自然語言處理
2024 Term Project
LLM Classification Finetuning

第 87 組

組員:
國立政治大學 資管四 李宥萱
國立陽明交通大學 資工三 吳定霖
國立陽明交通大學 資工三 吳宗樺

一、專案介紹

1. 任務介紹

此任務目的是預測使用者更偏好哪一個大型語言模型所回答的答案，幫助縮短 LLM 能力與人類偏好之間的差距。且與 Reinforcement Learning from Human Feedback (RLHF)的概念密切相關，將不僅是根據固定的規則學習，也通過人類的指導來理解更複雜的部分，使得機器能夠更貼近人類的期望和行為方式，將有助於改善聊天機器人與人類的互動。

2. 資料集介紹

資料集由 ChatBot Arena 的使用者互動資料組成。在每次使用者互動中，會向兩個不同的大型語言模型提供一個或多個prompt，然後指出哪個模型給出的回應更令人滿意。訓練資料包含 55,000 筆資料，而測試集約為 25,000 筆。

欄位	描述
id	A unique identifier for the row.
model_[a/b]	The identity of model_[a/b]. Included in train.csv but not test.csv.
prompt	The prompt that was given as an input (to both models).
response_[a/b]	The response from model_[a/b] to the given prompt.
winner_model_[a/b/tie]	Binary columns marking the judge's selection. The ground truth target column.

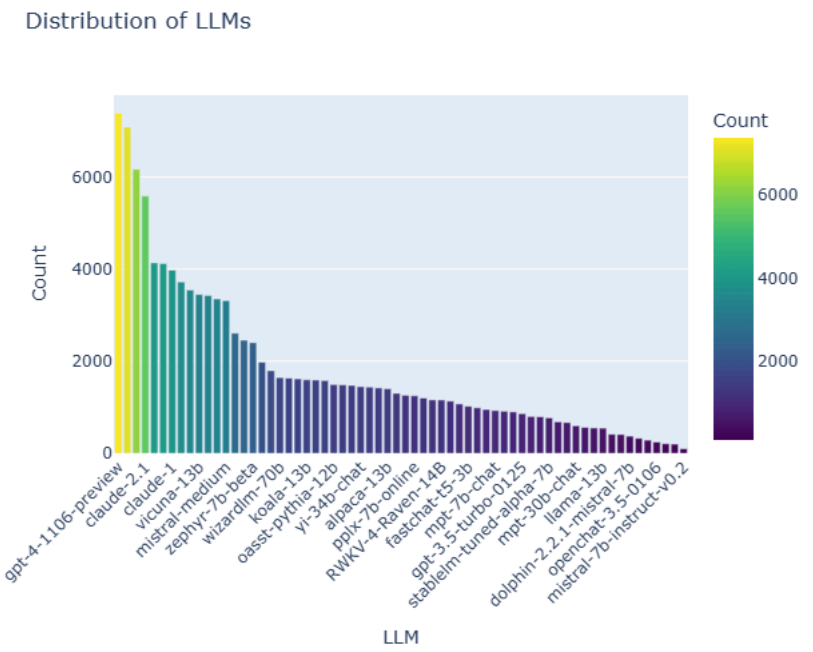
3. 輸入/輸出/評估方式

- 輸入: prompt, response_[a/b]
- 輸出: a 模型回應較好的機率、b 模型回應較好的機率、兩模型回應一樣好的機率 (三個機率值總和為 1)
- 評估方式: 三個預測機率值與正確答案的 Log Loss
  - N : 樣本的數量
  - y_i : 真實標籤
  - p_i : 該樣本屬於positive class的預測機率

