Text Classification Report

1. Classification Approach

In this project, I fine-tuned a pre-trained large language model (LLM)—specifically, "MoritzLaurer/deberta-v3-large-zeroshot-v1", a zero-shot learning model—to adapt it to the given dataset (training.csv). The fine-tuning process involved training the model on labeled examples to improve its performance on the target classification task. This approach leverages the advantages of a pre-trained transformer-based model, enabling it to generalize well with limited training data while effectively capturing contextual relationships within the text.

2. Preprocessing and Fine-Tuning Steps

To ensure data quality and model efficiency, I implemented the following preprocessing and fine-tuning steps:

Data Preprocessing

- **Lowercasing:** Converted all text to lowercase to maintain uniformity and reduce variations in word representation.
- **Feature Engineering:** Combined the “title” and “abstract” into a single large column, “text” column, for simplicity and to provide more context for classification.
- **Handling unbalanced Class problem:** Oversampled all classes to match the largest category to mitigate the impact of unequal class distribution. (reference1)
- **Data Augmentation:** Applied synonym replacement and random word shuffling on half of the samples in each class to introduce variability.
- **Stratified Data Split:** Used an 80-20 stratified split to maintain equal class distribution in both training and validation sets. The final dataset consisted of 748 training samples and 188 validation samples.

Fine-Tuning Process

- **Epoch Selection:** Experimented with different epoch values to ensure the model was fully trained without overfitting.
- **Batch Size:** Due to computational constraints, a batch size of 2 was used for fine-tuning, which ensured stable training while optimizing available resources.
- **Gradient Accumulation:** Implemented gradient accumulation with 4 steps to simulate a larger batch size (like we set batch size = 8) for more stable gradient updates without increasing memory usage or avoiding OOM errors

3. Conclusion

In this project, I fine-tuned the **"MoritzLaurer/deberta-v3-large-zeroshot-v1"** model for text classification while implementing various data preprocessing and fine-tuning techniques to enhance performance.

To mitigate class imbalance, I applied oversampling and data augmentation techniques, ensuring a more balanced dataset. Additionally, I utilized gradient accumulation to simulate a larger batch size and prevent memory-related issues. Lastly, Hyperparameter tuning was conducted to find the optimal number of epochs and batch size while avoiding overfitting.

As a result of these efforts, the final model achieved an accuracy of **0.913** on the validation dataset.

4. Future Improvements

To further improve model performance in the future, I would explore the following strategies:

- **Data Enlargement & Augmentation:** Increase dataset size by fetching more labeled data or applying advanced data augmentation techniques, such as back translation and sentence paraphrasing, to improve generalization. Also, I might try different Boosted oversampling methods like SMOTE or ADASYN to address class imbalance.
- **Experimenting with Other LLMs:** Evaluate various transformer-based models (e.g., BERT, RoBERTa, T5) to compare classification performance and determine which one is the most effective architecture.
- **Alternative Machine Learning Methods:** Assess the feasibility of traditional classifiers, such as Multi-label Logistic Regression, SVM, and Random Forest, to see if simpler models can achieve comparable or better results with lower computational costs.
- **Boosted Resampling Techniques:** Implement Bootstrap Aggregating method to reduce variance and improve robustness. This method can also be used to compare performance across different models.
- **Ensemble Learning:** Explore Stacking and Voting Classifiers to combine predictions from multiple models, enhancing classification accuracy. (Like Boosted tree combines multiple small trees to get better prediction)
- **Hyperparameter Tuning:** Conduct research for how to reach the optimal hyperparameters (e.g., learning rate, dropout rate, batch size) to fine-tune model performance and prevent overfitting.

Reference 1

```
Before Oversampling:
category
Methods of Building Qubits                        0.461538
Models of Manipulating Qubits for Computation     0.256410
Applications of Quantum Computing                 0.193294
Address Obstacles to Quantum Computation          0.088757
Name: proportion, dtype: float64

After Oversampling:
category
Models of Manipulating Qubits for Computation     0.25
Methods of Building Qubits                        0.25
Applications of Quantum Computing                 0.25
Address Obstacles to Quantum Computation          0.25
Name: proportion, dtype: float64
```