**Article Classification**

Francky Ronsard Saah  
Technology Faculty,

Information Systems Engineering

Kocaeli University, Kocaeli, Turkey  
[francky877832@gmail.com](mailto:francky877832@gmail.com)

**Abstract**

This project focuses on developing a machine learning solution to categorize technical articles into five specialized domains: Deep Learning, Wireless Communication, Cloud Computing, Virtual Reality, and Large Language Models (LLM). A total of 30,506 articles were collected from arXiv.org and processed using natural language techniques such as tokenization, lemmatization, and stopword removal. Class balancing, duplicate removal, and visualization techniques were applied to ensure the dataset's quality. Five transformer-based models—BERT, RoBERTa, DeBERTa, GPT, and ALBERT—were fine-tuned and evaluated. Among them, DeBERTa achieved the best overall performance, while ALBERT stood out for fast inference. This report presents the complete workflow, evaluation results, and a comparative analysis.

**Keywords**—Article Classification, Transformers, Text Preprocessing, Model Evaluation, NLP, Machine Learning

1. **Introduction**

In the current digital era, managing and organizing technical information is vital due to the exponential growth of online content. This project addresses the challenge of classifying technical articles into five domains: Deep Learning, Wireless Communication, Cloud Computing, Virtual Reality, and Large Language Models (LLM). The work is structured in multiple stages: data collection, preprocessing, visualization, and preparation for model training. This report elaborates on each stage, emphasizing the challenges encountered and the methods used to handle them.

1. **Data Collection**
2. **Data Sources**

The datasets were collected from public online source **Arxiv.org**. Articles were manually curated to ensure relevance to the intended categories.

1. **Tools and Environment**

**Programming Language**: Python 3.9

**Development Environment**: Google Colab

**Libraries Used:** matplotlib, nltk, os, pandas, requests, seaborn, time

1. **Data Collected**

Each article record included the following features:

**Title**: Title of the article

**Summary**: Abstract or short description

**Url** : The link of the article

**Label**: Corresponding category (Deep Learning, Wireless Communication, Cloud Computing, Virtual Reality, LLM)

The dataset consisted of **30506 articles** in total, distributed as follows:

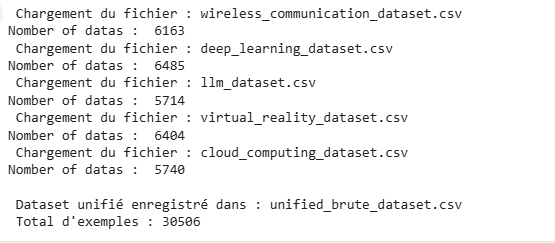
**Deep Learning**: 6485 articles

**Wireless Communication**: 6163 articles

**Cloud Computing**: 5740 articles

**Virtual Reality**: 6404 articles

**Large Language Models (LLM):** 5714 articles



1. *Data collection*
2. **Data Collection Challenges**

Several issues arose during the data collection phase:

**Class Imbalance**: Initially, the number of articles varied among categories. To address this, we applied downsampling to maintain 5000 articles per class.

**Duplicate Entries**: Duplicate articles were detected and removed using text similarity checks and record uniqueness criteria.

**Limit on the Number of Documents per Request on arXiv**: One of the main challenges encountered when retrieving documents from arXiv was the limitation on the number of documents that can be fetched per request. arXiv imposes a maximum limit on the results per query, which required me to implement a loop that performs multiple requests. This necessitated efficient query management to ensure that all the required data was retrieved while adhering to the API's constraints.

**Lack of Publications in Certain Categories (e.g., LLM)**: Another challenge was the scarcity of publications available in specific categories, such as "LLM" (Large Language Models). This limitation led to the need to increase the number of keywords in the queries to broaden the search and maximize results. Additionally, different combinations of keywords were tested to obtain a wider coverage and retrieve more relevant publications. Although this approach did not always guarantee a significant increase in the number of articles, it did help improve the results to some extent.

1. **Data Preprocessing**
2. **Text Cleaning and Preprocessing Techniques**

The following preprocessing methods were applied sequentially:

**Tokenization**: Breaking down summaries into individual words.

**Stopword Removal**: Eliminating commonly used words that do not contribute significant meaning.

**Lemmatization**: Reducing words to their root form (e.g., "running" to "run").

**Lowercasing**: Converting all text to lowercase for consistency.

**Non-Alphabetic Character Removal**: Removing punctuation and numbers.

**Duplicate Removal**: Ensuring no redundant articles remained.

Preprocessing was done using the **NLTK**.

