

# Topic

The research question is about NLP. The task for supervised learning models is sentiment analysis, and for unsupervised learning model is topic modeling.

# Description Of Dataset

The training dataset is obtained from comments of a video on YouTube. The topic of the video is the election debate between Joe Biden and Donald Trump. These comments are fetched from Google API, filling the id of the video and save into a csv file. Then remove emojis in the comments and get rid of those comments that are not in English. Finally, use Textblob model to label each comment, 1 for Positive, 0 for Neutral, -1 for Negative. This dataset is for training and testing during the training process. Besides the comments from the debate, I downloaded other comments from a video whose topic is the discussion of the debate between NewJeans, a Korean girl group, and its companies, Hybe and Ador in court. This dataset is for testing the model only.

	Training	Testing
Election debate	55877	13970
Court debate	X	123

	Positive	Neutral	Negative
Election debate	27917	25364	16566

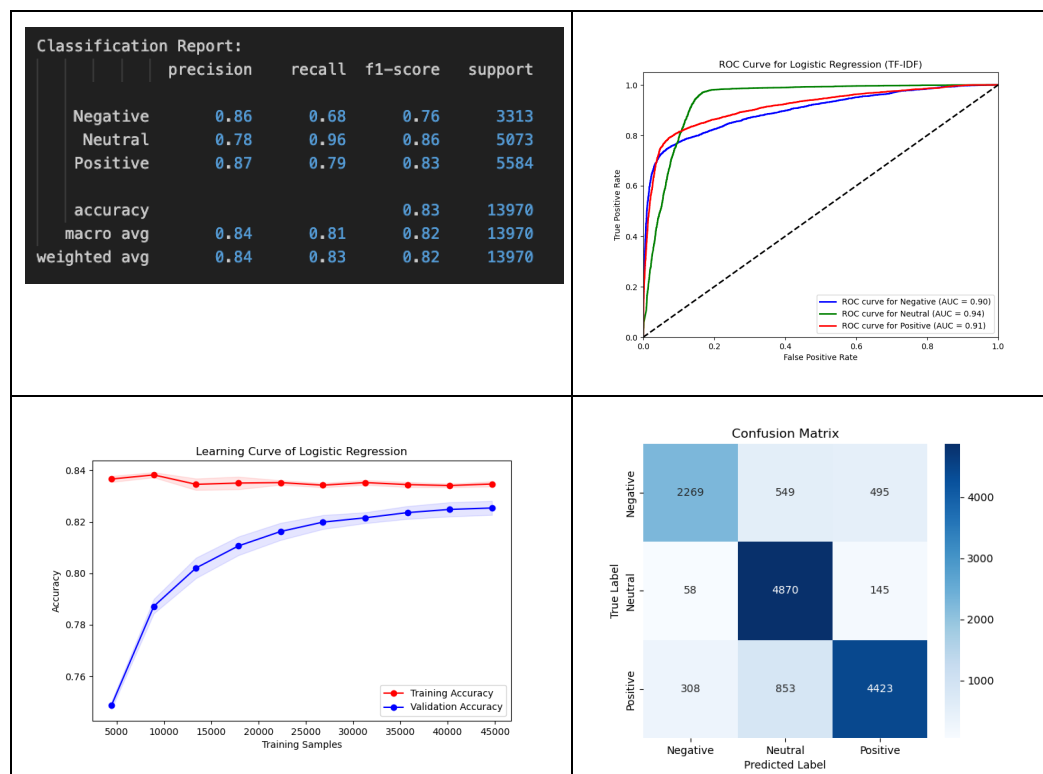
# Supervised Learning

## Logistic Regression with Tfidf

Term Frequency-Inverse Document Frequency (Tfidf) is a statistical measure used in information retrieval and text mining to evaluate the importance of a word in a document relative to a collection of documents. Logistic Regression model is a model for classification problem. In this task, it takes the Tfidf value of a comment as input. Both Tfidf and Logistic Regression model are implemented in Scikit-Learn (sklearn), an open-source machine learning library for Python, built on top of Numpy, SciPy, and Matplotlib.

