

# Natural Language Processing

## Term Project – LLM Classification Finetuning

NTHU 112064528 林澤宇 NTHU 112064535 劉珩 NTHU 113064513 彭智崇

### 1. 摘要

本專題聚焦於 Kaggle 競賽改進大型語言模型(LLM)的回應分類性能，分析來自 Chatbot Arena 的互動資料，並利用 DebertaV3、XGBClassifier 模型和加權平均法等多種方法進行模型微調與優化，通過數據整理、特徵工程(如 TF-IDF 與 Word2Vec)和超參數調整，我們提升了模型的準確率與評分表現。實驗結果顯示，參數調整與特徵工程改進讓原有的表現有所提升，加權平均法結合多模型輸出，能有效降低偏差，並在排行榜中取得很好的成績。未來方向可以探索語義特徵與使用高效預訓練模型(如 Llama3、Gemma2 等)，進一步提升模型效能與任務表現。

### 2. 任務介紹

大型語言模型(LLMs)已成為現代自然語言處理應用的核心，但它們的回應並非總能滿足使用者的偏好，為了縮小模型能力與使用者偏好的差距，本次專題聚焦於 LLM 回應分類任務，目標是預測使用者在 Chatbot Arena 平台上面對兩個匿名模型的回應時，更傾向於選擇哪一個回應，該競賽鼓勵參賽者探索各種機器學習技術，建立準確預測使用者偏好的模型，使用者回應偏好的決策可能受多種偏差影響，例如冗長偏差、位置偏差等等，在本專案中，我們採用了 DebertaV3、XGBClassifier 和加權平均模型等方法，結合競賽提供的 Chatbot Arena 真實數據進行模型微調與優化，目標是提升模型對人類偏好的預測準確性，我們希望透過本次專題與競賽，為 LLM 的互動品質帶來提升與優化。

### 3. 資料分析

本次競賽的數據來自 Chatbot Arena，包含用戶與兩個大型語言模型之間的互動數據。資料集中評使用者基於提示詞(Prompt)與模型回應，選擇出更為滿意的回應作為標籤(Label)。

- **數據概述**

- (1) **訓練資料(Training data)**：包含約 55,000 筆互動數據，每筆記錄包含提示詞(Prompts)、兩個模型的回應(response\_a、response\_b)以及評審的選擇結果(winner\_model\_a、winner\_model\_b 與 winner\_tie)。
- (2) **測試資料(Testing data)**：包含約 25,000 筆互動數據，與訓練資料相似，但不包含標籤。
- (3) **提供檔案**：train.csv 為訓練數據，test.csv 測試繳交數據，sample\_submission.csv 提交格式範例。

- **探索性數據分析(Exploratory Data Analysis, EDA)**

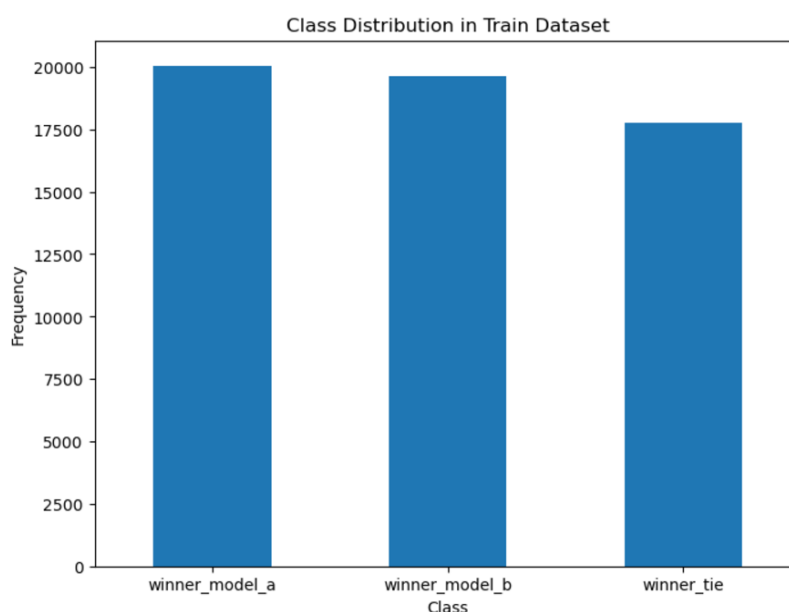
- (1) **訓練資料資訊**：從資料中能看出總共有 57,477 筆資料，每筆資料共有 9 項資訊，分別為 id, model\_a, model\_b, prompt, response\_a, response\_b, winner\_model\_a, winner\_model\_b, winner\_tie。

```
Train Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57477 entries, 0 to 57476
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   id                    57477 non-null  int64
1   model_a               57477 non-null  object
2   model_b               57477 non-null  object
3   prompt                57477 non-null  object
4   response_a            57477 non-null  object
5   response_b            57477 non-null  object
6   winner_model_a        57477 non-null  int64
7   winner_model_b        57477 non-null  int64
8   winner_tie            57477 non-null  int64
dtypes: int64(4), object(5)
memory usage: 3.9+ MB
```