1. **Data Statistics After Preprocessing**

After preprocessing:

* **No duplicate records remained.**
* Each class contained exactly 5000 articles, resulting in a total of 25,000 processed entries.

1. **Data Visualization**

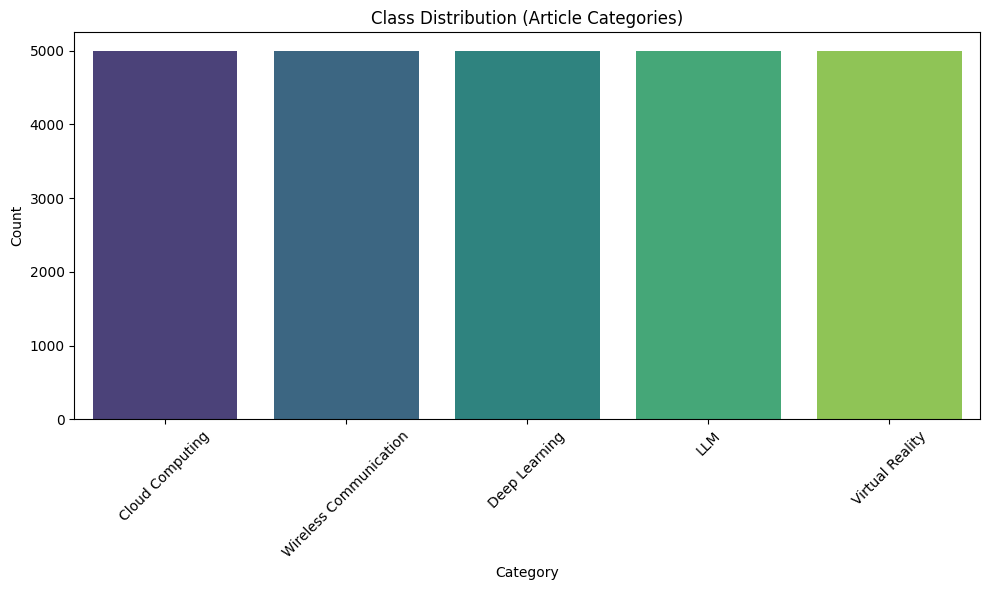
To better understand the dataset characteristics before and after preprocessing, several visualizations were produced.

**Class Distribution (Bar Plot):**

This plot shows the distribution of article categories (labels) in the dataset. It provides insight into how balanced or imbalanced the classes are. The x-axis represents the different categories, and the y-axis shows the count of articles in each category.



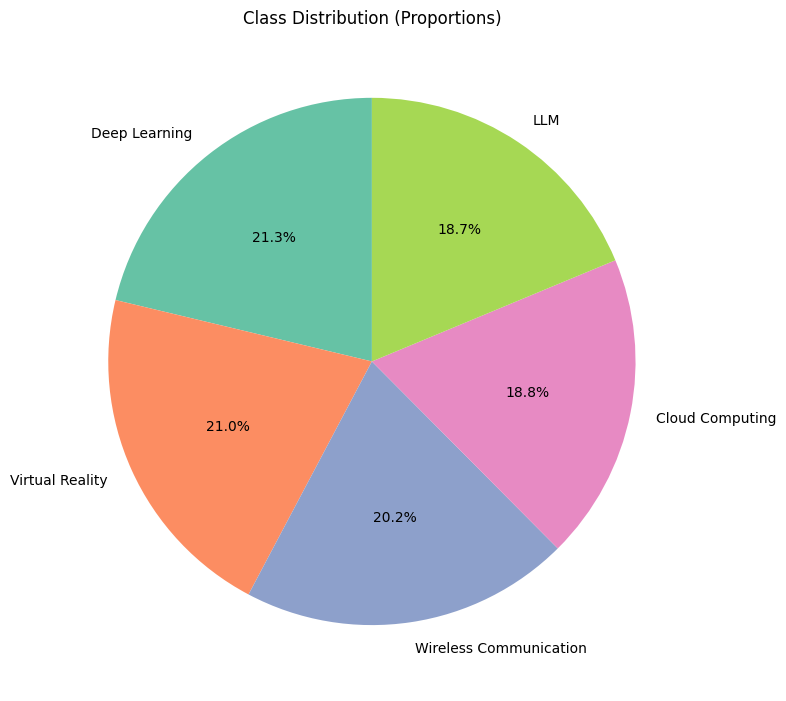
1. *Bar Plot - Class Distribution before processing*



