Predicting Human Preferences for LLM Response Enhancement

1. Baseline model (public score: 1.11228)

Data Preprocessing

Before feature engineering, several consistency checks were performed. To verify ID uniqueness, the total count and unique count of the id column were compared, confirming no duplicates existed.

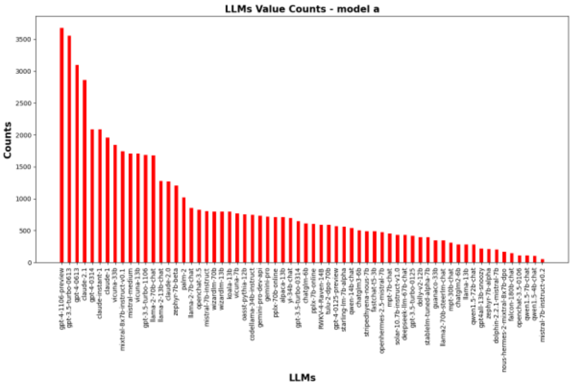
NAN and null value checks

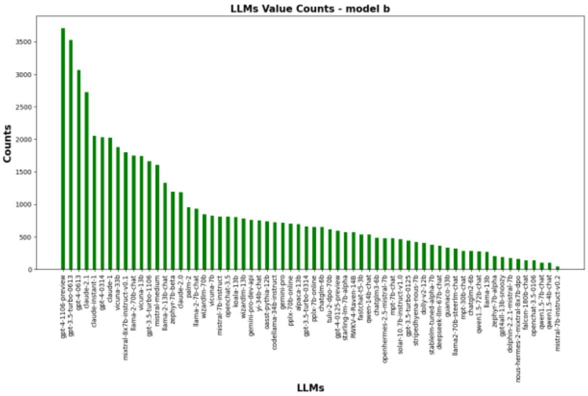
isna() and isnull() were used to check for null or empty cells, confirming neither existed.

A Pipeline block was configured to process the prompt and responses a and b. Functions were written to convert all text to lowercase, remove digits and special characters using regular expressions, and then tokenize the text. Additionally, stop words like ‘a’ and ‘the’, which have no semantic impact on the sentences, were removed.

We consolidated winner a, b, and tie into a single column named winner (a:0, b:1, tie:2), securing the essential column required for training the multiclass classification model.

Subsequently, we visualized the winner data by creating a graph using some of the data. Below is data regarding the LLM types for models a and b. (Red: a, Green: b)





Training

Training method used - logistic regression (public score: 1.11288)

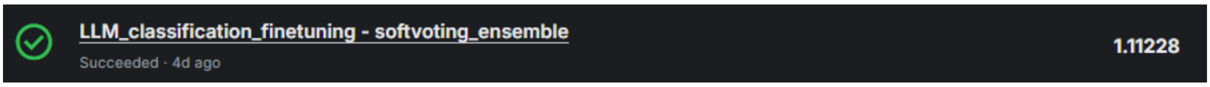
LLM_classification_finetuning - ver2

Succeeded · 8d ago

1.11288

We used logistic regression to vectorize the newly added winner column and tokenized text for training. The scoring result was 1.11288, which fell short of the sample submission score of 1.09861. Recognizing the need for further improvement, we attempted an ensemble by adding several learning methods.

Training methods used: - Softvoting ensemble (logistic regression, k-nearest neighbors, random forest) (public score: 1.11228)


LLM_classification_finetuning - softvoting_ensemble

Succeeded · 4d ago

1.11228

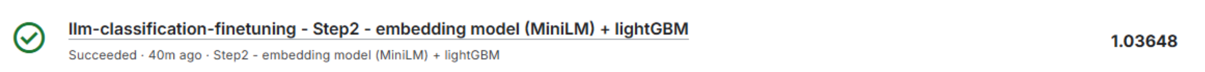
We trained an ensemble model combining logistic regression with kNN and random forest, using the same dataset as for the logistic regression. The scoring result was 1.11228, showing improved performance compared to using logistic regression alone, but still performing poorly relative to the sample.

2. Embedding-based Model - (all-MiniLM-L6-v2 + lightGBM) (public score: 1.03647)

The pre-trained sentence embedding model required for the task was implemented using MiniLM (all-MiniLM-L6-v2). The MiniLM-L6-v2 model maps sentences or paragraphs into a 384-dimensional dense vector space. It offers excellent performance while maintaining a compact parameter size and fast processing speed compared to other models.

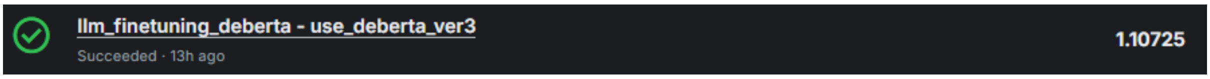
Since Test.csv lacks information about model_a and model_b, only the prompt, response_a, and response_b were used for embedding during training with train.csv. Our goal is to find the more correct response among response_a and response_b, the outputs of models a and b. Therefore, to improve performance in the a/b comparison task, we added (response_a - response_b) and (response_a * response_b) as new features. Therefore, the embedding result is [prompt_emb, resp_a_emb, resp_b_emb, diff_emb (a-b), prod_emb (a*b)], forming a vector with a length of 384 x 5 = 1920.