### Result

- Cross-Validation Scores:  
[0.82202935, 0.82784538, 0.82487696, 0.82290828, 0.82917226]
- Mean Accuracy: 82.54%
- Standard Deviation: 0.0028
- Final Accuracy: 82.763%



# BERT with Word2Vec

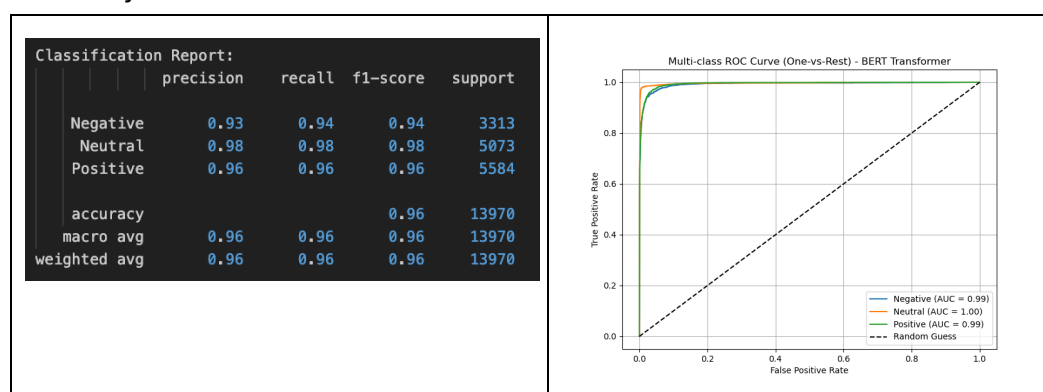
Word2Vec is a shallow, two-layer neural network to create word embeddings. It learns relationships between word based on contest. It creates fixed-size word embeddings. But doesn't handle polysemy. Bidirectional Encoder Representations from Transformers (BERT) is a deep neural network based on the transformer architecture. It revolutionized NLP by understanding context bidirectionally, means it considers both left and right context when processing words. In this work, I don't train a BERT model from begin. Instead, I used pretrained BERT model to do transfer learning. This approach can save me a lot of time. Word2vec is implemented in sklearn; while BERT can be done with PyTorch transformer.

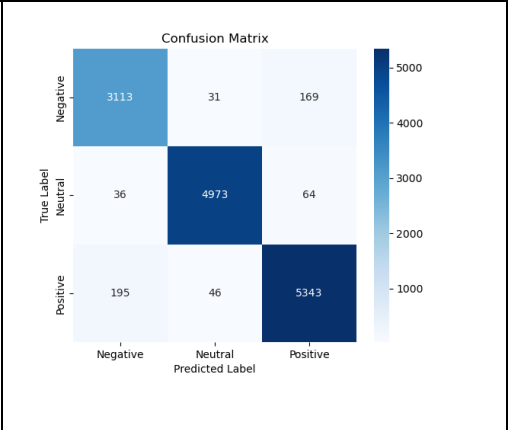
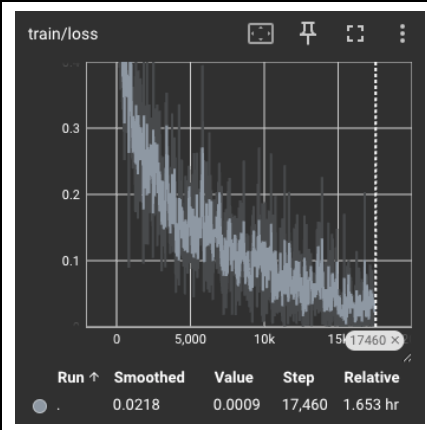
## Training Parameters

- Batch Size: 16
- Epochs: 5
- Initial Learning Rate: 5e-5
- Weight Decay: 0.01

## Result

- Accuracy: 96.13%





# Unsupervised Learning

## LDA

Latent Dirichlet Allocation (LDA) is a generative probabilistic model for topic modeling in NLP. IT helps uncover the hidden topics in a collection of documents. LDA model is implemented in sklearn.

