Sentiment Analysis on Yelp Restaurant Reviews

The goal of this project is to perform sentiment analysis on the Yelp Restaurant Review dataset. The dataset contains reviews labeled from 1-star to 5-star. Our objective is to beat the baseline model (AdhamEhab/fine-tuned-bert-yelp) in classification performance using a fine-tuned pre-trained language model.

```
pip install -q transformers datasets evaluate scikit-learn accelerate torch torchvision
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 3.2 MB/s eta 0:00:00
```

Imports & Global Config

```
import time
import re
import numpy as np
import torch
import json

from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding
)

from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    precision_recall_fscore_support
)

import evaluate
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

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

Loading the Yelp Review Dataset (Train + Test Only)

```
dataset = load_dataset("yelp_review_full")

train_ds = dataset["train"]
test_ds = dataset["test"]

print(train_ds)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md:       6.72k/? [00:00<00:00, 316kB/s]
yelp_review_full/train-00000-of-00001.pa(…): 100%                                                    299M/299M [00:02<00:00, 185MB/s]
yelp_review_full/test-00000-of-00001.par(…): 100%                                                    23.5M/23.5M [00:00<00:00, 29.1MB/s]
Generating train split: 100%                                                    650000/650000 [00:05<00:00, 165585.56 examples/s]
Generating test split: 100%                                                    50000/50000 [00:00<00:00, 127873.37 examples/s]
Dataset({
    features: ['label', 'text'],
    num_rows: 650000
})
```

Text Cleaning

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)       # remove URLs
    text = re.sub(r"\s+", " ", text)          # normalize spaces
    text = text.strip()
    return text
```

Applying Preprocessing to Dataset

Applies the clean_text function to all reviews in the training and test sets to standardize and remove noise.

```
train_ds = train_ds.map(lambda x: {"text": clean_text(x["text"])}, num_proc=4)
test_ds = test_ds.map(lambda x: {"text": clean_text(x["text"])}, num_proc=4)
```

```
Map (num_proc=4): 100%  650000/650000 [01:43<00:00, 3620.92 examples/s]
Map (num_proc=4): 100%  50000/50000 [00:07<00:00, 7101.68 examples/s]
```

Loading RoBERTa Tokenizer

Loads the tokenizer for the `roberta-base` model to convert text into token IDs suitable for model input.

```
MODEL_NAME = "roberta-base"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
```

```
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 458B/s]
config.json: 100%  481/481 [00:00<00:00, 12.0kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 10.1MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 4.09MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 5.31MB/s]
```

Tokenization Function

Defines a function to tokenize the text using the RoBERTa tokenizer.

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

Tokenizing the Dataset

Applies the `tokenize` function to all reviews in the training and test sets and formats them as PyTorch tensors for model training.

```
train_ds = train_ds.map(tokenize, batched=True, remove_columns=["text"], num_proc=4)
test_ds = test_ds.map(tokenize, batched=True, remove_columns=["text"], num_proc=4)

train_ds.set_format("torch")
test_ds.set_format("torch")
```

```
Map (num_proc=4): 100%  650000/650000 [06:16<00:00, 1853.40 examples/s]
Map (num_proc=4): 100%  50000/50000 [00:23<00:00, 3126.42 examples/s]
```

Loading the RoBERTa Model

Loads the pre-trained `roberta-base` model for sequence classification, configured to predict 5 sentiment labels.

```
model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME,
    num_labels=5
)
model.gradient_checkpointing_enable()
```

```
model.safetensors:  100%                                                499M/499M  [00:03<00:00,  214MB/s]
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newl
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Defining Evaluation Metrics

Loads the accuracy metric and defines a function to compute **accuracy, precision, recall, and weighted F1-score** for model evaluation.

```
accuracy_metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)

    precision, recall, f1, _ = precision_recall_fscore_support(
        labels, preds, average="weighted"
    )

    acc = accuracy_metric.compute(
        predictions=preds,
        references=labels
    )["accuracy"]

    return {
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1
    }
```

```
Downloading  builder  script:          4.20k/?  [00:00<00:00,  294kB/s]
```

Setting Training Arguments

Defines the hyperparameters and configuration for fine-tuning the RoBERTa model.

