```
from datasets import load_dataset
from transformers import BertTokenizerFast, BertForSequenceClassification, Trainer, TrainingArguments
import torch
import numpy as np
import joblib
from sklearn.metrics import accuracy_score, f1_score, classification_report

MODEL_NAME = "bert-base-uncased"
NUM_LABELS = 4
MODEL_DIR = "bert_news_model"
DUMP_PATH = "news_classifier_metadata.joblib"
RANDOM_SEED = 42


# 1. Load dataset
print("Loading AG News dataset...")
dataset = load_dataset("ag_news")
```

```
Loading AG News dataset...
```

```
# 2. Tokenizer
tokenizer = BertTokenizerFast.from_pretrained(MODEL_NAME)

def tokenize_batch(batch):
    return tokenizer(batch["text"], truncation=True, padding="max_length", max_length=128)

dataset = dataset.map(tokenize_batch, batched=True)
```

```
Map: 100%                                                  7600/7600 [00:01<00:00, 4841.95 examples/s]
```

```
# 3. Set format for PyTorch
dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])

train_dataset = dataset["train"]
test_dataset = dataset["test"]

# 4. Model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
model = BertForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=NUM_LABELS)
model.to(device)
```

```
Using device: cuda
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initiali
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
          )
        )
        (pooler): BertPooler(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (activation): Tanh()
        )
      )
      (dropout): Dropout(p=0.1, inplace=False)
      (classifier): Linear(in_features=768, out_features=4, bias=True)
    )
```

```
# 5. Metrics function
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    acc = accuracy_score(labels, preds)
    f1 = f1_score(labels, preds, average="macro")
    return {"accuracy": acc, "f1_macro": f1}
```

```
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    weight_decay=0.01,
    load_best_model_at_end=True,
    seed=RANDOM_SEED,
    logging_dir="./logs",
    logging_steps=50,
    dataloader_num_workers=4,
    fp16=True
)
```

```
# 7. Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    processing_class=tokenizer,
    compute_metrics=compute_metrics
)
```

```
# 8. Train
trainer.train()
```

```
list-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes

i. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
I key in your browser here: https://wandb.ai/authorize?ref=models
m your profile and hit enter:wandb: WARNING If you're specifying your api key in code, ensure this code is no
ting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
 creating one.
i.wandb.ai to your netrc file: /root/.netrc
as: moizpirzada11 (moizpirzada11-szabist) to https://api.wandb.ai. Use `wandb login --relogin` to force relog
0.21.0
tent/wandb/run-20250812_233742-b8i480rv
) Weights & Biases (docs)
oizpirzada11-szabist/huggingface
pirzada11-szabist/huggingface/runs/b8i480rv
list-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes

[15001/22500 32:06 < 16:03, 7.78 it/s, Epoch 2/3]

dation Loss  Accuracy  F1 Macro
    0.177929  0.943289  0.943271
    0.191554  0.946316  0.946413

list-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes

list-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes

[22500/22500 48:50, Epoch 3/3]

dation Loss  Accuracy  F1 Macro
    0.177929  0.943289  0.943271
    0.191554  0.946316  0.946413
    0.232124  0.946842  0.946889

00, training_loss=0.15091253214412265, metrics={'train_runtime': 3022.4575, 'train_samples_per_second':
econd': 7.444, 'total_flos': 2.368042020864e+16, 'train_loss': 0.15091253214412265, 'epoch': 3.0})
```

```
# 9. Evaluate
print("\n=== Final Evaluation on Test Set ===")
preds_output = trainer.predict(test_dataset)
y_true = preds_output.label_ids
y_pred = np.argmax(preds_output.predictions, axis=-1)
print("Accuracy:", accuracy_score(y_true, y_pred))
print("F1-macro:", f1_score(y_true, y_pred, average="macro"))
print(classification_report(y_true, y_pred, digits=4))
```

```
st Set ===
dist-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes

06
32
  recall  f1-score   support

  0.9511    0.9543      1900
  0.9905    0.9879      1900
  0.9089    0.9140      1900
  0.9226    0.9168      1900

            0.9433      7600
  0.9433    0.9433      7600
  0.9433    0.9433      7600
```

```
# 10. Save full model + tokenizer
model.save_pretrained(MODEL_DIR)
tokenizer.save_pretrained(MODEL_DIR)
print(f"Model + tokenizer saved to {MODEL_DIR}")
```

```
Model + tokenizer saved to bert_news_model
```

```
# 11. Save metadata with joblib
metadata = {
```

```
    "model_dir": MODEL_DIR,
    "label_map": {0: "World", 1: "Sports", 2: "Business", 3: "Sci/Tech"},
    "base_model": MODEL_NAME
}
joblib.dump(metadata, DUMP_PATH)
print(f"Metadata saved to {DUMP_PATH}")
```

```
Metadata saved to news_classifier_metadata.joblib
```