The classifier used the LGBMClassifier model from the LightGBM library. The distribution of training class labels (y_train) was [a, b, tie] = [0.349, 0.342, 0.309]. The Kaggle competition score for test.csv was 1.03648.

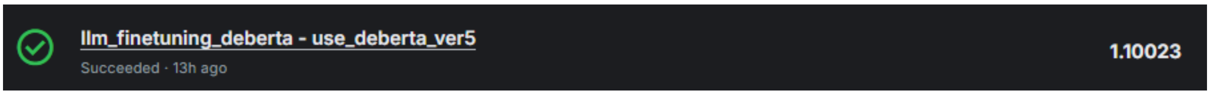


llm-classification-finetuning - Step2 - embedding model (MiniLM) + lightGBM
Succeeded · 40m ago · Step2 - embedding model (MiniLM) + lightGBM
1.03648

3. Model extension - DeBERTa-v3-LoRA (public score: 1.10023)



llm_finetuning_deberta - use_deberta_ver3
Succeeded · 13h ago
1.10725



llm_finetuning_deberta - use_deberta_ver5
Succeeded · 13h ago
1.10023

The initial training using basic methods yielded poor results, so we uploaded a lightweight model fine-tuned for improvement to the Kaggle notebook and proceeded with training. The model used was DeBERTa-v3, chosen for its lightweight nature, decent performance, and accessibility. To circumvent training time constraints, the model's fine-tuning was performed beforehand on a desktop. The fine-tuned lightweight model was then uploaded to the Kaggle Notebook dataset under the name deberta_lora_weights. Additionally, the original model that had not undergone fine-tuning was also uploaded for use in training. For fine-tuning, train.csv was used, and weights and config were loaded from HuggingFace to process the data.

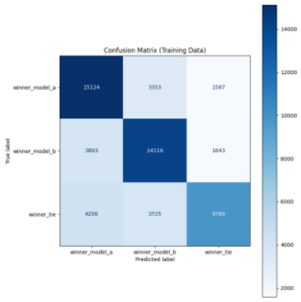
After completing fine-tuning and uploading it to the Kaggle notebook, training was performed using the data processed through the feature engineering from Step 1, with a batch size set to 64. It was confirmed that inference was completed normally through the model, generating the submission.csv file.

However, when attempting to submit this Kaggle notebook for scoring, we encountered a timeout issue that violated the competition's terms. We resolved this by adjusting the batch size. Subsequently, we discovered that GPU acceleration had not been utilized. We enabled GPU acceleration supported by the notebook to reduce inference time, ultimately achieving a public score of 1.10023.

4. Error Analysis

For error analysis, we focused on step 2, which yielded the best Kaggle competition score among steps 1, 2, and 3. First, we performed quantitative analysis by calculating the Per-Class Log Loss and Overall Log Loss, then visually represented the Confusion Matrix using the matplotlib library. Additionally, we calculated precision, recall, and f1-score for each class based on the confusion matrix.

Class \ analysis	log loss	precision	recall	f1-score	True Label data count
winner_model_a	0.8363	0.65	0.75	0.70	20064
winner_model_b	0.8527	0.67	0.72	0.69	19652
winner_tie	0.9432	0.75	0.55	0.64	17761
Total	0.8750	x	x	x	x
Weighted Avg	x	0.69	0.68	0.68	57477



The log loss for winner_tie was the highest at 0.9432, while its recall value was the lowest at 0.55. This indicates that the model struggled with predictions when the actual value was winner_tie. However, winner_tie's precision was 0.75, higher than both winner_model_a and winner_model_b. This indicates the model exhibited a strong bias toward being overly cautious in predicting Tie. When examining accuracy based on the relative length of response_a and response_b, the accuracy when response_a was longer was 0.6779, and the accuracy when response_b was longer was 0.6777. This suggests that the bias due to response length was not significant.

5. Final Model and Results (public score: 0.83038)

The application of machine learning-based techniques to date has yielded minimal performance gains over baseline models. While attempts at using LLM models showed noticeable performance improvements, these gains remained at a certain level. This led to explore methods for achieving maximum performance. In this process, we identified the highest-performing model among publicly available models, focused on reproducing it, and built upon it for improvement. This approach stems from the current understanding that it is insufficient to improve existing models or create new ones based on our current knowledge. Therefore, the goal is to understand and extend the architecture of high-performance models.