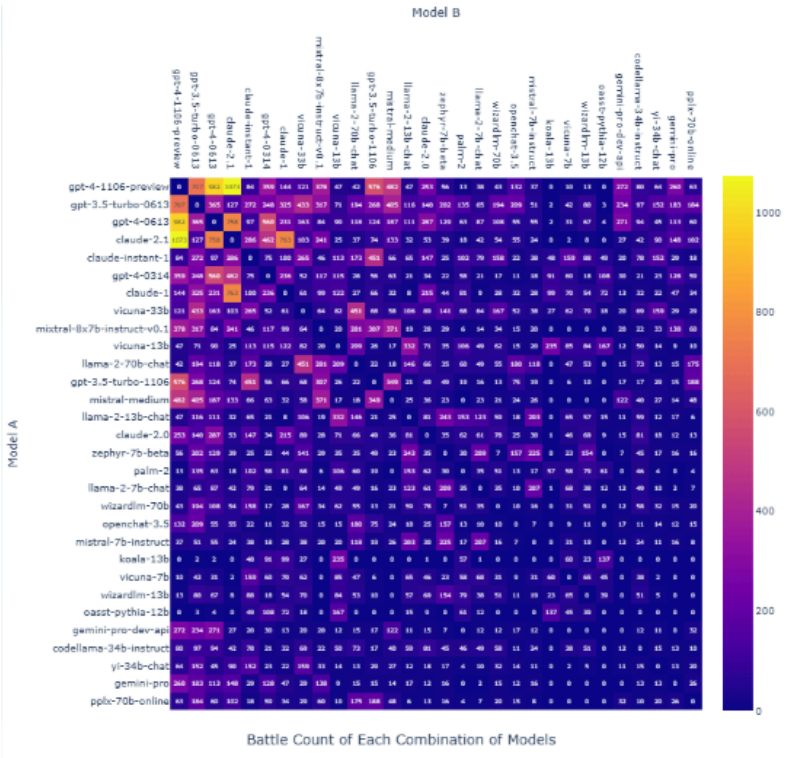
$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

二、資料分析(EDA)

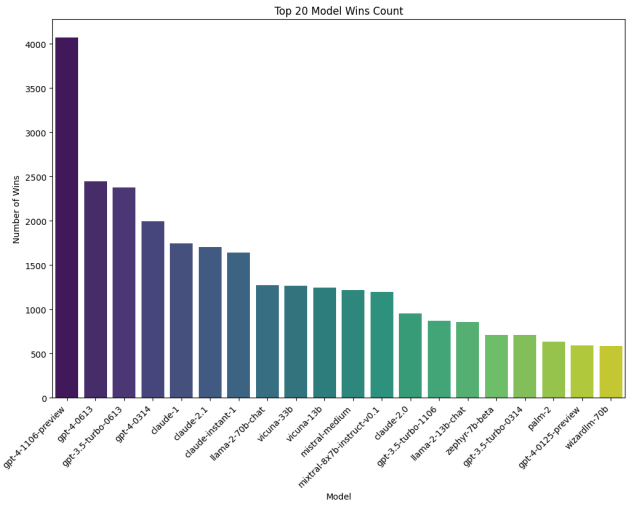
- 共64種模型，其中最常被做比較的模型前三名分別為gpt-4-1106-preview, claude-2.1, claude-1。



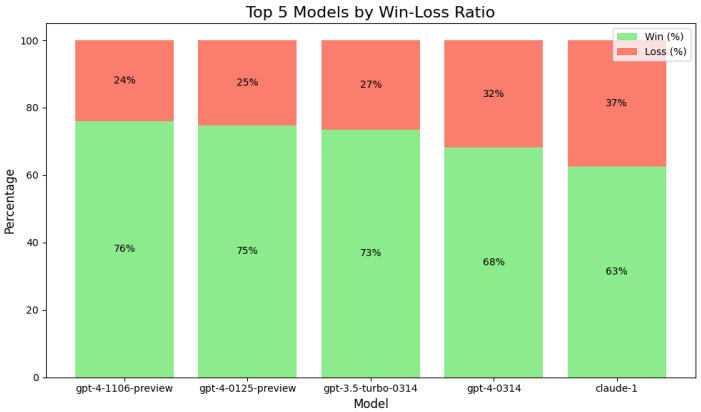
- 最常被兩兩比較的模型
  - gpt-4-1106-preview V.S. claude-2.1
  - gpt-4-1106-preview V.S. gpt-4-0613
  - claude-1 V.S. claude-2.1