## Result

```
Topic 1:
people vote american want biden election dont need watch real
Topic 2:
old war country man run candidates ukraine number biden president
Topic 3:
trump biden president 2024 best love donald win mr liar
Topic 4:
debate bidens biden trumps lying face presidential cnn brandon rfk
Topic 5:
biden trump question questions answer like golf inflation 100 went
Topic 6:
biden idea joe needs cnn home like feel way retire
Topic 7:
biden trump im time lies says saying thought tell speak
Topic 8:
world joe country america biden usa god im americans president
Topic 9:
dont think said know biden lol people like black jobs
Topic 10:
biden like hes president years america looks make worst great
```

## Analysis

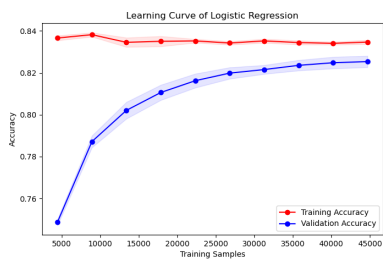
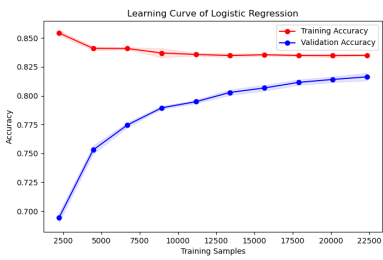
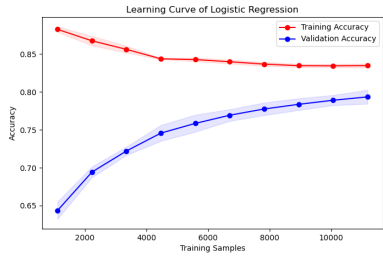
The result from LDA is not as my expected. I expected that the topics the model could have found was terms in some specific aspect such as economic, civilian, transport, and war. But it turned out that a lot of key word are “Biden” and “Trump”, which are obviously not I wanted to see.