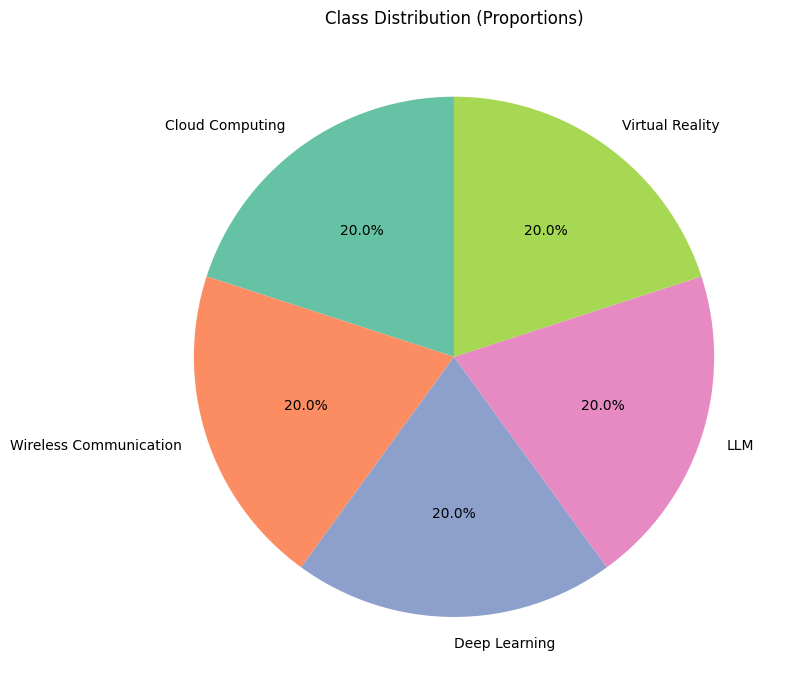
1. *Bar Plot Class Distribution after processing*

**Class Proportions (Pie Chart):**

This pie chart visualizes the proportions of each article category in the dataset, giving a clearer picture of the relative frequency of each category. It helps to understand the distribution of the dataset in terms of category proportions.



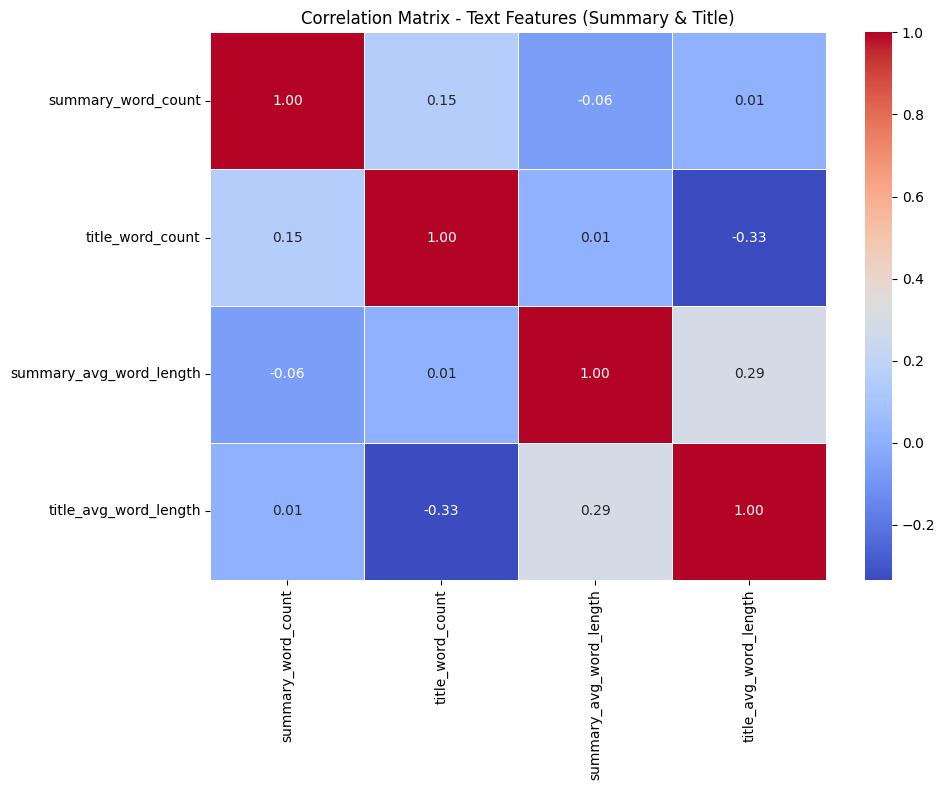
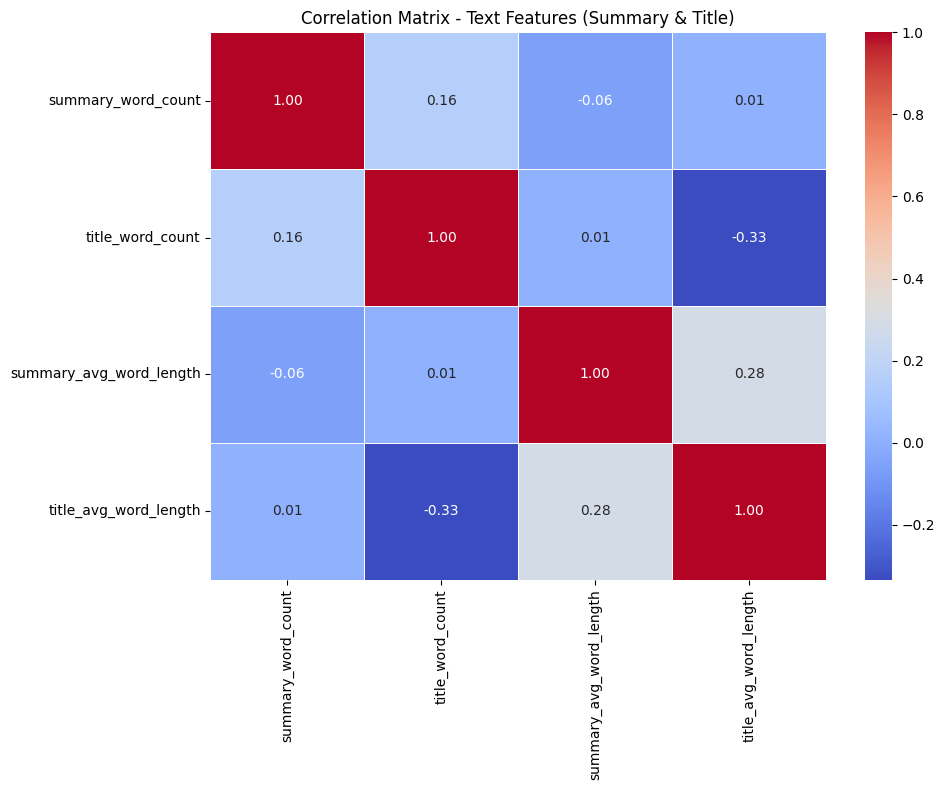
1. *Pie Chart Data Distribution before processing*



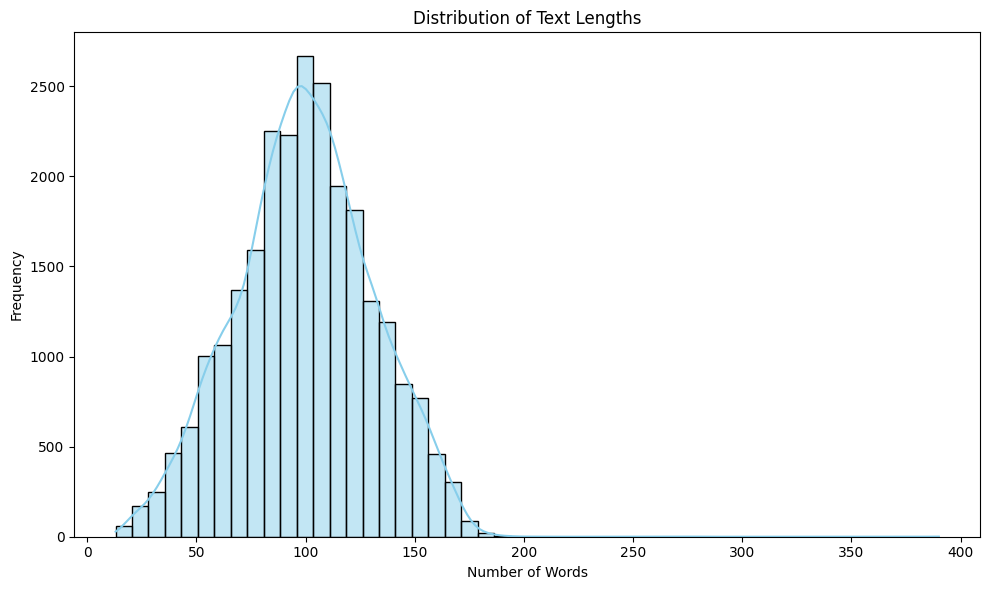
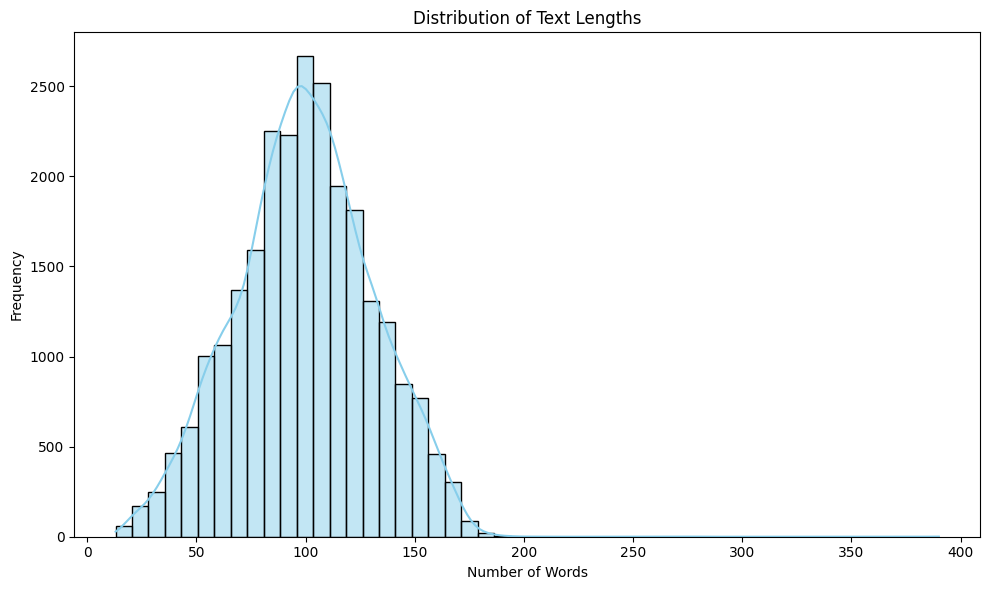
1. *Pie Chart Data Distribution after processing*

**Correlation Matrix (Text Features):**

The heatmap of the correlation matrix illustrates the relationships between different numeric features derived from the text, such as word count and average word length of the article summary and title. This helps to identify if there are any notable correlations between these text-based features, which could be useful for feature engineering in model building.

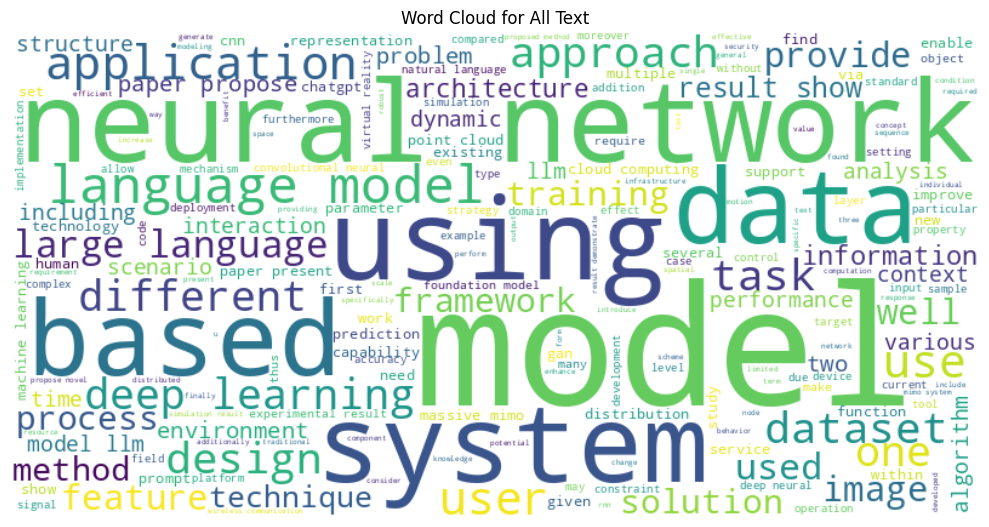


**Summary Length**



**Word Frequency(Only after processing)**

Wordclouds were created to visualize the most common words. Commonly frequent terms included *"network", "model", "data", "neural", “using”, “based”, “system”.*