- (2) **缺失值檢查**：從資料中可以看出本次競賽附上的訓練資料無缺失資料。

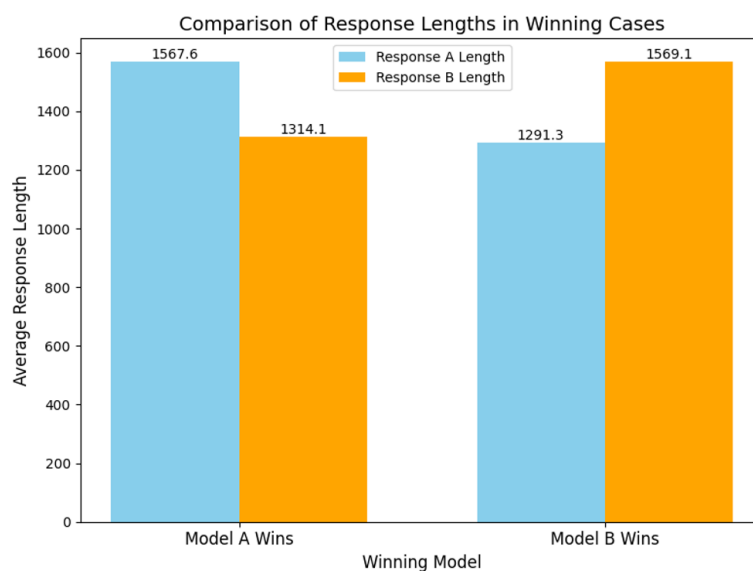
```
Missing Values in Train Dataset:
id                0
model_a           0
model_b           0
prompt            0
response_a        0
response_b        0
winner_model_a    0
winner_model_b    0
winner_tie        0
prompt_length     0
response_a_length 0
response_b_length 0
dtype: int64
```

**(3) 分類標籤分布：**從資料中可以看出訓練資料的標籤分布算平均，每項標籤類別數量均在 17,500 ~ 20,000 間。



通過分析訓練集資訊為模型的數據預處理與特徵工程提供關鍵信息，有助於提升分類性能與模型穩定性。

**(4) 模型回答分析：**分析模型回答後發現，當模型回答的長度較長時更容易被選為較好的回應。



通過可視化圖型可以看出當 Model A 獲勝時 A 的回應是比較長的，同樣當 Model B 獲勝時，B 的回應也是比較長的，能夠看出回答的好壞可能跟回應的長度有所關係。

## 4. 實作方法

### ● DebertaV3

(1) **模型選擇**：在模型的選擇上，我們採用了微軟提出的 DeBERTaV3 預訓練模型(預設為"deberta\_v3\_extra\_small\_en")，該模型基於改進的 Transformer 架構實現，具有以下特點：優異的語義理解能力、高效的運算性能以及靈活的微調策略，DeBERTaV3 支持逐步解凍骨幹層的微調策略，使得模型能更好地適應特定任務需求，同時避免過度更新預訓練權重，且利用改進的自注意力機制，在文本上下文語義捕捉方面表現出色。

### (2) 參數調整 (Parameter Tuning)：

**階段 1**：凍結骨幹層，在初始訓練階段，我們凍結了 DeBERTaV3 的所有骨幹層，僅對分類頭進行訓練，使用較高的學習率(1e-3)，快速適配特定任務並穩定分類頭輸出。

**階段 2**：部分解凍骨幹層，解凍 DeBERTaV3 的最後兩層骨幹層，對語義特徵進行細粒度調整，降低學習率至 5e-6，以更穩定地更新部分參數。

**階段 3**：完全解凍骨幹層，允許模型全參數更新，充分利用預訓練特徵，學習率進一步降低至 1e-6，以避免過擬合。

```
# 定義模型架構
inputs = {
    "token_ids": keras.Input(shape=(2, None), dtype=tf.int32, name="token_ids"),
    "padding_mask": keras.Input(shape=(2, None), dtype=tf.int32, name="padding_mask"),
}
backbone = keras_nlp.models.DebertaV3Backbone.from_preset(PRESET)
embed_a = backbone({k: v[:, 0, :] for k, v in inputs.items()})
embed_b = backbone({k: v[:, 1, :] for k, v in inputs.items()})
embeds = keras.layers.Concatenate(axis=-1)([embed_a, embed_b])
embeds = keras.layers.GlobalAveragePooling1D()(embeds)
outputs = keras.layers.Dense(3, activation="softmax", name="classifier")(embeds)
model = keras.Model(inputs, outputs)

# 凍結 Backbone 層，僅訓練分類頭
backbone.trainable = False
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss=keras.losses.CategoricalCrossentropy(label_smoothing=0.02),
    metrics=["accuracy"],
)
```

```

# 訓練分類頭
model.fit(train_ds, validation_data=valid_ds, epochs=4, callbacks=[
    ModelCheckpoint("best_model_stage1.keras", save_best_only=True, monitor="val_loss"),
])

# 部分解凍 Backbone，進行第二階段訓練
for layer in backbone.layers[-2:]:
    layer.trainable = True
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=5e-6),
    loss=keras.losses.CategoricalCrossentropy(label_smoothing=0.02),
    metrics=["accuracy"],
)
model.fit(train_ds, validation_data=valid_ds, epochs=1, callbacks=[
    ModelCheckpoint("best_model_stage2.keras", save_best_only=True, monitor="val_loss"),
])

# 完全解凍 Backbone，進行第三階段訓練
backbone.trainable = True
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-6),
    loss=keras.losses.CategoricalCrossentropy(label_smoothing=0.02),
    metrics=["accuracy"],
)
model.fit(train_ds, validation_data=valid_ds, epochs=4, callbacks=[
    ModelCheckpoint("best_model_stage3.keras", save_best_only=True, monitor="val_loss"),
])

```

**(4) 特徵工程(Feature Engineering)：**在特徵工程部分，我們結合了數據處理與語義提取，構建了模型的核心輸入。首先，在數據處理環節，我們對文本進行了清理，包括移除非 ASCII 字符並執行 Unicode 轉換，同時將標籤轉換為數字類別，便於後續模型訓練。此外，我們創建了一個新列，將 Prompt 和 Response 結合作為模型的輸入文本，提供更加完整的上下文信息。

在特徵預處理階段，我們使用了 DeBERTaV3 的預處理器(DebertaV3 Preprocessor)，設定序列長度為 512，對文本進行了標準化的分詞與編碼處理，從而為後續的模型訓練打下了良好的基礎。這種數據處理與特徵提取方法有效地提升了文本表示的質量，使得模型能夠更準確地捕捉文本的語義特徵。

```

# 將標籤列轉換為數字類別
label_map = {"winner_model_a": 0, "winner_model_b": 1, "winner_tie": 2}
df["class_label"] = df[["winner_model_a", "winner_model_b", "winner_tie"]].idxmax(axis=1).map(label_map)

```

```

# 將 prompt 與 response 配對
def make_pairs(row):
    return [
        f"Prompt: {row.prompt}\nResponse: {row.response_a}",
        f"Prompt: {row.prompt}\nResponse: {row.response_b}",
    ]

df["options"] = df.apply(make_pairs, axis=1)
test_df["options"] = test_df.apply(make_pairs, axis=1)
df = df[df["options"].map(lambda x: all(len(opt.strip()) > 0 for opt in x))]

# 分割訓練數據為訓練集和驗證集
train_df, valid_df = train_test_split(df, test_size=0.2, stratify=df["class_label"])

# 加載 DebertaV3 模型的預處理器
PRESET = "deberta_v3_extra_small_en"
preprocessor = keras_nlp.models.DebertaV3Preprocessor.from_preset(PRESET, sequence_length=512)

```

## ● XGBClassifier

在本專題中，我們也使用 XGBClassifier 作為核心模型之一，其結合了高效的計算能力與強大的預測性能，適用於大部分的多分類問題，以下為詳細實作方式：

- (1) **模型選擇**：在模型上的選擇我們使用了 XGBoost 中的 XGBClassifier 模型，該模型基於 Gradient Boosting Tree 演算法實現，具有幾個特點，包含支援 GPU 的平行化運算有效提升運算效率、能有效處理缺失值提高模型穩定性等等。
- (2) **參數調整(Parameter Tuning)**：我們採用 RandomizedSearchCV 隨機搜索最佳超參數組合，能夠在與設定好的子參數數值中隨機選擇搭配並得到對應結果，減少計算時間並提升模型表現，搜索的參數包括樹的深度(max\_depth)、樹的數量(n\_estimators)、學習率(learning\_rate)、子樣本比例(subsample)以及其他正則化係數等，最後得出最佳化參數。

```
param_dist = {
    'n_estimators': [200, 210, 220, 230, 240, 260, 280],
    'max_depth': [5, 6, 8, 9],
    'learning_rate': [0.015, 0.02, 0.025, 0.03, 0.05],
    'subsample': [0.5, 0.55, 0.6, 0.65, 0.7],
    'colsample_bytree': [0.65, 0.7, 0.75, 0.8],
    'gamma': [0, 0.1, 0.2, 0.3],
    'reg_alpha': [0, 0.01, 0.1, 1],
    'reg_lambda': [2, 2.5, 3, 3.5]
}
```

```
Best Parameters found:
{'subsample': 0.65, 'reg_lambda': 3, 'reg_alpha': 0.01, 'n_estimators': 280, 'max_depth': 9, 'learning_rate': 0.03, 'gamma': 0.3, 'colsample_bytree': 0.7}
Best Log Loss: 1.0367653021728511
```

- (3) **特徵工程(Feature Engineering)**：我們不直接使用已經預訓練好的 Embedding 模型，而是依據訓練資料重新訓練 Word2Vec 模型將詞語轉換為向量表示，我們認為從競賽提供的訓練資料自行訓練 Embedding 模型能夠更有效的捕捉文本語義關係與上下文資訊，並且可以依據需求調整訓練模型實的參數，如向量維度、epoch 等等，最後我們決定的訓練參數與結果如下圖。

```
# Train Word2Vec model
w2v_model = Word2Vec(sentences, vector_size=300, window=5, min_count=1, workers=4, epochs=10)
```

```
Generating sentence embeddings for training data...
Feature matrix shape: (57477, 900)

Generating sentence embeddings for test data...
Test feature matrix shape: (3, 900)
```

## ● Weighted Average Method

除了上述兩種方法，我們也嘗試了基於 LightGBM 庫的 LGBMClassifier 分類模型，並結合由其他參賽者預訓練的 Gamma2-9b 及 Llama3-8b 兩個 LLM，簡化訓練時的硬體配置，同時能達到高準確度的預測。

- (1) **模型集成**：在此方法中，我們主要專注於訓練 LGBMClassifier，透過不同提取不同維度的特徵用於訓練。Gamma2-9b 及 Llama3-8b 是原作者用 Full-Parameters Tuning 訓練在 4 顆 A100 80G 上，總共分三個階段訓練，並且加入了競賽外部 Dataset(240k pseudo labels)一起訓練，作者有提到，他參考了 Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena [\[4\]](#)，考慮到 token length 有限的情況下，他只用了 prompt - response\_a - response\_b 這種 sequence 作為輸入格式，該篇論文有提到，可以透過交換位置解決數學問題的有限評分能力，但不確定作者是否是為了解決數學問題的位置偏差問題，在第三階段的 Finetune 中，他將餵給 Llama3-8b 的 response\_a - response\_b 改為 response\_b - response\_a，leaderboard score 也從 0.873 降到 0.869，最終以 2:1 的權重分配給 Finetune 好的 Gamma2-9b 跟 Llama3-8b，分數達到 0.83101。
- (2) **特徵工程**：我們在訓練 LGBMClassifier 的過程中，針對 prompt 跟 response 分別用了詞級跟字級級的 TF-IDF 跟 CountVectorizer 來提取文本特徵，增加細粒度的語言模式分析，並設定 n-gram 來捕捉短語和上下文依賴性，同時，我們對 prompt, response\_a, response\_b 最多提取 500 個詞特徵，並忽略極少出現的詞，另外，我們也引入額外特徵分析計算特定字符出現次數(如換行、空格、句號等)，提取結構性特徵，補足語義特徵中無法識別的信息，我們也將額外特徵做變換(平方根與對數)，避免極端值的影響，最後將所有特徵合併。



```

tfidf_prompt = TfidfVectorizer(
    max_features=500,
    stop_words='english',
    min_df=0.002,
    ngram_range=(1, 3)
)
count_prompt = CountVectorizer(
    max_features=500,
    stop_words='english',
    min_df=0.002,
    ngram_range=(1, 3)
)

tfidf_prompt_char = TfidfVectorizer(
    analyzer='char',
    ngram_range=(1, 3),
    max_features=500,
    min_df=0.002
)
count_prompt_char = CountVectorizer(
    analyzer='char',
    ngram_range=(1, 3),
    max_features=500,
    min_df=0.002
)

tfidf_response = TfidfVectorizer(
    max_features=1000,
    stop_words='english',
    min_df=0.002,
    ngram_range=(1, 3)
)
count_response = CountVectorizer(
    max_features=1000,
    stop_words='english',
    min_df=0.002,
    ngram_range=(1, 3)
)

tfidf_response_char = TfidfVectorizer(
    analyzer='char',
    ngram_range=(1, 3),
    max_features=1000,
    min_df=0.002
)
count_response_char = CountVectorizer(
    analyzer='char',
    ngram_range=(1, 3),
    max_features=1000,
    min_df=0.002
)

extras = []
EXTRAS = ['\n', '\n\n', '.', ' ', '"', ',']
for e in EXTRAS:
    for c in ['prompt', 'response_a', 'response_b']:
        extras.append(df[c].str.count(e).values)

extras.append(df['prompt'].str.len().values)
extras.append(df['prompt'].str.split().apply(lambda x: len(x)).values)

extras = np.stack(extras, axis=1)
extras = np.hstack([extras ** 0.5, np.log1p(extras)])

```

```
final_features = hstack([response, extras, X_prompt_combined])
```

## 5. 實驗與結果

在本專案中，我們針對 DebertaV3、XGBClassifier 和加權平均法(Weighted Averaging Method)等模型進行實驗與性能評估，以下為主要實驗結果：

### ● DebertaV3

```

2874/2874 ————— 2639s 799ms/step - accuracy: 0.4775 - loss: 1.0327 - val_accuracy: 0.4718 - val_loss: 1.0368
Epoch 2/4
2874/2874 ————— 2021s 703ms/step - accuracy: 0.4816 - loss: 1.0275 - val_accuracy: 0.4762 - val_loss: 1.0354
Epoch 3/4
2874/2874 ————— 2024s 704ms/step - accuracy: 0.4867 - loss: 1.0220 - val_accuracy: 0.4765 - val_loss: 1.0331
Epoch 4/4
2874/2874 ————— 2033s 707ms/step - accuracy: 0.4897 - loss: 1.0187 - val_accuracy: 0.4790 - val_loss: 1.0310

```

Accuracy 是訓練集上的準確率，Val\_accuracy 是驗證集上的準確率。



Loss 是訓練集上的損失(CrossEntropyLoss)。

Val\_loss 是驗證集上的損失(CrossEntropyLoss)與 Log loss 是相同的。



$$CrossEntropyLoss = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \cdot \log(\hat{y}_{ij})$$

最後 Leaderboard 分數以及 Log loss:

34	team48		1.02899	7	1mo
 Your Best Entry! Your most recent submission scored 1.02899, which is an improvement of your previous score of 1.03474. Great job!					
<a href="#">Tweet this</a>					



## ● XGBClassifier

在訓練時的評估指標使用平均與 OOF(Out-Of-Fold) Log loss 觀察訓練完後模型的效果，由結果可以發現 Log loss 數值接落在 1.035 左右，表明模型在多分類任務中的預測性能具有一定穩定性，預測概率接近真實標籤的分佈，並且平均與 OOF 的分數相當接近，顯示出交叉驗證中的穩定性，模型在不同資料分折下的表現波動較小，未出現明顯的過擬合或欠擬合現象。最後在排行榜上的分數為 1.02907，雖然排名沒有非常靠前，但我們認為其穩定性為後續改進提供了良好基礎。

Average Log Loss across all folds: 1.0351167876752485  
OOF Log Loss: 1.0351169523514439

```
# Save submission file
submission.to_csv('submission.csv', index=False)
# print("\nSubmission file created successfully!")
print(submission.head())
```

	id	winner_model_a	winner_model_b	winner_tie
0	136060	0.153348	0.475892	0.370760
1	211333	0.378870	0.302541	0.318589
2	1233961	0.245354	0.503474	0.251172

40	Zhi-Chong Peng		1.02907
 Your Best Entry! Your submission scored , which is not an improvement of your previous score. Keep trying!			

## ● Weighted Average Method

該方法主要分為兩個階段。第一階段中我們使用 k-fold cross validation 訓練 LGBMClassifier，總共進行 10 個 folds 的訓練，並賦予分類器 max\_depth, num\_leaves, min\_child\_samples 等參數，控制模型的複雜度。每一次訓練結束後計算當前的 log loss，並將每一個 fold 訓練完的模型及權重視為獨立模型及權重存起來，權重則取 log loss 的倒數，越小損失代表模型表現越好，對應權重越高。並在 10 個 fold 都完成訓練後計算平均 log loss，可以發現目前在訓練集上

的 loss 已經達到大約 1.03，在測試集的表現上 Leaderboard score 已經可以達到 1.01347。

```
model = LGBMClassifier(  
    objective='multiclass',  
    num_class=3,  
    learning_rate=0.05,  
    n_estimators=1000,  
    max_depth=7,  
    num_leaves=127,  
    min_child_samples=50,  
    random_state=random_state,  
    n_jobs=-1  
)
```

Fold 10 Log Loss: 1.0394382946487262

Average 10-Fold Log Loss: 1.0313123315221486



LLM Classification Finetuning - (LGBMClassifier) - Version 1

Succeeded · 21h ago · Notebook LLM Classification Finetuning - (LGBMClassifier) | Version 1

1.01347

第二階段中，有鑒於 Gamma2-9b 及 Llama3-8b 的高效推論和優秀表現，我們並未對這兩個 LLM 多做訓練，而是引入前一階段中產生的模型及權重加入推論，因為此階段並無訓練部分，大幅縮減了運算需求，唯一調整的僅有 Gamma2-9b, Llama3-8b 跟 LGBMClassifier 的推論結果的權重比例，在多次嘗試後，最終在 Leaderboard score 的表現達到 0.83084，位居第二。

2	henryliu513436		0.83084	16	15h
Your Best Entry! Your most recent submission scored 0.83084, which is the same as your previous score. Keep trying!					

## (1) 模型性能與改進方法比較：

- **DebertaV3** 模型作為核心架構，其基於預訓練的 Transformer 模型進行微調，模型訓練過程採用了逐步凍結策略，在多輪訓練中穩步降低了 Log Loss，從完全未調整之原始程式碼提高了約 0.026，最終排行榜分數為 1.02899。未來可以嘗試的做法，可以去往前處理的方式去做，我覺得可以多加一些特徵，像是回答的長度，加入這個特徵去訓練我認為效果應該會更好，但由於太晚發現，以及訓練時長的關係，並沒有實測出來，後續我們也使用了其他的模型方法，效果更加優秀，可以在 Leaderboard 達到非常靠前排名。

- **XGBClassifier** 模型使用自行訓練的 Word2Vec 特徵進行訓練，並通過 RandomizedSearchCV 調整其模型參數，最終排行榜分數為 1.02907，相較完全未做調整之原始程式碼[1]提高了 0.035 左右，並且提高了模型的穩定性。未來可以嘗試的方向包括進一步優化文本嵌入方法(如融合多種文本特徵等)，或在成本許可下探索更多的超參數組合提升學習能力，以及將 XGBClassifier 與其他基於神經網絡的模型結合，以多模型的方式降低 Log Loss 等等，都是我們認為可以持續嘗試的方向。
- **加權平均方法**中，雖然訓練 LGBMClassifier 時礙於硬體限制，並未捕捉更深層的語意特徵，而是用 TF-IDF 及詞頻等較淺層，但細粒度較高的語意特徵來訓練，同時，訓練過程中透過控制分類樹的數量及葉子節點的參數，能夠有效將 LGBMClassifier 從原始程式[3]的表現 1.01426 降為 1.01347。而我們認為 Gamma2-9b 及 Llama3-8b 已經具有成熟的深度語意處理能力，因此加入推論階段，簡化訓練負擔，同時保留良好表現，從預訓練 Gamma2-9b, Llama3-8b 的原始表現 0.83101（原始程式已關閉）下降至 0.83084。然而，在最好的表現中可以發現，即便是三個模型突論結果的加權平均，在這項任務中更仰賴的是具有深度語意的預測結果，Gamma2-9b, Llama3-8b, LGBMClassifier 三個模型的推論結果權重分別是 50:42:0.5。或許之後在訓練 LGBMClassifier 時，能夠嘗試將淺層語意特徵產生的高維稀疏矩陣降維，並適時加入深層語意特徵（例如其他組別提到的 prompt 與 response\_a, response\_b 之間的 similarity）訓練，亦或嘗試加入 data augmentation 改變 response 位置，相信都能更進一步降低 log loss。

```
preds = np.average([
    np.load("prob_m0.npy"),
    np.load("prob_m3.npy")[:, [1, 0, 2]],
    y_test_proba
],
axis=0,
weights=[50, 42, 0.5],
)
```

## 6. 結論與未來方向

即使加權平均法已經有不錯的準確度，我們仍然從針對訓練集的預測結果中挑選 log loss 最高的 10% 資料，可以發現，在這些資料的預測結果中，模型的預測結果相當平均，但更傾向預測 model\_a 跟 model\_b。

	id	winner_model_a	winner_model_b	winner_tie	log_loss
count	5.748000e+03	5748.000000	5748.000000	5748.000000	5748.000000
mean	2.149849e+09	0.353102	0.352275	0.294623	1.628678
std	1.240471e+09	0.201077	0.202906	0.162232	0.336623
min	6.508900e+04	0.004011	0.004318	0.028887	1.309457
25%	1.099694e+09	0.192261	0.195052	0.198826	1.405587
50%	2.149037e+09	0.295699	0.285117	0.243203	1.528133
75%	3.212990e+09	0.514138	0.510530	0.328133	1.736539
max	4.294947e+09	0.963568	0.948729	0.954344	4.886958

經過進一步的分析後，我們觀察到預測機率最高的選項為 winner\_model\_a 或 winner\_model\_b 的資料中(pred\_mean)，與另一個沒被預測到的 model 的 response(opp\_mean)相比，雖然在情感描述上(polarity, subjectivity)相差並不多，但在長度上具有顯著的差異，這也很符合的資料集中原本就存在的冗長偏差，加上 pred\_mean 跟 opp\_mean 在 entity\_count 也存在一些差距，顯然我們訓練好的模型仍然更偏好描述詳細，甚至列舉實例的回答。

feature_diff			
	pred_mean	opp_mean	diff_pct
char_count	2045.355203	1478.459048	38.343717
word_count	299.383880	220.300456	35.897985
sentence_count	20.913752	15.409298	35.721637
avg_word_length	6.996730	7.007639	-0.155679
avg_sentence_length	17.147921	16.950051	1.167373
polarity	0.116024	0.114045	1.734979
subjectivity	0.469362	0.474716	-1.127711
noun_count	79.384097	57.455790	38.165532
verb_count	37.713013	28.077558	34.317283
adj_count	28.430371	20.661091	37.603440
entity_count	13.445579	9.595698	40.120899

儘管加權平均法已經得力於 pretrained LLM 對深度語義的成熟處理技巧，但還是會受限於資料集原本的困境，如果在資料預處理時先將所有 response padding 到同樣長度，並在特徵提取時加入 Sentence Embeddings 獲取句子提語意特徵，而不限於詞級與字符級特徵，或許是個不錯的方向，而 Gamma2-9b 跟 Llama3-8b 由於已經經過大量資料(此任務訓練集+外部資料集)訓練，進步空間甚微，只能嘗試優化 LGBMClassifier，或在硬體效能許可下，替換成較輕量的 LLM。

## 7. 參考資料

- [1] Vishnu Priya. (2024). *LLM Classification Finetuning (XGB)*. Kaggle.  
<https://www.kaggle.com/code/vishnupriyagarige/llm-classification-finetuning-xgb>
- [2] Addison Howard. (2024). *LMSYS: KerasNLP Starter*. Kaggle.  
<https://www.kaggle.com/code/addisonhoward/lmsys-kerasnlp-starter>
- [3] Filtered. (2024). *LLM Classification Finetuning - (LGBMClassifier)*. Kaggle.  
<https://www.kaggle.com/code/huanligong/llm-classification-finetuning-lgbmclassifier>
- [4] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhonghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. “*Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.*”, *arXiv preprint arXiv: 2306.05685*, 2023.