MLP Team Project2 - LLM Classification Finetuning

Team 6

20206582 Hyogeon Park

20216159 Seyeon Kim

20200009 Dongyoung Kim

This project was carried out through a Kaggle competition called LLM Classification Finetuning. The competition aims to predict which response users prefer and provides a dialogue dataset. Each data consists of a user question (prompt) and two LLM responses (response_1 and response_2), and the train set also includes a more human-preferred response (label).

The baseline model was implemented by learning with Logistic Regression using text length-based features. As input features, length information of Prompt, response_a, response_b, and response_b, and the length difference between the two responses, and the ratio of response length were used. For learning, K-Fold Cross Validation (n=5) was applied to eliminate the imbalance of the data. And the final result was generated by ensemble (average calculating) the prediction result for each fold.

After that, the embedding was performed using the all-MiniLM-L6-v2 model of the Sentence-Transformers library to reflect the semantic information of the sentence. The two sentences were composed in the form of "prompt + [SEP] + response", and the embedding vectors were extracted for each response_a and response_b. And the difference between the two vectors was combined to generate the final input vector. Furthermore, we tried to improve the learning performance by adding the length information (bias) of the text to the embedded learning data to improve the final performance. LightGBM model was used, and like the baseline, cross-validation was performed using K-Fold Cross Validation (n=5). The performance of this model was the best, and it was also used as a key model in the final ensemble.

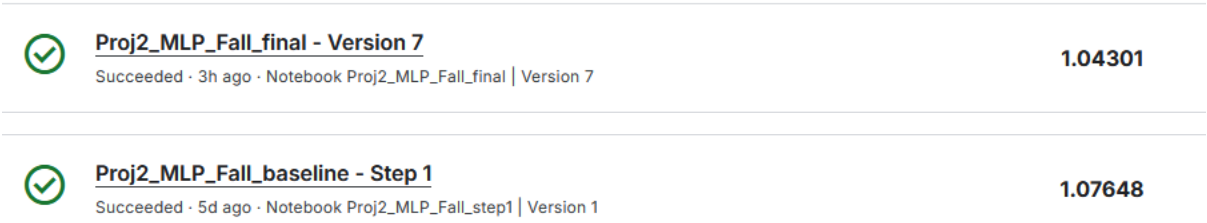
The second extended model is finetuning with the LoRA technique applied to DeBERTa. The input was made by connecting the training data in the form of "prompt + response_a + response_b" to create a single sentence sequence and tokenized it with the model's tokenizer. And by applying the LoRA technique, only small additional modules were learned to perform finetuning quickly and efficiently. In this experiment,

hyperparameters such as r=8, lora_alpha=16, and lora_dropout=0.05 were used, and learning was conducted for 2 epoch using a trainer.

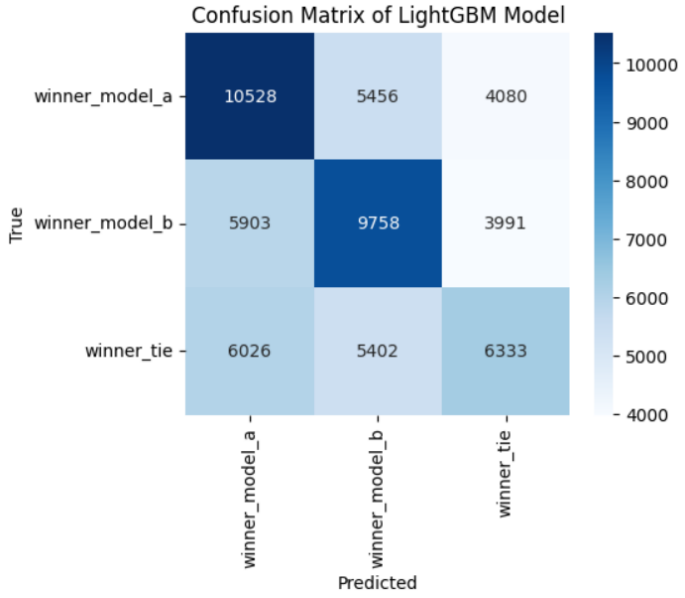
All models were trained and verified using the K-Fold Cross Validation (n=5) strategy to reduce data bias and increase generalization performance. The final model was composed of an ensemble model that combines the prediction results of the three models proposed above: the baseline model, the Bias+LightGBM model, and the DeBERTa-v3-small+LoRA model. At this time, based on the log loss value of each model, a weight-based average calculation was used so that the model with good performance could have more influence on the final result. The performance evaluation was based on the Log Loss index based on Cross Entropy, and the greatest weight (0.5) was given to the LightGBM model with the best performance. In addition, weights of 0.3 were applied to the DeBERTa model and 0.2 to the baseline model.