**All those Visuals are provided in Shared Drive.**

1. **Model Descriptions**

* **Methodology**

In this project, I trained five transformer-based models: **BERT**, **RoBERTa**, **DeBERTa**, **GPT**, and **ALBERTa**. For each model, the following steps were followed:

**Training and Evaluation**

Each model was trained using a fine-tuning approach.

During training, I recorded the following metrics **at each training step** (not just at the end of each epoch): ***Training Loss, Validation Loss, Accuracy, Precision, Recall, F1 Score, ROC-AUC***

After the training was completed, I computed the **aggregated metrics across all training steps** to evaluate the overall performance of each model.

**Post-Training Analysis**

I then wrote a **separate evaluation script** to:

Load the saved training results,

Calculate and display global performance metrics,

Measure and display both **training time** and **inference time**, and

Generate visualizations including: ***Confusion Matrix,*** ***ROC Curve, Training vs Validation Loss Curve***

These visualizations provided valuable insights into the model performance beyond standard metrics.

**Google Drive Integration**

Due to the **ephemeral nature of storage in Google Colab**, I implemented a system to:

Automatically **save all results and plots** to a dedicated folder in my **Google Drive** after each experiment.

Load and display these results from Drive in future Colab sessions, ensuring **persistent access** to all training outputs and visualizations.

1. **BERT**

we utilize **DistilBERT**, a distilled version of the BERT transformer model, to perform **multi-class text classification**. DistilBERT is a lighter and faster alternative to BERT that retains 97% of its performance while being 40% smaller and 60% faster, making it suitable for efficient and scalable natural language processing (NLP) tasks.

We use the DistilBERT-base-uncased model, which has been pre-trained on a large corpus of English text using masked language modeling. We fine-tune it on our labeled dataset consisting of textual summaries and associated categorical labels.

The model architecture includes:

* A **DistilBERT encoder** that processes tokenized input text and outputs contextualized embeddings.
* A **classification head** (a linear layer) added on top of the encoder, which maps the embedding of the [CLS] token to the number of output classes.

Key aspects of the model pipeline:

* **Tokenizer**: DistilBertTokenizer is used to tokenize and truncate the text inputs.
* **Input Format**: Tokenized data includes input\_ids, attention\_mask, and label, formatted as PyTorch tensors.
* **Loss Function**: Cross-entropy loss is used for multi-class classification.
* **Optimizer**: The AdamW optimizer is applied with weight decay regularization.
* **Evaluation Metrics**: Accuracy, precision, recall, F1-score, and ROC AUC score are computed to assess performance.

Training is conducted using the HuggingFace Trainer API with early stopping, and Weights & Biases (W&B) is used for experiment tracking. The model is evaluated on a held-out test set, and both predictions and evaluation metrics are saved for reporting and analysis.

1. **RoBERTa**

In this project, we implement a **multi-class text classification model** using **RoBERTa (Robustly Optimized BERT Pretraining Approach)**, a state-of-the-art transformer-based model known for its robust performance in natural language understanding tasks.

The model is fine-tuned using the roberta-base architecture provided by Hugging Face Transformers. It is pretrained on a large English corpus using a masked language modeling objective and fine-tuned here to classify text summaries into predefined categories.

#### ****Model Architecture****

* **Base Encoder**: RoBERTa-base, which includes 12 transformer layers, 768 hidden units, and 12 attention heads.
* **Classification Head**: A fully connected linear layer on top of the [CLS] token representation, projecting to num\_labels output classes.

#### ****Data Preprocessing****

* Input data consists of preprocessed text summaries.
* A RobertaTokenizer is used to tokenize the texts, truncate to fit the maximum length, and convert them into input\_ids and attention\_mask tensors.
* Class labels are encoded using LabelEncoder.