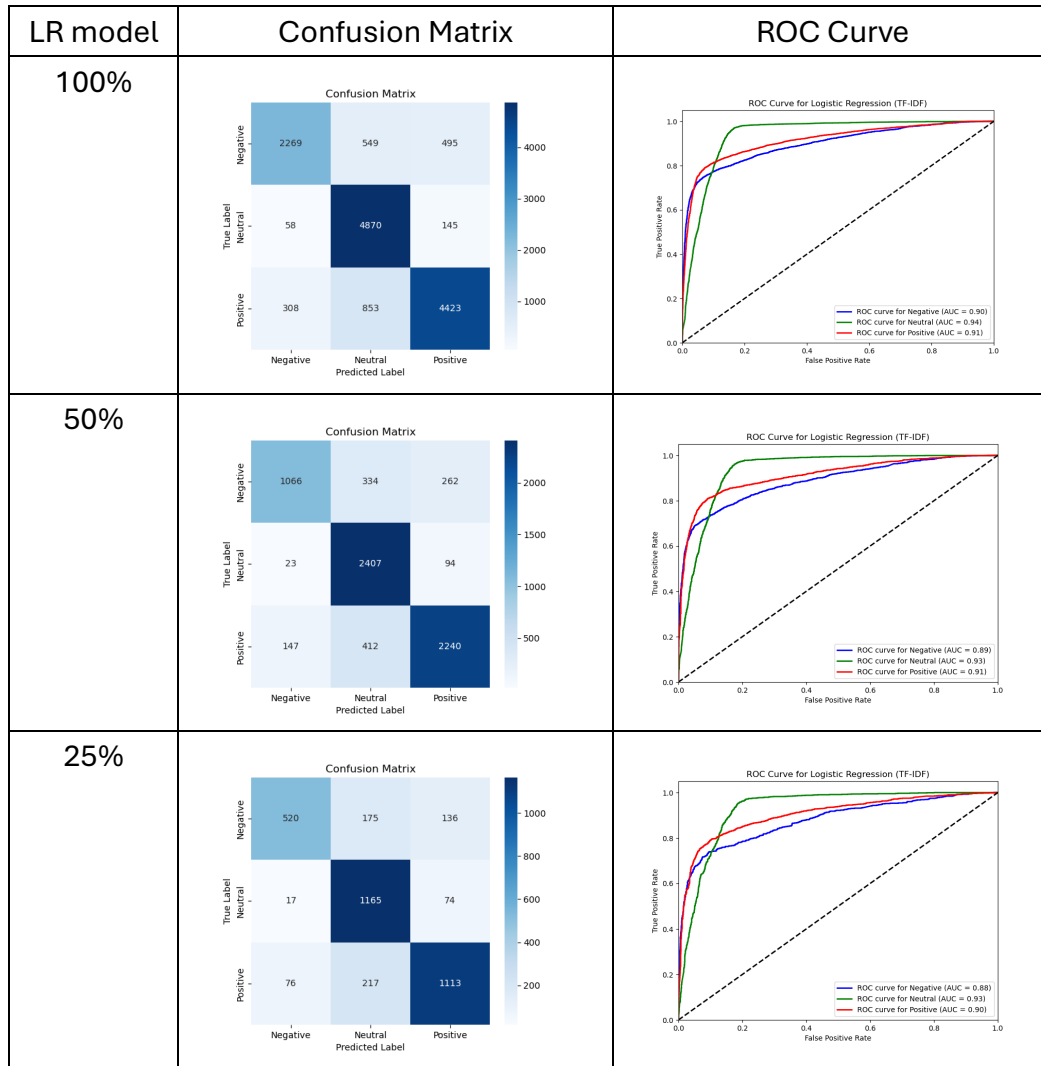
# Experiment

## The Influence of The Size of The Dataset

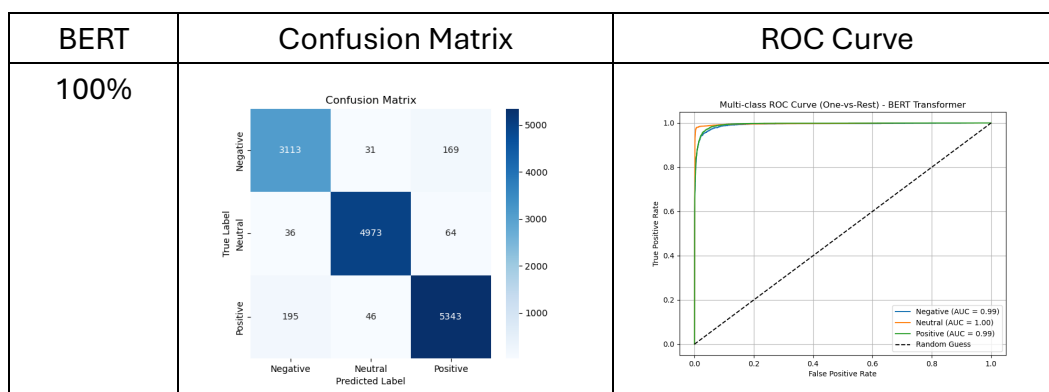
The Logistic Regression model and BERT model are trained with different size dataset, 100%, 50%, 25%. The expected result was that the accuracy went down as the size of the dataset shrink. The real result was that the accuracy indeed went down as the size of the dataset shrink. But it didn't decrease significantly, for both Logistic Regression model and BERT model. Therefore, the conclusion for my experiment is that the amount of training data doesn't really influent the performance.

