Leveraging an LLM Ensemble to Predict User Preferences Between ChatBots

Thomas Lawton, Matthew Ramirez

Abstract

This project aims to predict human user preferences in opposing battles between responses from two established large language models (LLMs). Utilizing data from the LMSYS ChatBot Arena Kaggle Competition (formally titled Chatbot Arena Human Preference Predictions), we employed a model ensemble approach, combining two pre-trained models, Gemma 2 and Llama 3, enhanced with Parameter-Efficient Fine-Tuning (PEFT) in tandem with Low-Rank Adaptation (LoRA). Our methodology included a custom tokenization function and validation testing for weighting the model ensemble. The models' performance was evaluated through the LMSYS test database. This project contributes to the broader field of Reinforcement Learning (RL) through its exploration on the value of human feedback, ultimately providing a framework for improving LLM-to-user interactions via predicting user preferences. The present report accentuates the latent potential of model ensembles to enhance AI-powered conversation systems by predicting these human preferences, paving the way for better user-centric AI applications.

Code -
https://colab.research.google.com/drive/1D5NdBufRLmPHiijUKN9WfTjF94XuHxVk?usp=sharing

Datasets Utilized -

LMSYS:

https://www.kaggle.com/competitions/lmsys-chatbot-arena

Gemma 2 (9b-it-4bit):
https://www.kaggle.com/models/google/gemma-2

Llama 3 (8b-chat-hf):
https://www.kaggle.com/models/metaresearch/llama-3/Transformers/8b-chat-hf/1

Introduction

Within the rapidly evolving field of large language models, predicting human preferences accurately is critical for enhancing user experience in AI interactions. Drawing on a dataset from ChatBot Arena, where users engage with two anonymous LLMs and select the response they personally find more satisfactory, our goal is to develop a model that can anticipate these same user preferences with high accuracy.

To achieve this, we have fine-tuned and applied a combination of state-of-the-art LLMs, integrating several machine learning techniques, including model ensembling and Parameter-Efficient Fine-Tuning (PEFT) using Low-Rank Adaptation (LoRA).

Some key techniques in our approach include the use of a custom tokenization function for efficient data processing and a validation testing framework to optimize model weighting for the final prediction. Additionally, we implemented parallel processing and mixed precision inference to enhance computational efficiency, ensuring that the models operate effectively within the resource and runtime constraints established by the LMSYS Competition.

This work demonstrates the potential of combining multiple pre-trained models with sophisticated fine-tuning and ensemble techniques to improve the accuracy of human preference predictions in AI-generated content.

Methods

In this section, we delve into the methodologies employed in the project. Each subsection will provide an overview of the techniques and processes utilized to illustrate how these methodologies were implemented and how they functioned in tandem with one another.

Initialization and Environment

The notebook was run on the Kaggle Notebook environment using 2 T4 GPUs after implementing the following data sets as an input into the environment, also mentioned above: LMSYS, Gemma 2, and Llama 3. Key libraries such as PyTorch and Hugging Face Transformers were imported as well. The notebook uses a main inference function to process the input data in batches, generating predictions from each model, then combining these predictions using a weighted average based on validation performance to select the final output. The notebook ran in 225 seconds total with the placeholder test set without validation testing and outputs a submission.csv file for evaluation using the LMSYS Kaggle Competition private test set.

Data Preprocessing

The LMSYS dataset used for this project consisted of user prompts and their corresponding responses from two distinct LLMs. A custom function was designed and employed to process the text, removing any extraneous characters and normalizing the input into a common structure fit for tokenization.

Following text standardization, the dataset was tokenized using two distinct tokenizers (with distinct prefixes to align with each tokenizer as seen in *Figure 1*), *GemmaTokenizerFast* for the Gemma model and *AutoTokenizer* from Hugging Face for the Llama model. We utilized *GemmaTokenizerFast* as it is designed to handle the specific tokenization needs of Gemma, but also to match the tokenization process used during the LoRA training.

```
def tokenize(tokenizer, prompt, response_a, response_b,
             max_length=cfg.max_length, spread_max_length=cfg.spread_max_length):
    # Handle different formats for different tokenizers
    if isinstance(tokenizer, GemmaTokenizerFast):
        prompt = ["<prompt>: " + p for p in prompt]
        response_a = ["\n\n<response_a>: " + r_a for r_a in response_a]
        response_b = ["\n\n<response_b>: " + r_b for r_b in response_b]
    else:
        prompt = ["User prompt: " + p for p in prompt]
        response_a = ["\n\nModel A :\n" + r_a for r_a in response_a]
        response_b = ["\n\n--------\n\nModel B:\n" + r_b for r_b in response_b]
```

Figure 1

A "spread maximum length" strategy was implemented in the tokenization function, where the prompt and each response were tokenized separately and combined later, however, this was not used in calculating the predictions.

To optimize the inference process, the tokenized data was sorted according to input length before batching. Grouping inputs of similar lengths together minimized the need for padding, ultimately improving computational efficiency during model processing. These were necessary steps given the runtime constraints placed by the LMSYS Competition.