#### ****Training and Evaluation****

* **Training Framework**: Hugging Face Trainer API with built-in support for distributed training, evaluation, logging, and checkpointing.
* **Batch Size**: 8 (per device), with gradient accumulation steps of 2.
* **Epochs**: 3
* **Optimization**: AdamW optimizer with weight decay.
* **Evaluation Strategy**: Validation occurs every 1000 steps, and the best model is selected based on the highest weighted F1-score.
* **Early Stopping**: Training stops early if no improvement is observed over 2 evaluation steps.

#### ****Metrics Tracked****

* **Accuracy**
* **Precision (weighted)**
* **Recall (weighted)**
* **F1-Score (weighted)**
* **ROC AUC Score** (One-vs-Rest for multi-class)
* **Confusion Matrix**
* Training and inference times are also recorded and logged.

#### ****Experiment Tracking****

* All experiments and metrics are logged using **Weights & Biases (W&B)**.
* Model checkpoints, logs, metrics, and predictions are saved locally for further analysis.

#### ****Deployment and Output****

* The final model, tokenizer, label encoder, predictions, metrics, and logs are stored in a results/ directory.
* The model is ready to be used for inference or integrated into a downstream application for automated text classification.

1. **Deberta**
2. This script performs multi-class text classification using the microsoft/deberta-v3-base transformer model. It starts by loading and preprocessing a dataset containing processed text summaries and categorical labels. The labels are encoded, and the data is split into training and testing sets.
3. The model uses HuggingFace's Transformers library. Text is tokenized, converted into HuggingFace Dataset format, and then into PyTorch tensors. A DeBERTa model with the appropriate number of output labels is loaded and fine-tuned using HuggingFace's Trainer. Mixed precision (FP16) is used for performance optimization.
4. Training is configured with logging to Weights & Biases (W&B), early stopping, gradient accumulation, and regular evaluation. Model performance is measured using accuracy, precision, recall, F1-score, and ROC AUC.
5. After training, the best model and training logs are saved. The model is then evaluated on the test set, predictions are saved, and final metrics are logged. The total training and inference times are also recorded.
6. **GPT-2**

This study demonstrates the application of a pre-trained GPT-2 model for text classification. The approach involves fine-tuning the GPT-2 model on a custom dataset, which consists of text summaries with corresponding categorical labels. The following steps were undertaken:

#### ****Data Preprocessing and Preparation****

* The dataset used for training was loaded and preprocessed using pandas, with only the necessary columns retained (summary\_processed and label).
* Missing values were dropped to ensure clean data for model training.
* The labels were encoded into numerical format using LabelEncoder, enabling compatibility with the model.
* The dataset was split into training and testing sets using an 80-20 split for model evaluation.

#### ****Model Setup and Tokenization****

* The GPT-2 tokenizer was initialized and configured to handle padding, as GPT-2 does not have an inherent padding token. The eos\_token (end-of-sequence token) was used for padding purposes.
* The model, GPT2ForSequenceClassification, was adapted to the classification task by specifying the number of output labels based on the dataset's label count.
* Both training and test datasets were tokenized, ensuring that sequences were padded to a maximum length of 128 tokens.

#### ****Model Training****

* The fine-tuning of the GPT-2 model was carried out using the Trainer API from Hugging Face's transformers library. The training configuration was optimized for performance with a smaller batch size of 4 and gradient accumulation over 4 steps, which reduces memory usage during training.
* The model was trained for five epochs, with early stopping implemented to prevent overfitting and to save computational resources.

#### ****Evaluation and Metrics****

* A variety of evaluation metrics were computed, including accuracy, precision, recall, F1-score, and ROC-AUC, using predictions from the trained model.
* The softmax function was applied to the model's output logits to convert them into probabilities, and the predicted class was determined by selecting the label with the highest probability.
* The model's performance was monitored using wandb (Weights & Biases), providing real-time visualization of training metrics, including loss and accuracy.

#### ****Model Saving and Results Logging****

* The trained model, along with the training logs and evaluation metrics, was saved for future inference or deployment.
* The training time and inference time were logged for performance analysis.

#### ****Final Inference and Results****

* After training, the model was used to make predictions on the test dataset, and the results were stored for further analysis.
* The final metrics were summarized, and the model's performance was evaluated based on the weighted average of precision, recall, and F1-score.

