Text Classification Using Transformer Networks (BERT)

Para empezar el código, como primer paso se establecen las configuraciones iniciales del entorno, incluyendo la carga de bibliotecas necesarias como torch, numpy y pandas.

Se selecciona el dispositivo de cómputo según la disponibilidad de GPU o CPU y se define una semilla aleatoria en este caso 1122, esto para garantizar reproducibilidad, esto asegura un entorno controlado para el entrenamiento y evaluación del modelo.

Some initialization:

```
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
```

Como siguiente paso realizamos la carga y procesamiento de los datos. Los datos se leen desde archivos CSV que contienen las noticias categorizadas, en donde cada registro incluye una etiqueta, un título y una descripción, estos se combinan en una nueva columna llamada text. Además, las etiquetas se ajustan para que comiencen en 0. Y también, el conjunto de datos se divide en entrenamiento, validación y prueba. Esto proceso se realiza de las celdas 2 a la 4

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

```
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


labels = open('/kaggle/input/ag-news/ag_news_csv/classes.txt').read().splitlines()
train_df = read_data('/kaggle/input/ag-news/ag_news_csv/train.csv')
test_df = read_data('/kaggle/input/ag-news/ag_news_csv/test.csv')
train_df
```

	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...

```
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
```

En la celda 5 realizamos una conversión a formato de dataset. Los dataframes procesados se convierten en un objeto DatasetDict compatible con la biblioteca Hugging Face. Esto permite gestionar los conjuntos de datos de manera eficiente y facilita su integración con las etapas posteriores.

```
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
```

```
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
    })
})
```

En las siguientes 2 celdas realizamos la tokenización. Se utiliza el tokenizador preentrenado bert-base-cased para transformar el texto en representaciones numéricas comprensibles para el modelo. Este proceso incluye la truncación de texto largo y la eliminación de las columnas title, description y text ya no son necesarias.

Tokenize the texts:

```
from transformers import AutoTokenizer

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

```
tokenizer_config.json:   0%|          | 0.00/49.0 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
/opt/conda/lib/python3.10/site-packages/transformers/tokenization_utils_base.py:1617: FutureWarning
  warnings.warn(
```

```
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]
```

	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, ...

En los siguientes pasos definimos nuestro modelo. Se implementa una clase personalizada que extiende el modelo BERT para realizar tareas de clasificación de secuencias. Esto incluye agregar capas adicionales como Dropout y una capa lineal para mapear las salidas de BERT a las etiquetas de clasificación. El modelo se inicializa con una configuración específica que define el número de categorías en el conjunto de datos.

Create the transformer model:

```
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/65659a29cf5a079842e61a63d57fa24474288998/src/transf

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, labels=None, **kwargs)
        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,
        )


from transformers import AutoConfig

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

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

```
model.safetensors:   0%|          | 0.00/436M [00:00<?, ?B/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
```

Una vez definido nuestro modelo realizamos la configuración del entrenamiento. Se configuran los parámetros clave para entrenar el modelo, como el número de épocas, el tamaño de lote y la estrategia de evaluación (por época). También se define una función para calcular métricas de evaluación, como la precisión. Con estos elementos, se inicializa un objeto Trainer que gestiona todo el proceso de entrenamiento y evaluación.

Create the trainer object and train:

```
from transformers import TrainingArguments

num_epochs = 2
batch_size = 24
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,
)
```

```
/opt/conda/lib/python3.10/site-packages/transformers/training_args.py:1545: FutureWarning: `evalua
  warnings.warn(
```

```
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)}


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,
)
```

Una vez configuadas las especificaciones del entrenamiento se entrena el modelo utilizando los datos procesados. Se desactiva el servicio de registro automático de Weights & Biases (W&B) para simplificar la ejecución y evitar interacciones no deseadas.

```
import os
os.environ["WANDB_DISABLED"] = "true"

trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more i
```

Tracking run with wandb version 0.18.3

W&B syncing is set to `offline` in this directory.

```
Run `wandb online` or set WANDB_MODE=online to enable cloud syncing.
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
```

[4500/4500 52:04, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.187400	0.173580	0.940167
2	0.102200	0.162218	0.945750

```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
  with torch.cuda.device(device), torch.cuda.stream(stream), autocast(enabled=autocast_enabled):
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked
  warnings.warn('Was asked to gather along dimension 0, but all '
TrainOutput(global_step=4500, training_loss=0.16200994957817924, metrics={'train_runtime':
```

Y finalmente realizamos la predicción y evaluación del modelo. El conjunto de prueba se procesa utilizando el mismo esquema de tokenización que los conjuntos de entrenamiento y validación. El modelo genera predicciones sobre estos datos, que se evalúan con métricas detalladas como precisión, recall y F1-score mediante el reporte de clasificación de sklearn. Esto proporciona una visión clara del rendimiento del modelo en datos nuevos.

Evaluate on the test partition:

```
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]
```

	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, ...

```
output = trainer.predict(test_ds)
output
```

```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/parallel_apply.py:79: FutureWarning: `to
```

```
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       0.97      0.96      0.96      1900
      Sports       0.99      0.99      0.99      1900
    Business       0.92      0.91      0.92      1900
    Sci/Tech       0.91      0.93      0.92      1900

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

Conclusión

Usar una implementación basada en PyTorch de BERT para la tarea de clasificación de texto ofrece un desempeño superior en comparación con enfoques tradicionales como regresión logística con embeddings GloVe. Esto se debe a que BERT es un modelo preentrenado basado en transformers, diseñado para capturar el contexto bidireccional del lenguaje, lo que permite comprender mejor las relaciones complejas y dependencias en los datos textuales. A diferencia de GloVe, que genera representaciones estáticas de palabras, BERT produce embeddings contextuales, adaptándose al significado específico de las palabras según su contexto en la oración. Además, su arquitectura profunda y capacidad de ajuste permiten optimizar el modelo para tareas específicas, maximizando la precisión en la clasificación. Por estas razones, BERT supera en desempeño a métodos más simples, especialmente en conjuntos de datos complejos como el de noticias categorizadas.