Sample-Efficient Human Evaluation of LLMs via MAD Competition

Quick Start

Setup

我們在本项目中使用`python 3.10.9`。你可利用这个命令创建一个虚拟环境

```
conda create -n YOUR_ENV_NAME python=3.10.9 -y
```

然后，我们需要安装`requirements.txt`中的全部python库，务必保证版本的正确性

```
pip install -r requirements.txt
```

Usage

我们的方法包含以下6个步骤：

1. 从instruction seeds开始，通过instruction evolution method生成新的指令，即Instruction Pool；
2. 选择多个模型，收集模型在instruction pool中的回复；
3. 计算两两模型针对同一指令生成的回复的相似性；
4. 用过MAD competition选择Top-K个指令
5. 人工偏好标注
6. 使用Elo Rating System进行排名

如果你想跳过步骤1~2，利用现成的数据进行探索，那么可参考我们在`Chatbot Arena conversations`数据的实验过程，跳转至step 3

step 1: Instruction Evolution

我们可以通过如下命令，运行`instruction_evol.py`，进行指令生成。以`Writing` scenario为例：

```
# The avaliable scenario: Understanding, Reasoning, Writing, Coding
dataset_name=Writing
# The LLM that generate new instruction
model=gpt-4-1106-preview  # avaliable model: gpt-3.5 or gpt-4
output_path=./data/instruction/${dataset_name}.jsonl
max_tokens=2048
temperature=0.7
top_p=0.9
# The number of evolution iterations
iter=1
```

```
export OPENAI_API_KEY='Your OpenAI API KEY'
python instruction_evol.py \
  --dataset_name ${dataset_name} \
  --output_path ${output_path} \
  --model ${model} \
  --max_tokens ${max_tokens} \
  --temperature ${temperature} \
  --top_p ${top_p} \
  --iter ${iter} \
  --api_batch 200 \
```

我们可以直接通过修改./scripts/instruction_evol.sh然后运行，生成指令：

```
bash ./scripts/instruction_evol.sh
```

数据将保存在./data/instruction目录下。

step 2: Model Inference

对于API类型的模型，可以通过修改vllm_api_infernece.sh中的命令进行模型推理，以Writing scenario为例：

```
# The inference model, default model used in paper:
MODEL_NAME=gpt-3.5-turbo-1106
MAX_TOKENS=2048
DEV_SET=Writing    # Reasoning,Writing,Understanding,Coding,Chatbot_Arena
GEN_OUTPUT_PATH=./outputs/inference/

# For gpt-3.5-turbo and gpt-4-turbo
export OPENAI_API_KEY='Your OpenAI API KEY'
# For gemini-pro
export GOOGLE_API_KEY='Your GOOGLE API KEY'

python vllm_api_inference.py \
    --model_name ${MODEL_NAME} \
    --max_tokens ${MAX_TOKENS} \
    --temperature 0.0 \
    --output_file_name ${GEN_OUTPUT_PATH} \
    --dev_set ${DEV_SET} \
    --sample_num -1 \
    --api_batch 100 \
```

代码支持OpenAI各系列聊天模型、Gemini-Pro、和基于vLLM API的本地部署模型.

对于本地部署的非API模型，我们采用vLLM框架进行推理，详见vllm_inference.sh。
综上，我们运行如下命令：

```
bash ./scripts/vllm_api_inference.sh
```

或者

```
bash ./scripts/vllm_inference.sh
```

step 3: Similarity measurement

我们使用了3个相似度指标：GPT-4、text-embedding-ada-002 (OpenAI)、Bert-Score。我们通过如下命令设置评估相似性的相关参数：

```
DEV_SET=Writing
# The models needs to be evaluated. Using ',' to split
# models in paper: qwen-14b,vicuna-13b,wizardlm-13b,chatglm3-6b,gpt-4-1106-
preview,gpt-3.5-turbo-1106,openchat-3.5,gemini-pro
EVAL_MODELS=chatglm3-6b,gpt-3.5-turbo-1106,...
# 3 metrics: gpt-4-1106-preview, bert-score, text-embedding-ada-002
model=text-embedding-ada-002
gen_prompt_type=gpt-4-eval     # The prompt for gpt-4 metric
max_tokens=1024
temperature=0.0
sample_num=-1
output_path=./outputs/eval/${model}

# For gpt-4 metric
export OPENAI_API_KEY='Your OpenAI API KEY'
CUDA_VISIBLE_DEVICES=0,1 python similarity_check.py \
    --gen_prompt_type ${gen_prompt_type} \
    --dev_set ${DEV_SET} \
    --eval_models ${EVAL_MODELS} \
    --model ${model} \
    --max_tokens ${max_tokens} \
    --temperature ${temperature} \
    --sample_num ${sample_num} \
    --output_path ${output_path} \
    --api_batch 200 \
```

> 对于Chatbot_Arena，设置`DEV_SET=chatbot_arena`即可

step 4: MAD competition

选定场景、MAD metric和Top-K的取值，我们就可以获得经过MAD competition选择的差异最大的数据。

```
bash ./scripts/mad_competition.sh
```

step 5: Human Preference Annotation

在人工偏好标注之前，一个数据的格式应当如下：

```
{
    "instruction": "xxx",
    "input": "",
    "output": "",
    "response_1": "model_1 response",
    "response_2": "model_2 response",
    "score": "similarity score",
    "source": "file name, e.g., model_1-vs-model_2.jsonl"
}
```

在标注过程中，标注的结果应当是赢家的模型名称，或者是'tie'。在获得人工标注结果后，应将结果放置于`output`中，即：

```
"instruction": "xxx",
    "input": "",
    "output": "winner name or tie",
    ...
```

最后，我们可以将结果用如下代码统一，保存于json文件中。以chatbot_arena为例：

```
eval_model = 'text-embedding-ada-002'
domain = 'chatbot_arena'

f = open(f'./outputs/MAD/{eval_model}/{domain}.jsonl', 'r')
save_f = open(f'./outputs/annotation/{eval_model}/{domain}.jsonl', 'w')

total_data = [json.loads(line) for line in f.readlines()]       # The MAD
selected data
# we save the human annotation of chatbot arena in `output`

save_dict_list = []
for i, d in enumerate(total_data):
    model_a, model_b = d['source'].replace(f'.jsonl', '').split('-vs-')
    responses = [(model_a, d['response_1']), (model_b, d['response_2'])]
    random.shuffle(responses)

    if d['output'] == responses[0][0]:
        winner = 'model_a'
    elif d['output'] == responses[1][0]:
        winner = 'model_b'
    else:
```

```
            winner = 'tie'

        dt = datetime.now()
        save_dict = {
            'model_a': responses[0][0],
            'model_b': responses[1][0],
            'winner': winner,
            'judge': 'arena_user',
            'turn': 1,
            'anony': True,
            'language': 'English',
            'tstamp': dt.timestamp(),
            'source': d['source'],
            'data': {
                'instruction': d['instruction'],
                'input': d['input'],
                'output': d['output'],
                'response_1': responses[0][1],
                'response_2': responses[1][1],
                'score': float(d['score']),
                'explanation': d['explanation']
            }
        }
        if domain in ['Reasoning', 'Understanding', 'Coding']:
            save_dict['data']['answer'] = d['answer']
        save_dict_list.append(save_dict)

    json.dump(save_dict_list, save_f, indent=4, ensure_ascii=False)
    save_f.close()
```

Elo Ranking

Elo