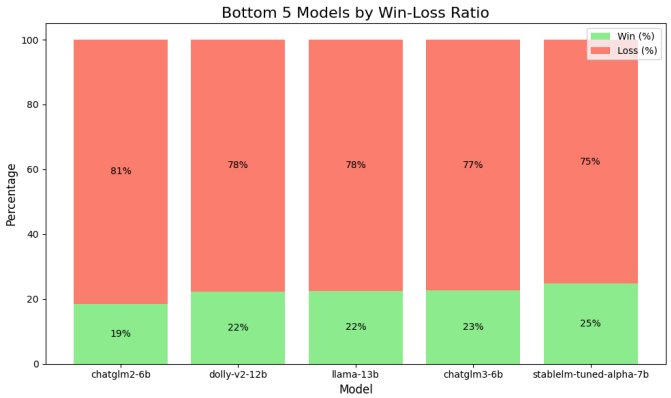
- 模型赢的次数最多的前三名：gpt-4-1106-preview, gpt-4-0613, gpt-3.5-turbo-0613



- 模型win-loss比率最高的前五名：gpt-4-1106-preview, gpt-4-0125-preview, gpt-3.5-turbo-0314, gpt-4-0314, claude-1



- 模型win-loss比率最低的前五名：chatglm2-6b, dolly-v2-12b, llama-13b, chatglm3-6b, stablelm-tuned-alpha-7b



三、研究方法

1. 問題困難點
    - 數據特徵差異與異質性：不同語言模型在回答問題方面可能有各自的特色，使得response的語言特徵可能非常多樣。尤其是不同的prompt類型（如問題回答、翻譯、文本生成等）會引發模型給出不同風格的回應。
    - 模型對長文本的處理能力：Prompt和response的長度對於某些模型而言可能太長，導致長文本中的關鍵信息可能被模型忽略，造成較差的成效。
    - 多類別分類：必須同時準確預測三類結果的概率分佈（a模型回應較好的機率、b模型回應較好的機率、兩模型回應一樣好的機率）。
    - 模型泛化能力：模型需要對未曾見過的prompt和response進行預測，因此模型須有一定的泛化能力以應對大量未知的資料特徵。

2. 我們提出的研究方法
    - 資料前處理
        - 文本預處理
            - 使用 NLTK 將文本分詞
            - 移除停用詞與特殊字符
            - 詞形還原

```python
# Preprocess Data
def preprocess_text(text):
    #convert text to lower case
    text = text.lower()
    #remove digits and special characters using regular expressions
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    #tokenize the text
    text = nltk.word_tokenize(text)

    return text

# Remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    text_no_stopwords = [word for word in text if word not in stop_words]

    return text_no_stopwords

def lemmatization(text):
    lemmatizer = nltk.WordNetLemmatizer()
    lemmatizer_text = [lemmatizer.lemmatize(text) for text in text]

    return lemmatizer_text
```

        - 資料格式化

- 建立Prompt-Response配對
- 將winner_model_[a/b/tie] 轉換為數字標籤

- 模型設計
  - DeBERTa-v3 small
    - max_length = 512, epochs = 1, batch_size = 16, lr = 2e-5
    - 各自的隱層輸出經過平均池化（mean-pooling）。將兩者拼接後，通過一層全連接層進行分類。最終使用 softmax 將 logits 轉化為概率分布。

```
# Configuration
class CFG:
    seed = 42
    model_name = "./deberta-v3-small/"
    max_length = 512
    epochs = 3
    batch_size = 16
    lr = 2e-5
    label2name = {0: 'winner_model_a', 1: 'winner_model_b', 2: 'winner_tie'}
    name2label = {v: k for k, v in label2name.items()}
```

- 訓練與驗證
  - 損失函數與優化器
    - CrossEntropyLoss 作為分類任務的損失函數。
    - 使用 AdamW 並透過線性學習率調度器控制學習率。

```
criterion = nn.CrossEntropyLoss()
optimizer = AdamW(model.parameters(), lr=CFG.lr)
scheduler = get_scheduler("linear", optimizer, num_warmup_steps=0, num_training_steps=len(train_loader) * CFG.epochs)
```

