In [8]:
```
import evaluate
import transformers
import numpy as np
import pandas as pd
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DistilBertForSequenceClassification,
)
```

In [9]:
```
"""
Load dataset.
"""
dataset = load_dataset("yelp_review_full")
print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['label', 'text'],
        num_rows: 650000
    })
    test: Dataset({
        features: ['label', 'text'],
        num_rows: 50000
    })
})
```

In [10]:
```
"""
Initialize tokenizer and model.
"""
model_id = "distilbert-base-uncased"

# init tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id)

# init model
model = DistilBertForSequenceClassification.from_pretrained(
    model_id,
    num_labels=5,
)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the mo
del checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bi
as', '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 [11]:
```
"""
Tokenize dataset.
"""
def tokenize(X):
    return tokenizer(
```

```
        X["text"],
        padding="max_length",
        truncation=True,
        return_tensors="pt",
    )

# tokenize data
tokenized_datasets = dataset.map(tokenize, batched=True)
print(tokenized_datasets)
```

```
DatasetDict({
    train: Dataset({
        features: ['label', 'text', 'input_ids', 'attention_mask'],
        num_rows: 650000
    })
    test: Dataset({
        features: ['label', 'text', 'input_ids', 'attention_mask'],
        num_rows: 50000
    })
})
```

In [12]:
```
"""
Create dataset splits.
"""
seed = 777
n_samples = 10_000
n_test = 1000

train_dataset = tokenized_datasets["train"].shuffle(seed=seed).select(range(n_sampl
eval_dataset = tokenized_datasets["test"].shuffle(seed=seed).select(range(0, n_test
test_dataset = tokenized_datasets["test"].shuffle(seed=seed).select(range(n_test, 2
```

In [16]:
```
"""
Create Trainer.
"""
# define metric
metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)

# training args
training_args = TrainingArguments(
    output_dir="C:/Users/Admin/Desktop/cse842_hw3",
    eval_strategy="epoch",
)

# init trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    compute_metrics=compute_metrics,
```

```
)
```

In [17]:
```
"""
Train model.
"""
trainer.train()
```

[3750/3750 9:13:23, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.005000	0.998041	0.568000
2	0.661800	1.056703	0.605000
3	0.370600	1.343305	0.605000

Out[17]:
```
TrainOutput(global_step=3750, training_loss=0.6600034830729167, metrics={'train_runtime': 33212.5936, 'train_samples_per_second': 0.903, 'train_steps_per_second': 0.113, 'total_flos': 3974234572800000.0, 'train_loss': 0.6600034830729167, 'epoch': 3.0})
```

In [ ]:
```
"""
Evaluate trained model.
"""
trainer.evaluate(test_dataset)
```

In [18]:
```
"""
Summarize model.
"""
print(model)
```

```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): DistilBertSdpaAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

In [ ]: