# Training the dataset

- Two type of dataset was created , dataset with stopword removal and lemmatization and dataset without stopword removal and lemmatization
- Dataset was trained on Bert and Roberta with 6 epoch

| Feature | BERT-base | RoBERTa-base |
|---|---|---|
| **Architecture** | 12 Transformer encoder layers, 768 hidden size, 12 self-attention heads | 12 Transformer encoder layers, 768 hidden size, 12 self-attention heads |
| **Parameters** | ~110 million | ~125 million |
| **Tokenizer** | **WordPiece** tokenizer (vocabulary size ~30k) | **Byte-Pair Encoding (BPE)** tokenizer (vocabulary size ~50k) |

## Training argument

```
training_args = TrainingArguments(
    output_dir="./bert-review-classifier",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=4,
    learning_rate=4e-5,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=50,
    save_total_limit=1,
    save_strategy="epoch",
    report_to="none",
    fp16=True
)
```

# 1.BERT-BASE

## 1.1 Dataset with stop word removal and lemmatization

Learning rate of 5e-5: Train accuracy=96%, val accuracy=58%

Learning rate of 4e-5: Train accuracy=95%, val accuracy=58%

Learning rate of 3e-5: Train accuracy=94%, val accuracy=59%

## 1.2 Dataset without stop word removal and lemmatization

Learning rate of 5e-5: Train accuracy=97% , val accuracy=63.4%

**Learning rate of 4e-5: Train accuracy=97.1% , val accuracy=63.9%**

Learning rate of 3e-5: Train accuracy=96.2% , val accuracy=63.7%

Learning rate of 2e-5: Train accuracy= 93.7%, val accuracy=64.3%

Learning rate of 1e-5: Train accuracy=84.2% , val accuracy=65.4%

# 2.ROBERTA-BASE

## 2.1 Dataset without stop word removal and lemmatization

Learning rate of 1e-5: Train accuracy=77% , val accuracy=67.4%

Learning rate of 2e-5: Train accuracy=87.1% , val accuracy=66.4%

Learning rate of 3e-5: Train accuracy=87% , val accuracy=66.4%

# Result

Learning rate of 4e-5 without stop word removal and lemmatization using Bert base achieved higher accuracy of Train accuracy=97.1% , val accuracy=63.9%

**Model was trained on whole dataset**

```
from sklearn.metrics import classification_report
target_names = ["Rating 1", "Rating 2", "Rating 3", "Rating 4", "Rating 5"]
predictions_output = trainer.predict(new_dataset)
y_pred = np.argmax(predictions_output.predictions, axis=1)
y_true = predictions_output.label_ids

print("\nClassification Report:")
print(classification_report(y_true, y_pred, digits=4,target_names=target_names))
```

```
Classification Report:
              precision    recall  f1-score   support

    Rating 1     0.9270    0.9386    0.9328     27888
    Rating 2     0.8377    0.8137    0.8255     14823
    Rating 3     0.8673    0.8622    0.8648     21670
    Rating 4     0.8963    0.8897    0.8930     38052
    Rating 5     0.9380    0.9476    0.9428     47118

    accuracy                         0.9056    149551
   macro avg     0.8933    0.8904    0.8918    149551
weighted avg     0.9052    0.9056    0.9053    149551
```

The classification report summarizes the performance of the model across all five rating categories. For each rating (1 to 5), the metrics reported include **precision, recall, and F1-score**, along with the total number of instances (support).

- **Rating 1** achieved the highest balance between precision (0.9270), recall (0.9386), and F1-score (0.9328), indicating strong performance in correctly classifying this class.
- **Rating 2** shows the lowest performance overall, with a precision of 0.8377, recall of 0.8137, and F1-score of 0.8255, suggesting the model struggled more with this class compared to others.
- **Ratings 3, 4, and 5** show consistently high performance, with F1-scores ranging from 0.8648 to 0.9428, demonstrating that the model was effective at predicting these categories.

At the overall level, the model achieved an **accuracy of 90.56%**, with a **macro-average F1-score of 0.8918**, and a **weighted-average F1-score of 0.9053**. The macro-average indicates that performance was fairly balanced across classes, while the weighted-average accounts for class imbalance and confirms strong overall predictive ability.

Code:

```
model = AutoModelForSequenceClassification.from_pretrained('./epoch 6')
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {
        "accuracy": accuracy_score(labels, predictions),
        "f1": f1_score(labels, predictions, average="weighted")
    }


from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    per_device_eval_batch_size=16,
    do_train=False,
    do_eval=True,
    report_to="none",
    fp16=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
from sklearn.metrics import classification_report
```

```
target_names = ["Rating 1", "Rating 2", "Rating 3", "Rating 4", "Rating 5"]
predictions_output = trainer.predict(new_dataset)
y_pred = np.argmax(predictions_output.predictions, axis=1)
y_true = predictions_output.label_ids

print("\nClassification Report:")
print(classification_report(y_true, y_pred, digits=4,target_names=target_names))
```