  - 訓練流程
    - 使用 torch.cuda.amp 提高混合精度訓練效率。
    - 前向傳播：計算 logits 和損失。
    - 反向傳播：通過梯度縮放 (amp.GradScaler) 避免數值不穩定。
    - 優化器更新權重。

```python
scaler = amp.GradScaler()  # Initialize GradScaler for mixed precision

for epoch in range(CFG.epochs):
    model.train()
    train_loss = 0
    loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{CFG.epochs}", leave=True)  # TQDM progress bar

    for batch in loop:
        inputs = {k: v.squeeze(1).to(device) for k, v in batch.items() if k != "labels" and k != "token_type_ids"}
        labels = batch["labels"].to(device)

        optimizer.zero_grad()
        with amp.autocast():  # Enable mixed precision
            outputs = model(**inputs)
            loss = criterion(outputs, labels)

        # Backpropagation with mixed precision scaling
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        train_loss += loss.item()

        # Update TQDM progress bar
        loop.set_postfix(loss=loss.item())

    print(f"Epoch {epoch+1}, Train Loss: {train_loss / len(train_loader):.4f}")
```

- 推理過程
  - 應用 softmax，將 logits 轉換為 [0, 1] 範圍的概率。

```python
with torch.no_grad():
    loop = tqdm(test_loader, desc="Prediction", leave=True)  # TQDM progress bar
    for batch in loop:
        inputs = {k: v.squeeze(1).to(device) for k, v in batch.items() if k != "token_type_ids"}
        with amp.autocast():  # Enable mixed precision during inference
            outputs = model(**inputs)
        # 使用 softmax 將 logits 轉換為概率
        batch_probs = torch.softmax(outputs, dim=-1).cpu().tolist()
        probs.extend(batch_probs)
```

3. 問題困難點與提出方法間的關聯
   - 數據特徵差異與異質性：透過使用 NLTK 分詞、移除停用詞和特殊字符，將有效減少雜訊並強調關鍵語義特徵。而詞形還原處理減少語義冗餘，提升特徵表達的一致性。
   - 模型對長文本的處理能力：DeBERTa-v3 small模型支援512tokens，可以盡量使完整上下文不丟失，提升模型的預測精準度，適合處理長文本。而透過平均池化（mean-pooling）可以確保在隱層輸出中均衡提取長文本的重要信息，減少丟失細節。
   - 多類別分類：CrossEntropyLoss是專為多類別分類問題設計的損失函數，能有效評估預測分佈和真實分佈的偏差，確保模型在概率估計上更準確。此外，CrossEntropyLoss的設計與Log Loss的評估方式一致，都強調預測分佈與真實分佈的匹配，能直接優化模型在Log Loss評估中的表現。

- 模型泛化能力：DeBERTa透過Disentangled Attention將內容嵌入 (content embeddings) 和位置嵌入 (position embeddings) 分開建模，讓模型能夠更精準地處理句子結構和上下文關係，特別是在需要捕捉多層次語義的情境中。並使用Replaced Token Detection，以捕捉語言的「自然性」，如流暢度和語義合理性，將在處理語言特徵時更具泛化能力，更貼近人類偏好。

四、研究成果

模型輸出loss後直接應用softmax	inference的時候再應用softmax	epoch	score
X	O	1	1.04219
X	O	3	1.07467
O	O	1	1.04398
O (log_softmax)	O	1	1.07435
O	X	1	1.35201
O	X	3	1.58984

如果在計算損失前應用了 softmax，會導致兩個問題：

- 數值不穩定：當 logits 的值非常大或非常小時，指數函數容易導致溢出或下溢。
- 冗餘計算：CrossEntropyLoss 內部會再一次計算 log(softmax)，重複操作可能會降低效率。

五、其他額外嘗試

1. 針對Llama fine tune，嘗試直接微調語言模型，可能在情感、文意理解能力較強。
   - 模型 : Llama 3.2 1b
     - 使用這個模型的理由是，可以使用較少的計算資源，就實現出大型語言模型的微調
   - Fine tune : Lora Fine tune

```python
# LoRA Configuration
lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    inference_mode=False,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1
)