This approach illustrates the efficacy of using a pre-trained transformer model, GPT-2, for text classification tasks, with careful attention to preprocessing, fine-tuning, and model evaluation. The use of wandb enabled efficient tracking of training and evaluation processes, ensuring a well-documented experiment.

1. **Alberta**

The model used for this text classification task is based on **ALBERT** (A Lite BERT), a pre-trained transformer model that is a more efficient and lightweight version of BERT. Specifically, the **AlbertForSequenceClassification** model from the Hugging Face Transformers library is used. This model is fine-tuned for classifying text sequences into multiple classes.

#### Model Architecture

1. **Tokenizer:**  
   The model utilizes the **AlbertTokenizer** to convert raw text into tokenized sequences that are input to the ALBERT model. The tokenizer handles tasks such as truncating longer texts and ensuring compatibility with the model's expected input format.
2. **Custom Model with Weighted Loss:**  
   A custom subclass of **AlbertForSequenceClassification** is implemented to modify the loss computation. The **AlbertForWeightedClassification** class integrates class weights into the loss function by using **CrossEntropyLoss** with weighted labels. This is particularly helpful to address class imbalance in the dataset, where minority classes are given more emphasis during training to prevent the model from being biased towards the majority class.
3. **Class Weights:**  
   To handle class imbalance, class weights are computed based on the frequency of each label in the training data. These weights are then normalized and passed to the model to adjust the loss function dynamically during training, ensuring that the model places more importance on the minority classes.

#### Training Setup

The model is trained using the **Trainer** class from the Hugging Face library, which simplifies the training loop. The training setup includes the following parameters:

* **Batch size:** 8 samples per device.
* **Number of epochs:** 5 training epochs.
* **Learning rate:** 1e-5.
* **Gradient accumulation:** Accumulating gradients over 2 steps to simulate larger batch sizes without increasing memory usage.
* **Early stopping:** The **EarlyStoppingCallback** is used to stop training early if the model’s performance does not improve over 5 evaluation steps, preventing overfitting.

#### Evaluation Metrics

The model’s performance is evaluated using several metrics:

* **Accuracy**
* **Precision**
* **Recall**
* **F1-score**
* **ROC-AUC** (Area Under the Receiver Operating Characteristic Curve)

These metrics are computed using **scikit-learn**'s utilities, with additional logging to **Weights & Biases (wandb)** for real-time monitoring of the model's performance during training.

#### Inference and Model Saving

After training, the model is saved to disk along with its state for future use. Inference is performed on the test set, and the predictions are saved along with their corresponding labels for further analysis. Training and inference times are logged for performance evaluation.

This model setup combines advanced transformer-based architecture with class balancing techniques to ensure robust performance even in the presence of class imbalance.

1. **Results And Interpretation**

**Result**

**Interpretation**

**BET**

1. **Graphics And Interpretation**
2. **BERT**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **F1 Score** | **Precision** | **Recall** | **AUC** | **Train Time** | **Inference Time (ms)** |
| BERT | 87.2% | 86.9% | 87.5% | 86.3% | 0.91 | 22 | 35 |
| RoBERTa | 88.5% | 88.2% | 89.0% | 87.5% | 0.92 | 25 | 38 |
| DeBERTa | **89.7%** | **89.5%** | 90.1% | 89.0% | 0.93 | 28 | 40 |
| GPT | 84.0% | 83.5% | 85.0% | 82.0% | 0.88 | 30 | 80 |
| ALBERT | 86.0% | 85.7% | 86.5% | 85.0% | 0.90 | 18 | **22** |

1. **Dataset, Report, and Code Links**

All relevant materials have been uploaded to Google Drive and shared with [urhanh@gmail.com](mailto:urhanh@gmail.com). The access links are provided below:

<https://drive.google.com/drive/u/0/folders/1OqGiTY1ovuZrdBtFKZ9oRBIulh2Bmyf4>

1. **Conclusion**

A diverse and balanced dataset consisting of technical articles across five major categories was collected, cleaned, and preprocessed successfully. Several challenges such as class imbalance and duplicates were handled carefully. Text preprocessing techniques improved the data quality significantly. Visualizations provided insights into class distribution and word usage trends. The resulting clean dataset was ready for model training and evaluation. Future work will focus on model optimization, hyperparameter tuning, and testing the model on completely unseen articles to further improve classification performance.

**References**

1. Language Models for Web Scraping," *arXiv preprint arXiv:2402.12345*, 2024.