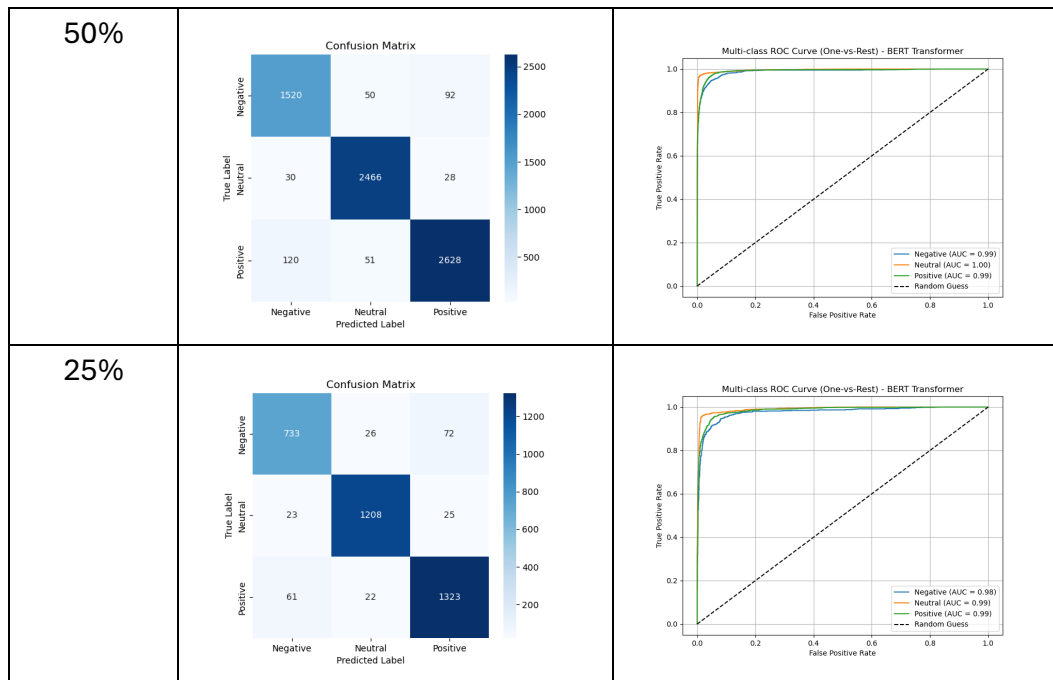
LR model	100%	50%	25%
Accuracy	82.763%	81.789%	80.103%

LR model	Classification Report	Learning Curve																																			
100%	<div>Classification Report:</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>Negative</td><td>0.86</td><td>0.68</td><td>0.76</td><td>3313</td></tr><tr><td>Neutral</td><td>0.78</td><td>0.96</td><td>0.86</td><td>5073</td></tr><tr><td>Positive</td><td>0.87</td><td>0.79</td><td>0.83</td><td>5584</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.83</td><td>13970</td></tr><tr><td>macro avg</td><td>0.84</td><td>0.81</td><td>0.82</td><td>13970</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.83</td><td>0.82</td><td>13970</td></tr></table>		precision	recall	f1-score	support	Negative	0.86	0.68	0.76	3313	Neutral	0.78	0.96	0.86	5073	Positive	0.87	0.79	0.83	5584	accuracy			0.83	13970	macro avg	0.84	0.81	0.82	13970	weighted avg	0.84	0.83	0.82	13970	
	precision	recall	f1-score	support																																	
Negative	0.86	0.68	0.76	3313																																	
Neutral	0.78	0.96	0.86	5073																																	
Positive	0.87	0.79	0.83	5584																																	
accuracy			0.83	13970																																	
macro avg	0.84	0.81	0.82	13970																																	
weighted avg	0.84	0.83	0.82	13970																																	
50%	<div>Classification Report:</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>Negative</td><td>0.86</td><td>0.64</td><td>0.74</td><td>1662</td></tr><tr><td>Neutral</td><td>0.76</td><td>0.95</td><td>0.85</td><td>2524</td></tr><tr><td>Positive</td><td>0.86</td><td>0.80</td><td>0.83</td><td>2799</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>6985</td></tr><tr><td>macro avg</td><td>0.83</td><td>0.80</td><td>0.80</td><td>6985</td></tr><tr><td>weighted avg</td><td>0.83</td><td>0.82</td><td>0.81</td><td>6985</td></tr></table>		precision	recall	f1-score	support	Negative	0.86	0.64	0.74	1662	Neutral	0.76	0.95	0.85	2524	Positive	0.86	0.80	0.83	2799	accuracy			0.82	6985	macro avg	0.83	0.80	0.80	6985	weighted avg	0.83	0.82	0.81	6985	
	precision	recall	f1-score	support																																	
Negative	0.86	0.64	0.74	1662																																	
Neutral	0.76	0.95	0.85	2524																																	
Positive	0.86	0.80	0.83	2799																																	
accuracy			0.82	6985																																	
macro avg	0.83	0.80	0.80	6985																																	
weighted avg	0.83	0.82	0.81	6985																																	
25%	<div>Classification Report:</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>Negative</td><td>0.85</td><td>0.63</td><td>0.72</td><td>831</td></tr><tr><td>Neutral</td><td>0.75</td><td>0.93</td><td>0.83</td><td>1256</td></tr><tr><td>Positive</td><td>0.84</td><td>0.79</td><td>0.82</td><td>1406</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.80</td><td>3493</td></tr><tr><td>macro avg</td><td>0.81</td><td>0.78</td><td>0.79</td><td>3493</td></tr><tr><td>weighted avg</td><td>0.81</td><td>0.80</td><td>0.80</td><td>3493</td></tr></table>		precision	recall	f1-score	support	Negative	0.85	0.63	0.72	831	Neutral	0.75	0.93	0.83	1256	Positive	0.84	0.79	0.82	1406	accuracy			0.80	3493	macro avg	0.81	0.78	0.79	3493	weighted avg	0.81	0.80	0.80	3493	
	precision	recall	f1-score	support																																	
Negative	0.85	0.63	0.72	831																																	
Neutral	0.75	0.93	0.83	1256																																	
Positive	0.84	0.79	0.82	1406																																	
accuracy			0.80	3493																																	
macro avg	0.81	0.78	0.79	3493																																	
weighted avg	0.81	0.80	0.80	3493																																	



BERT	100%	50%	25%
Accuracy	96.13%	94.69%	93.44%





## Performance on different topic dataset

The goal of this experiment is to check whether the model trained with politic debate dataset can also perform well on course debate dataset. I expected that the accuracy might decrease due to the different topic.

The result fits what I expected. Both models have different level of decrease in accuracy. The accuracy of Logistic Regression model drops from 82.763% to 76.422%. And the accuracy of BERT model drops from 96.13% to 86.17%.

	Politic Debate	Course Debate	Difference
LR model	82.763%	76.422%	6.341
BERT model	96.13%	86.17%	9.96



# Discussion

- The result of BERT and LR model are much better than I have thought. But after seeing the performance, I thought the performance of LDA model would have also been great. But it turned that the performance was not ideal. I thought the topics the model found would be from more deeper concepts.
- The good performance for both Logistic Regression model and BERT model can be attributed to the topic of the dataset. When it comes to politics, people tend to be fierce then give out comments with emotional words. For example, humiliating words. These kinds of words can be easily learned by models.
- I want to do more experiments on different dataset with different topics, some topics that people can give more objective comments, checking whether the performance can also be good.
- This is the first NLP project for me. I learned a lot about NLP, such as models and tokenization. I also learn how to use tensorboard to visualize the training process and Hugging face trainer to train a model.