```

- 超參數設定：

```python
class CFG:
    seed = 42
    model_name = "unsloth/Llama-3.2-1B"
    max_length = 512
    epochs = 1
    batch_size = 2
    lr = 2e-5
    label2name = {0: 'winner_model_a', 1: 'winner_model_b', 2: 'winner_tie'}
    name2label = {v: k for k, v in label2name.items()}

```

- 實驗:
  - 測試對於大型語言模型微調來說，是否加入系統提示詞 (system prompt) 會對於大型語言模型的性能提升。
  - 我們測試了不加入提示詞與加入提示此兩個實驗組與對照組。

```python
system_prompt = """
There is a prompt and its response.
Please tell me whether the human likes the response of this prompt.

"""
```

- 結果：
  - 沒有提示詞 : loss = 1.7
  - 有提示詞 : loss = 1.65
  - 可以看到，有無提示詞對於這個任務來說沒有太多的幫助。

2. 實作 Direct Preference Optimization 的 paper (https://arxiv.org/abs/2305.18290)
   - 核心概念
     - 偏好建模：DPO 將偏好學習轉化為一種概率建模問題，通過最大化用戶偏好樣本的對數似然來優化模型。
     - 去強化學習：與傳統基於強化學習的方法（例如 RLHF）不同，DPO 無需訓練額外的 reward model，直接在生成模型的基礎上進行調整，減少複雜性。

        - 數學框架：DPO 以偏好樣本為基礎，計算兩個回應在模型中的概率，調整模型使得用戶偏好的回應概率更高。

```python
def dpo_loss(outputs, labels, beta=1.0):
    logits_a, logits_b, logits_tie = outputs[:, 0], outputs[:, 1], outputs[:, 2]

    # Pairwise difference for logits
    diff = logits_a - logits_b
    exp_diff = torch.exp(beta * diff)

    # Compute loss for each case
    loss = torch.where(
        labels == 0,  # Prefer A
        -torch.log(exp_diff / (1 + exp_diff)),
        torch.where(
            labels == 1,  # Prefer B
            -torch.log(1 / (1 + exp_diff)),
            -torch.log(torch.softmax(torch.stack([logits_a, logits_b, logits_tie], dim=-1), dim=-1)[:, 2])  # Tie
        )
    )
    return loss.mean()
```

    - 如何提升模型的用戶偏好預測能力
        - 直接優化偏好目標：通過對比用戶偏好數據中的回應對，DPO 能夠更精確地捕捉用戶偏好的細微差異。
        - 減少過擬合風險：由於不需要單獨的回報模型，DPO 在學習過程中避免了額外的 noise and biases。

六、探討其他組別的作法

- 第 105 組
    - 一樣使用DeBERTa-v3 small作為預訓練模型，但在classifier的部分有使用Layer Normalization提高模型收斂速度，並在反向傳播時透過梯度剪裁（Gradient Clipping），防止梯度爆炸。
- 第 54 組
    - 比較多種DeBERTa和ALBERT模型，其中DeBERTa-v3_base_en的成效最好。
- 第 51 組
    - 使用TfidfVectorizer和CountVectorizer作為特徵表示，並透過LightGBM模型來評估特徵的重要性，以進一步使用Wrapper方法來進行特徵選擇。
    - 使用XGBoost模型進行訓練，並使用early stopping避免過度擬合。
- 第 83 組
    - 透過以下三個資料前處理作為特徵表示
        - prompt + response_a/b的embedding
        - embedding之間的attention score

        - TF-IDF, word counts
    - 藉由交換model_a和model_b(response_a和response_b)去實現data augmentation

七、後續研究方向

- 透過增強數據多樣性、分割特徵差異解決overfitting的問題。
- 結合其他LLM，預期能有效綜合兩個模型的優勢，減少單一模型的偏誤。
- 採用稀疏化技術（如 Sparse Fine-tuning），僅調整模型中的部分參數或模塊（例如 LoRA 方法），以降低資源需求，並減少模型過擬合的風險。
- 引入多維度的評估標準（例如語言流暢度、語義相關性和用戶滿意度），避免過於依賴單一指標，提升模型對真實偏好的準確度，而非僅基於 Log Loss。