```
training_args = TrainingArguments(
    output_dir="./yelp_roberta",
    do_train=True,
    do_eval=True,
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=64,
    gradient_accumulation_steps=2,
    num_train_epochs=2,
    weight_decay=0.01,
    logging_steps=2000,
    save_steps=5000,
    fp16=True,
    report_to="none"
)
```

Initializing the Trainer

Creates a **Trainer** object that brings together the model, training configuration, datasets, tokenizer, and evaluation metrics.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
```

```
    tokenizer=tokenizer,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)
```

```
/tmp/ipython-input-2422701701.py:1: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trair
  trainer = Trainer(
```

Training the Model and Measuring Training Time

Starts the fine-tuning process and measures the total training time in minutes.

```
# Path to save training time JSON
output_dir = "/content/drive/MyDrive/yelp_sentiment_model"
os.makedirs(output_dir, exist_ok=True)

# Measure training time
start_time = time.time()
train_output = trainer.train()
end_time = time.time()


# Compute training time
training_time_sec = int(end_time - start_time)
training_time_minutes = training_time_sec / 60
training_time_hms = time.strftime("%H:%M:%S", time.gmtime(training_time_sec))


# Create JSON content
training_time_info = {
    "training_time_sec": training_time_sec,
    "training_time_min": round(training_time_minutes, 2),
    "training_time_hms": training_time_hms,
    "stopped_early": trainer.state.global_step < trainer.args.max_steps,
    "stopped_step": trainer.state.global_step,
    "best_checkpoint_step": int(trainer.state.best_model_checkpoint.split("-")[-1])
        if trainer.state.best_model_checkpoint else None,
    "planned_max_steps": trainer.args.max_steps,
    "device": str(trainer.args.device),
    "precision": "fp16" if trainer.args.fp16 else "fp32"
}


# Save JSON file
json_path = os.path.join(output_dir, "training_time.json")
with open(json_path, "w") as f:
    json.dump(training_time_info, f, indent=2)


print(f"Training time: {training_time_minutes:.2f} minutes")
print(f"Training time JSON saved at: {json_path}")
```


[20314/20314 2:39:26, Epoch 2/2]

Step	Training Loss
2000	0.895100
4000	0.823300
6000	0.795500
8000	0.786400
10000	0.774700
12000	0.730200
14000	0.726000
16000	0.721700
18000	0.719700
20000	0.714700

```
Training time: 159.47 minutes
Training time JSON saved at: /content/drive/MyDrive/yelp_sentiment_model/training_time.json
```

Generating Predictions on the Test Set

Uses the trained model to generate predictions on the test dataset for final evaluation.

```
predictions = trainer.predict(test_ds)

y_true = predictions.label_ids
y_pred = np.argmax(predictions.predictions, axis=1)
```

[4496/20314 34:59 < 2:03:09, 2.14 it/s, Epoch 0.44/2]

Step	Training Loss
2000	0.734700
4000	0.728200

[782/782 01:23]

Computing the Confusion Matrix

Calculates and displays the confusion matrix to analyze classification performance across all sentiment classes.

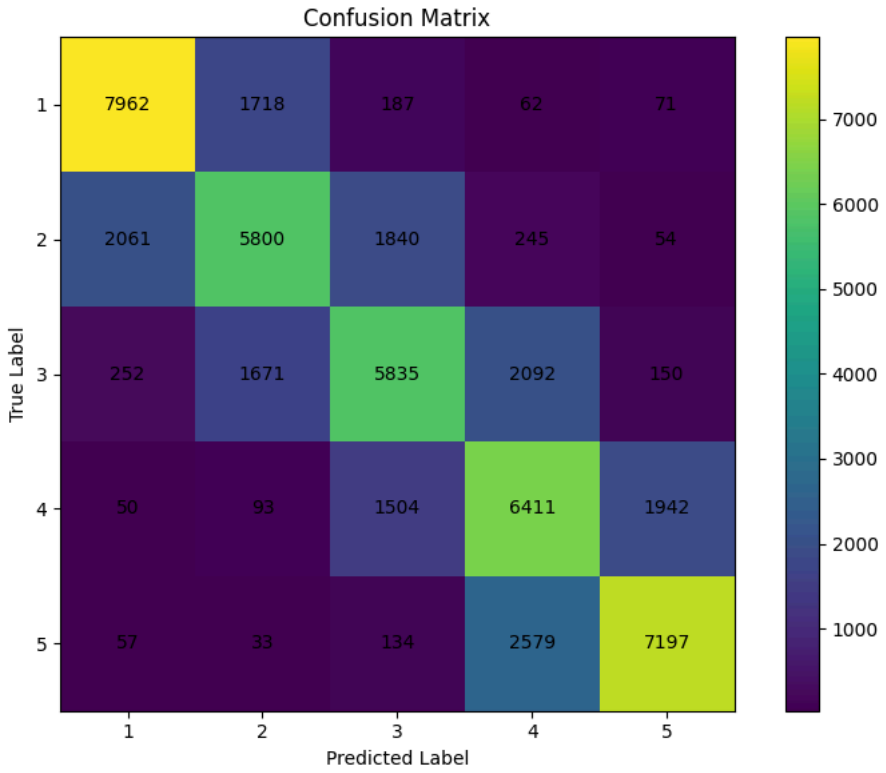
```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
plt.imshow(cm)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.colorbar()

plt.xticks(range(5), range(1, 6))
plt.yticks(range(5), range(1, 6))

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center")

plt.tight_layout()
plt.show()
```



RoBERTa Model Evaluation

This section presents the evaluation results of the fine-tuned RoBERTa model by reporting class-wise precision, recall, F1-score, and overall accuracy on the Yelp Review Full test dataset.

```
print(f"Model: {MODEL_NAME}")
print(
    classification_report(
        y_true,
        y_pred,
        target_names=[
            "1 star",
            "2 stars",
            "3 stars",
            "4 stars",
            "5 stars"
        ]
    )
)
accuracy = (y_true == y_pred).mean()
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Model: roberta-base
              precision    recall  f1-score   support

      1 star       0.77      0.80      0.78     10000
     2 stars       0.62      0.58      0.60     10000
     3 stars       0.61      0.58      0.60     10000
     4 stars       0.56      0.64      0.60     10000
     5 stars       0.76      0.72      0.74     10000

    accuracy                           0.66     50000
   macro avg       0.67      0.66      0.66     50000
weighted avg       0.67      0.66      0.66     50000

Test Accuracy: 0.6641
```

```
import os
import json
import numpy as np

# Output directory
roberta_dir = "/content/drive/MyDrive/yelp_sentiment_model/roberta"
os.makedirs(roberta_dir, exist_ok=True)

# 1. Save confusion matrix (already computed)
np.save(
    os.path.join(roberta_dir, "confusion_matrix_test.npy"),
    cm
)
print("confusion_matrix_test.npy saved successfuly")

# 2. Save metrics (already available from trainer)
eval_metrics = trainer.evaluate()

with open(os.path.join(roberta_dir, "test_metrics.json"), "w") as f:
    json.dump(eval_metrics, f, indent=2)
print("test_metrics.json saved successfuly")

# 3. Save classification report (already computed)
from sklearn.metrics import classification_report

report = classification_report(
    y_true,
    y_pred,
    target_names=["1 star", "2 stars", "3 stars", "4 stars", "5 stars"]
)

with open(os.path.join(roberta_dir, "classification_report_test.txt"), "w") as f:
    f.write(report)
print("classification_report_test.txt saved successfuly")
```

```
confusion_matrix_test.npy saved successfuly
```

[4496/20314 34:59 < 2:03:09, 2.14 it/s, Epoch 0.44/2]

Step	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
2000	0.734700					
4000	0.728200					
4495	0.728200	0.775072	0.664100	0.666236	0.664100	0.664240

```
test_metrics.json saved successfuly
classification_report_test.txt saved successfuly
```

Loading the Baseline BERT Model

Loads the pre-trained baseline model (`AdhamEhab/fine-tuned-bert-yelp`) and its tokenizer for comparison against our RoBERTa model.

```
BASELINE_MODEL = "AdhamEhab/fine-tuned-bert-yelp"

baseline_tokenizer = AutoTokenizer.from_pretrained(BASELINE_MODEL)
baseline_model = AutoModelForSequenceClassification.from_pretrained(BASELINE_MODEL)
```

```
tokenizer_config.json:  1.27k/?  [00:00<00:00,  123kB/s]
vocab.txt:  232k/?  [00:00<00:00,  10.2MB/s]
special_tokens_map.json:  100%  695/695  [00:00<00:00,  78.5kB/s]
config.json:  100%  917/917  [00:00<00:00,  113kB/s]
model.safetensors:  100%  438M/438M  [00:05<00:00,  66.2MB/s]
```

Tokenization Function for Baseline BERT Model

```
def tokenize_baseline(batch):
    return baseline_tokenizer(
        batch["text"],
        truncation=True,
        max_length=256
    )
```

Preparing the Baseline Test Dataset

Applies text preprocessing and tokenization to the test dataset to prepare it for evaluation with the baseline BERT model.

```
baseline_test = dataset["test"].map(
    lambda x: {"text": clean_text(x["text"])}
)

baseline_test = baseline_test.map(
    tokenize_baseline,
    batched=True,
    remove_columns=["text"]
)

baseline_test.set_format("torch")
```

```
Map:  100%  50000/50000  [00:04<00:00,  10724.57  examples/s]
Map:  100%  50000/50000  [00:30<00:00,  1745.10  examples/s]
```

Trainer for the Baseline BERT Model

```
baseline_trainer = Trainer(
    model=baseline_model,
    tokenizer=baseline_tokenizer,
    data_collator=DataCollatorWithPadding(baseline_tokenizer),
    compute_metrics=compute_metrics
```

```
)
```

```
/tmp/ipython-input-1501265769.py:1: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trair
  baseline_trainer = Trainer(
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in v5. Use the --report_to flag to control
```

Predictions with the Baseline BERT Model

```
baseline_preds = baseline_trainer.predict(baseline_test)

baseline_y_true = baseline_preds.label_ids
baseline_y_pred = np.argmax(baseline_preds.predictions, axis=1)
```

Computing the Confusion Matrix for the Baseline BERT Model

Calculates and displays the confusion matrix for the baseline model to analyze classification performance across all sentiment classes.

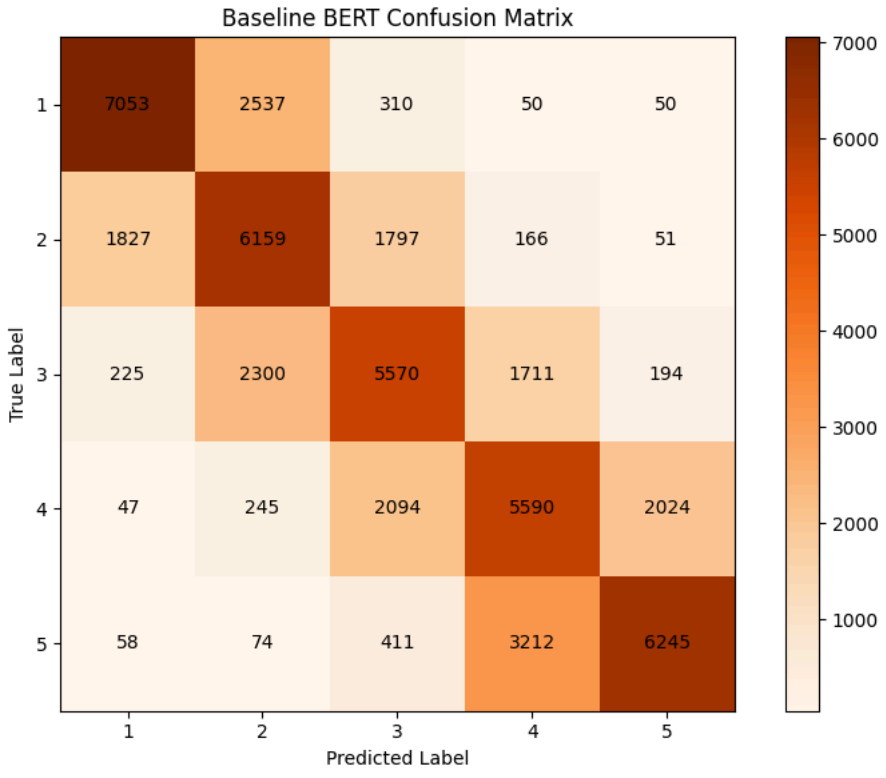
```
cm = confusion_matrix(baseline_y_true, baseline_y_pred)

plt.figure(figsize=(8, 6))
plt.imshow(cm, cmap="Oranges")  # Changed color to Oranges
plt.title("Baseline BERT Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.colorbar()

plt.xticks(range(5), range(1, 6))
plt.yticks(range(5), range(1, 6))

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha="center", va="center", color="black")  # Changed text color for visibility

plt.tight_layout()
plt.show()
```



Baseline BERT Model Classification Report

Generates a detailed classification report for the baseline BERT model, showing precision, recall, F1-score for each sentiment class, and overall test accuracy.

```
print("Baseline Classification Report:\n")
print(
    classification_report(
        baseline_y_true,
        baseline_y_pred,
        target_names=[
            "1 star",
            "2 stars",
            "3 stars",
            "4 stars",
            "5 stars"
        ]
    )
)

accuracy = (baseline_y_true == baseline_y_pred).mean()
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Baseline Classification Report:

              precision    recall  f1-score   support

      1 star       0.77      0.71      0.73     10000
     2 stars       0.54      0.62      0.58     10000
     3 stars       0.55      0.56      0.55     10000
     4 stars       0.52      0.56      0.54     10000
     5 stars       0.73      0.62      0.67     10000

    accuracy                           0.61     50000
   macro avg       0.62      0.61      0.62     50000
weighted avg       0.62      0.61      0.62     50000

Test Accuracy: 0.6123
```

```
import os
import json
import numpy as np

# Output directory
cm = confusion_matrix(baseline_y_true, baseline_y_pred)
baseline_dir = "/content/drive/MyDrive/yelp_sentiment_model/baseline_bert"
os.makedirs(baseline_dir, exist_ok=True)

# 1. Save confusion matrix
np.save(
    os.path.join(baseline_dir, "confusion_matrix_test.npy"),
    cm
)
print("confusion_matrix_test.npy saved successfuly")


# 2. Save classification report
from sklearn.metrics import classification_report

baseline_report = classification_report(
    baseline_y_true,
    baseline_y_pred,
    target_names=["1 star", "2 stars", "3 stars", "4 stars", "5 stars"]
)

with open(os.path.join(baseline_dir, "classification_report_test.txt"), "w") as f:
    f.write(baseline_report)

print("classification_report_test.txt saved successfuly")
```

```
confusion_matrix_test.npy saved successfuly
classification_report_test.txt saved successfuly
```

Comparison of Weighted F1-scores: Baseline BERT vs Fine-tuned RoBERTa

Creates a table comparing the weighted F1-scores of the baseline BERT model and fine-tuned RoBERTa model.

```
baseline_f1 = precision_recall_fscore_support(
    baseline_y_true, baseline_y_pred, average="weighted"
)[2]
```

```
our_f1 = precision_recall_fscore_support(
    y_true, y_pred, average="weighted"
)[2]

comparison = pd.DataFrame({
    "Model": [
        "Baseline (BERT – AdhamEhab)",
        "Our Model (RoBERTa)"
    ],
    "Weighted F1-score": [
        baseline_f1,
        our_f1
    ]
})

print(comparison)
```

```
                        Model  Weighted F1-score
0  Baseline (BERT – AdhamEhab)           0.615267
1          Our Model (RoBERTa)           0.664240
```

Saving the Trained Models and Tokenizers

Saves both the fine-tuned RoBERTa model and the baseline BERT model along with their respective tokenizers for future use or deployment.

```
trainer.save_model("./final_roberta_model")
tokenizer.save_pretrained("./final_roberta_model")

baseline_model.save_pretrained("./baseline_model")
baseline_tokenizer.save_pretrained("./baseline_model")
```

```
('./baseline_model/tokenizer_config.json',
 './baseline_model/special_tokens_map.json',
 './baseline_model/vocab.txt',
 './baseline_model/added_tokens.json',
 './baseline_model/tokenizer.json')
```