The following are the score table and screenshots when the baseline model and the final model are submitted to Kaggle. It can be seen that the final model improved by about 0.033 compared to the baseline based on log loss, but the execution time slightly increased.

Model	Score	Running time(s)
Baseline Model	1.07648	32.8
Final Model	1.04301	5350.3s



In order to understand the prediction trends and limitations of the model, error and bias analysis were performed by calculating the Log Loss value for each class for the Bias + LightGBM model that showed the best performance. The following are Confusion Matrix which visualize the prediction results of the LightGBM model and Performance Indicators by Class.



클래스	Log Loss	Precision	Recall	F1-score
winner_model_a	0.9914	0.47	0.52	0.50
winner_model_b	1.0113	0.47	0.50	0.48
Winner_tie	1.1246	0.44	0.36	0.39

It can be said that the winner_model_a and winner_model_b classes show relatively similar performance because the data distribution is uniform and the Precision, Recall, and F1 scores are similar. On the other hand, the Recall value of the winner_tie class was significantly low and the Log Loss was relatively high, so there was a tendency to incorrectly predict the data to be classified as a tie as one of the other two classes. This means that the model has difficulty in recognizing a case where the two responses are meaningfully similar as "tie". An example of the model's incorrect prediction is as follows. The two responses to the question "explain function calling. How could you call a function?" were seemingly different, but contained meaningfully the same content. However, the model tended to predict as winner_moel_b rather than classifying it as a tie by focusing on the surface difference of the text. These results show that the model is more sensitive to morphological differences than semantic similarity.

The overall acuity was about 46%, which did not show high performance. In particular, the winner_tie class showed low prediction performance because the ability to judge semantic similarity is limited. To solve this problem, we think that augmenting the data of the winner_tie class will improve the performance of the model. And we can think of

adding a feature based on semantic similarity so that model can understand semantic similarity well. In addition, various and in-depth learning could not be conducted due to the low performance of Colab and Kaggle and the limited runtime. Therefore, it would be good to proceed with various learning by adjusting hyperparameters such as batch_size and epoch.

The code implementation and test environment was conducted by Colab, and the code submission environment was conducted by Kaggle Notebook. T4 GPU was used by Colab, P100 GPU was used by Kaggle Notebook, and 42 was used as a fixed SEED value for all codes, and it is specified in the code.

Proj2_MLP_Fall_baseline is a code that can submit a baseline model, and you can submit the data of the competition by adding it as an input to Kaggle note without a separate dependency. Proj2_MLP_Fall_final is a code that can submit a final model and requires a dependency to use MiniLM-L6-v2, DeBERTa-v3-small, etc. Data from Competition must be added as input and the data set below must be added.

https://www.kaggle.com/datasets/shinomoriaoshi/sentencetransformersallminilml6v2, https://www.kaggle.com/datasets/miwojc/debertav3small

And you need to create and add a new dataset with folders created using the command "!pip download sentence-transformers peft accelerate datasets -d sheets_dir". This dataset is used for pip local install in the very first block of code.

- **GitHub Link**: https://github.com/CAU-2025-2-MLP-Team6/MLP-proj2-Fall-2025.git