second': 67.225, 'e
poch': 1.0}
{'loss': 0.1671, 'grad_norm': inf, 'learning_rate': 2.4111111111111113e-05, 'epoch':
1.04}
{'loss': 0.1216, 'grad_norm': 11.598967552185059, 'learning_rate': 2.040740740740740
8e-05, 'epoch': 1.19}
{'loss': 0.1195, 'grad_norm': 9.57097339630127, 'learning_rate': 1.6703703703703706e
-05, 'epoch': 1.33}
{'loss': 0.1169, 'grad_norm': 1.7359424829483032, 'learning_rate': 1.3000000000000000
1e-05, 'epoch': 1.48}
{'loss': 0.1166, 'grad_norm': 12.736505508422852, 'learning_rate': 9.296296296296298
e-06, 'epoch': 1.63}
{'loss': 0.1075, 'grad_norm': 21.27420997619629, 'learning_rate': 5.592592592592593e
-06, 'epoch': 1.78}
{'loss': 0.1139, 'grad_norm': 1.9818521738052368, 'learning_rate': 1.888888888888889
e-06, 'epoch': 1.93}
  0%|          | 0/1500 [00:00<?, ?it/s]
{'eval_loss': 0.18550314009189606, 'eval_accuracy': 0.94625, 'eval_runtime': 23.430
8, 'eval_samples_per_second': 512.146, 'eval_steps_per_second': 64.018, 'epoch': 2.
0}
{'train_runtime': 1361.4554, 'train_samples_per_second': 158.654, 'train_steps_per_s
econd': 4.958, 'train_loss': 0.16710691155327692, 'epoch': 2.0}
```

Out[13]:
```
TrainOutput(global_step=6750, training_loss=0.16710691155327692, metrics={'train_r
untime': 1361.4554, 'train_samples_per_second': 158.654, 'train_steps_per_second':
4.958, 'total_flos': 1.0013886474017088e+16, 'train_loss': 0.16710691155327692, 'e
poch': 2.0})
```

Evaluate on the test partition:

In [14]:
```
test_ds = ds['test'].map(
    tokenize,
    batched=True,
    remove_columns=['title', 'description', 'text'],
)
test_ds.to_pandas()
```

```
Map:   0%|          | 0/7600 [00:00<?, ? examples/s]
```

Out[14]:

	label	input_ids	token_type_ids	attention_mask
0	2	[101, 11284, 1116, 1111, 157, 151, 12966, 1170...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
1	3	[101, 1109, 6398, 1110, 1212, 131, 2307, 7219,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
2	3	[101, 148, 1183, 119, 1881, 16387, 1116, 4468,...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
3	3	[101, 11689, 15906, 6115, 12056, 1116, 1370, 2...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
4	3	[101, 11917, 8914, 119, 19294, 4206, 1106, 215...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
...	...	...	...	...
7595	0	[101, 5596, 1103, 1362, 5284, 5200, 3234, 1384...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7596	1	[101, 159, 7874, 1110, 2709, 1114, 13875, 1556...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7597	1	[101, 16247, 2972, 9178, 2409, 4271, 140, 1418...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7598	2	[101, 126, 1104, 1893, 8167, 10721, 4420, 1107...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
7599	2	[101, 142, 2064, 4164, 3370, 1154, 13519, 1116...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

7600 rows × 4 columns

In [15]:
```
output = trainer.predict(test_ds)
output
```

```
 0%|          | 0/950 [00:00<?, ?it/s]
```

Out[15]:
```
PredictionOutput(predictions=array([[ 0.61035156, -4.6054688 ,  5.5507812 , -2.326
1719 ],
       [-0.02011108, -3.8671875 , -3.0292969 ,  6.1914062 ],
       [ 0.08660889, -3.1953125 , -3.578125  ,  5.625     ],
       ...,
       [-1.3798828 ,  7.3203125 , -2.2226562 , -3.5996094 ],
       [-2.0175781 , -4.0273438 ,  6.1796875 , -1.7382812 ],
       [-2.3398438 , -4.359375  ,  3.46875   ,  1.9785156 ]],
      dtype=float32), label_ids=array([2, 3, 3, ..., 1, 2, 2], dtype=int64), metri
cs={'test_loss': 0.19400855898857117, 'test_accuracy': 0.9477631578947369, 'test_r
untime': 12.1629, 'test_samples_per_second': 624.849, 'test_steps_per_second': 78.
106})
```

In [18]:
```
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))
```

```
              precision    recall  f1-score   support

       World       0.96      0.95      0.96      1900
      Sports       0.99      0.99      0.99      1900
    Business       0.92      0.91      0.92      1900
    Sci/Tech       0.92      0.93      0.93      1900

    accuracy                           0.95      7600
   macro avg       0.95      0.95      0.95      7600
weighted avg       0.95      0.95      0.95      7600
```