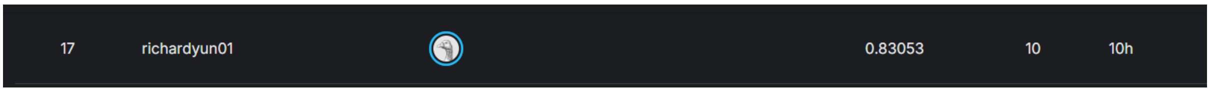
This code consists of a pipeline that combines the inference results of multiple models using a weighted ensemble to generate the final submission file. This involves running Gemma2 and LLaMA3-based models and FAISS, a lightweight custom model utilizing semantic similarity features, to obtain probability outputs for each. Finally, these outputs are combined using a weighted average to produce the result. The Gemma2 and LLaMA3 inference pipelines both use a common tokenizer and dynamic token count-based batching. They apply pipeline parallelization by splitting large models into front and back halves, assigning them to two GPUs, and overlapping the front and back halves in microbatch units. This process utilizes mixed-precision inference to reduce memory usage and computational cost, while employing parameters related to Rotated Position Embedding and Block-Diagonal Causal Masking to minimize data movement and masking overhead between stages. Another pipeline creates cosine similarity features for prompts and responses using Sentence-Transformer embeddings and FAISS inner product search. These are fed as key and value into a custom DeBERTa head's multi-head attention, fusing text representations and auxiliary features via cross-attention before calculating probabilities in the classifier.

In the ensemble stage, probabilities from each model are retrieved. Class order correction is then applied to align the LLaMA3 results with the A/B criteria. A weighted average is calculated using empirically determined weights to generate the final probability. Experiments confirmed that the weight parameters for Gemma2, LLaMA3, and FAISS play the most critical role in determining the model's performance.

The model weights in the original code were [0.7, 0.2, 0.1], and the test result for that model was 0.84012. To improve this, we first assumed that the weights for each model would be equally important, based on the finding from a previous task that weights for each model are crucial when performing ensemble tasks. We then modified these weights and conducted experiments for each case. The scores for each and the best result are as follows.

Intent to Change Weights	Weight Value	Results
Base Code	[0.7, 0.2, 0.1]	0.84012
Gemma2 Only	[1.0, 0.0, 0.0]	0.83890
LLaMA3 Only	[0.0, 1.0, 0.0]	0.84416

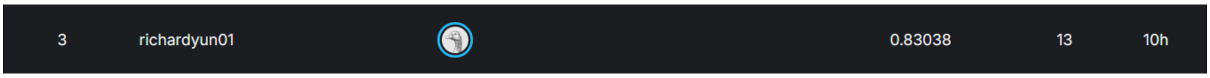
Gemma2 Centric	[0.8, 0.15, 0.05]	0.83661
Gemma2+LLaMA3 Equal	[0.5, 0.5, 0]	**0.83053**
FAISS Contribution Check	[0.7, 0.0, 0.3]	0.87102
FAISS Performance Verification	[0.3, 0.0, 0.7]	0.96759
Equal Weights	[0.33, 0.34, 0.33]	0.87427


17 richardyun01 0.83053 10 10h

The results of this experiment showed that the best performance was achieved when Gemma2 and LLaMA3 were set at the same ratio. The next best results were obtained when Gemma2 was used as the core model and other models were ensemble at a lower ratio. In this experiment, using Gemma2 alone yielded better results than LLaMA3, confirming Gemma2's significant contribution. However, ensemble combinations with specific weighting produced better results than either model alone, indicating that appropriately combining multiple models is more effective. Conversely, when FAISS was given a high weight, the model's performance decreased sharply, confirming that its performance and contribution rate are lower than those of Gemma2 or LLaMA3.

Also, we adjusted the input length limit parameter within the process to determine how the range of truncated tokens affects model performance. We suggested that a lower input length limit increases the number of truncated tokens, potentially causing information loss, while a higher limit reduces truncation but increases memory consumption. We then varied the default value of 4092 by -25%, -12.5%, +12.5%, and +25%.

Intent to Change Parameters	Base Code	-25%	-12.5%	+12.5%	+25%
Maximum Parameter Length	4092	3072	3584	4608	5120
Results	0.83053	0.83093	0.83068	**0.83038**	Timeout


3 richardyun01 0.83038 13 10h

The results of this experiment confirmed that higher values for the input length limit parameter yield better outcomes. While the respective deviations were 0.0004, 0.00015, and 0.00015, effectively negligible differences, a slight performance increase was still observable. Furthermore, consistent with the assumption above, we observed that higher parameter values increased execution time by approximately 5 minutes to over 1 hour. Notably, a 25% increase caused a timeout, confirming that higher values significantly impact performance.

In conclusion, the highest performance was achieved when Gemma2 and LLaMA3 were balanced in the ensemble. Adjusting the ratio with Gemma2 as the core also yielded stable results. Furthermore, increasing the input length limit showed a subtle yet consistent performance improvement, confirming that not only model combinations but also fine-tuning the input processing range contributes to performance enhancement. These results suggest that an approach combining the characteristics of multiple models appropriately and optimizing input parameters is most effective for maximizing the overall system's efficiency and performance, rather than relying on a single model.