Model Implementation Using LoRA

Parameter-Efficient Fine-Tuning (PEFT) is a technique allowing for efficient adaptation of large models via modifying only a small subset of their parameters. Low-Rank Adaptation (LoRA) is a specific method within PEFT that injects low-rank matrices into the model's architecture, enabling it to adapt to new tasks with minimal additional parameters. Instead of fine-tuning the entire model, LoRA focuses on key layers, such as the attention layers in transformers, making it both memory efficient and computationally efficient, as seen in *Figure 2* below.

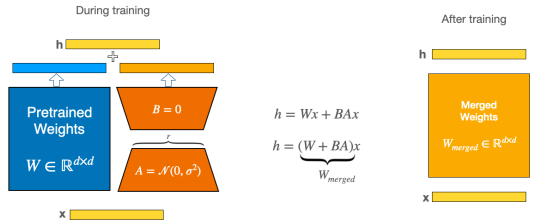

Figure 2

In this project, the Gemma model was fine-tuned using a LoRA checkpoint from another Kaggle user's training notebook. This pre-trained LoRA checkpoint allowed us to adapt the model to our task without the need for extensive re-training.

The Gemma model, integrated with the pre-trained LoRA checkpoint, was loaded using the *Gemma2ForSequenceClassification* class. For the Llama model, we utilized the *LlamaForSequenceClassification*, configuring it to run with 8-bit precision using the BitsAndBytes library to reduce memory usage.

Inference

After configuring and loading the pre-trained models, the next step was to run inference on the dataset to generate predictions. To run inference with the limited runtime and computational resources available, we utilized a combination of batch processing, parallel processing and mixed precision to increase efficiency.

To handle the large dataset efficiently, the input data was processed in batches. Batch processing helps manage memory usage by splitting the dataset into smaller, more feasible pieces, allowing the models to process multiple inputs simultaneously. We used a batch size of 4.

To further optimize the inference process, mixed precision was employed using the *torch.cuda.amp.autocast()* context manager. Mixed precision allows certain operations to be performed in half precision (16-bit floating point) rather than full precision (32-bit floating point), yielding faster computations and reduced memory usage.

We also used parallel processing to further optimize the inference process. The models were run in parallel using 2 GPUs as seen in *Figure 3*. By assigning the different models to separate GPUs, we were able to generate predictions simultaneously, significantly reducing the overall inference time.

```
with ThreadPoolExecutor(max_workers=2) as executor:
    results = executor.map(inference,
                           (gemma_data, llama_data),
                           (gemma_model, llama_model),
                           (gemma_tokenizer, llama_tokenizer),
                           (device_0, device_1))
```

Figure 3

Model Weighting from Validation Testing

To maximize the accuracy of the final predictions, the outputs from the different models were combined using a weighted average approach. This section details how the model weights were determined based on validation performance and how the final ensemble predictions were calculated.

Before determining the model weights, we evaluated each model's performance on a validation set taken from the training data. This involved running the inference process on the validation data and calculating performance metrics for each model. Log loss was used as the evaluation metric for scoring in the LMSYS Competition, therefore to keep consistency, we implemented it for assessing how well each model predicted user preferences as seen in *Figure 4*. The results were used to inform the weighting strategy.

```
# Calculate validation metrics (using log loss)
gemma_log_loss = log_loss(val_data['labels'].tolist(),
val_gemma_result_df[['winner_model_a', 'winner_model_b', 'winner_tie']].values)

llama_log_loss = log_loss(val_data['labels'].tolist(), v
al_llama_result_df[['winner_model_a', 'winner_model_b', 'winner_tie']].values)
```

Figure 4

The weights for combining model outputs were calculated based on the inverse of the log loss from the validation results. Models with lower log loss (i.e., better performance) were given higher weights. The final calculated weights were Gemma: 0.4257644179 and Llama: 0.5742355821.

Conclusion

In this paper, we employed a model ensemble approach to predicting user preferences in head-to-head battles between responses from large language models. The result of the test on the LMSYS private test database was 0.941. This shows that the model could be significantly improved with more fine-tuned training with computational and time resources that we did not have available for this project, as well as by implementing more advanced machine learning techniques.

Related Papers

Ganai, M.A., et al. April 6, 2021. “Ensemble deep learning: A review” arXiv:2104.0239
https://arxiv.org/abs/2104.02395

Hu, Edward J., et al. June 17, 2021. "Lora: Low-rank adaptation of large language models." *arXiv:2106.09685*.
https://arxiv.org/abs/2106.09685

Acknowledgements

The notebook used for this paper was forked from: https://www.kaggle.com/code/nabojyotipandey/dual-model-inference-gemma2-9b-llama3-8b and fine tuned.

LoRA Checkpoint from Gemma Model Training was taken from:

https://www.kaggle.com/datasets/emiz6413/73zap2gx

Below are other Kaggle notebooks in which code was used or inspired from:

https://www.kaggle.com/code/jaejohn/lmsys-combine-gemma-2-9b-llama-3-8b/notebook

https://www.kaggle.com/code/emiz6413/inference-gemma-2-9b-4-bit-qlora

https://www.kaggle.com/code/emiz6413/llama-3-8b-38-faster-inference