# References

- [Sklearn TfidfVectorizer](#)
- [Sklearn LogisticRegression](#)
- [transformers](#)
- [Torch.nn](#)
- [Hugging Face](#)
- [TensorBoard](#)
- [Sklearn LDA](#)

# Appendix

- [GitHub of all code](#)
- Logistic Regression

```
# ♦ Convert text into TF-IDF features
vectorizer = TfidfVectorizer(max_features=1000, stop_words="english")
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

logging.info(f"TF-IDF Feature Shape: {X_train_tfidf.shape}\n-----\n")

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
import numpy as np

# ♦ Initialize Logistic Regression Model
clf = LogisticRegression(max_iter=500, solver="lbfgs", random_state=42)

# ♦ Perform 5-Fold Cross-Validation
cv_scores = cross_val_score(clf, X_train_tfidf, y_train, cv=5, scoring="accuracy")

# ♦ Print Results
logging.info(f"Cross-Validation Scores: {cv_scores}")
logging.info(f"Mean Accuracy: {np.mean(cv_scores):.4f}")
logging.info(f"Standard Deviation: {np.std(cv_scores):.4f}\n-----\n")

from sklearn.metrics import accuracy_score, classification_report

# ♦ Train on Full Training Set
clf.fit(X_train_tfidf, y_train)

# ♦ Predict on Test Set
y_pred = clf.predict(X_test_tfidf)
```

- BERT

```
import torch.nn as nn
from transformers import AutoModel

class TransformerBERT(nn.Module):
    """Transformer model using BERT embeddings."""
    def __init__(self, num_classes=3):
        super().__init__()
        self.bert = AutoModel.from_pretrained("bert-base-uncased")
        self.transformer = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model=768, nhead=4,
                                       num_layers=2)
        )
        self.fc = nn.Linear(768, 256)
        self.dropout = nn.Dropout(0.3)
        self.out = nn.Linear(256, num_classes)

    def forward(self, input_ids, attention_mask, labels=None):
        x = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        x = self.transformer(x) # Get BERT embeddings
        x = x.mean(dim=1) # Mean pooling
        x = self.fc(x)
        x = torch.relu(x)
        x = self.dropout(x)
        logits = self.out(x)

        loss = None
        if labels is not None:
            loss_fn = nn.CrossEntropyLoss()
            loss = loss_fn(logits, labels)

        return {"loss": loss, "logits": logits}
```

```

# ♦ Define Training Arguments
training_args = TrainingArguments(
    output_dir="./bert_transformer_model_quarter_dataset",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_dir="./logs_quarter_dataset",
    logging_steps=10,
    report_to="tensorboard", # ✅ Enable TensorBoard
    use_mps_device=True # ✅ Enables Apple GPU acceleration
)

# ♦ Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)

# ♦ Train the model
trainer.train()

```

## ● LDA

```

# Download stopwords
nltk.download('stopwords')

# Convert dataset to DataFrame and use 2000 samples
df = pd.read_csv("../data/english_comments_labeled.csv")
# Text cleaning function
def clean_text(text):
    text = text.lower()
    text = re.sub(r'\W', ' ', text)
    text = ' '.join(word for word in text.split() if word not in stopwords.words('english'))
    return text

df["cleaned_review"] = df["text"].apply(clean_text)

from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

# Convert text into a document-term matrix
vectorizer_lda = CountVectorizer(max_features=500, stop_words='english')
X_train_lda = vectorizer_lda.fit_transform(df["cleaned_review"])

# Train LDA model (discover 10 topics)
lda_model = LatentDirichletAllocation(n_components=10, random_state=42)
lda_model.fit(X_train_lda)

# Print the top words per topic
feature_names = vectorizer_lda.get_feature_names_out()
for topic_idx, topic in enumerate(lda_model.components_):
    logging.info(f"Topic {topic_idx + 1}:")
    tem = " ".join([feature_names[i] for i in topic.argsort()[::-10 - 1:-1]])
    logging.info(tem)

```