تمرین سری هفتم کامیار مرادیان زه آب ۹۹۵۲۲۱۰۴

در ابتدا dependencyهای مورد نیاز نظیر transformers و datasets و نیز accelerate (جهت تسریع و بهبود کارایی مدل‌ها) را نصب می‌کنیم و کتابخانه‌های مورد نیاز را ایمپورت می‌کنیم. پس از قرار دادن دیوایس بر روی cuda، وارنینگ‌های transformers را غیر فعال می‌کنیم و در پایان جهت ثابت بودن وضعیت تصادفی بودن مدل‌ها، سید را برابر با مقدار ۱۱۱۱ قرار می‌دهیم.

سپس به سراغ دیتاست می‌رویم و با استفاده از load_dataset آن را از هاگینگ‌فیس لود می‌کنیم. برای اطمینان از برهم‌خوردن داده‌ها و تصادفی بودن انتخاب آن‌ها نیاز است که در ادامه با استفاده از shuffle آن‌ها را به صورت نامرتب درآوریم. در گام بعدی نیز جفت سوال‌های موجود در دیتاست را به همراه لیبل آن‌ها را استخراج می‌کنیم و به صورت یک لیست در می‌آوریم.

```
dataset = load_dataset("persiannlp/parsinlu_query_paraphrasing")
dataset = dataset.shuffle(seed=1111)
train_pairs = [{'question': (row['q1'], row['q2']), 'label': 'Yes' if row['label']=='1' else 'No'} for row in dataset['train']]
test_pairs = [{'question': (row['q1'], row['q2']), 'label': 'Yes' if row['label']=='1' else 'No'} for row in dataset['test']]
```

حال به سراغ مدل‌ها می‌رویم. مدل‌های زیر برای ارزیابی نهایی مورد استفاده قرار گرفته شده‌اند:

- NousResearch/Meta-Llama-3-8B-Instruct
- microsoft/Phi-3-mini-4k-instruct
- universitytehran/PersianMind-v1.0

از آنجا که وظیفهٔ مدل‌ها به طریقی چت کردن است، در نتیجه در ابتدا نیاز است که پرامپت‌ها را برای مدل‌ها تعریف کنیم. به منظور تفاوت در پرامپت‌ها و یافتن بهترین پرامپت برای هر مدل، از پنج فرمت مختلف برای پراپمت‌دهی به مدل‌ها استفاده شده است. هر یک از این پرامپت‌ها سه خروجی دریافت می‌کنند:

- questions: که پارامتر مربوط به جفت سوالات و لیبل مرتبط است. آخرین سوال که سوال مورد نظر است در سر آن قرار گرفته پس در اولین مرحله این لیست را معکوس می‌کنیم.
- user_add_prompt: پرامپت اضافی برای یوزر است که پیش از آخرین پیام از طرف یوزر قرار می‌گیرد.
- need_system_prompt: مشخص می‌کند که آیا در پرامپت مورد نظر نیاز است که از پرامپت system استفاده کنیم یا خیر.

نکته: در هر مدل داخل transformers_config.json تعدادی نقش (role) برای هر مدل تعیین شده است. از جملهٔ این نقش‌ها، نقش system، نقش user و نقش assistant هستند. البته برخی مدل‌ها مانند مدل Phi-3 از system پشتیبانی نمی‌کنند.

با توجه به نکات گفته شده، پنج تابع را می‌سازیم. به طوری که استایل اول و آخر کامل انگلیسی، و استایل‌های دوم و سوم ترکیب فارسی و انگلیسی و استایل چهارم کامل فارسی است. در کد می‌توانید جزئیات هر یک از استایل‌ها را در توابع generate_messages_style_[1,2,3,4,5] مشاهده کنید.

در ادامه به منظور تست کردن هر مدل سه سری تابع نوشته شده است:

- evaluate_model_on_message_generators: در این تابع یک output_generator را دریافت می‌کنیم که وظیفهٔ بدست آوردن خروجی هر پرامپت را بر عهده دارد. همچنین یک model_id را نیز می‌گیریم که برای لاگ نوشتن استفاده می‌شود. علاوه بر موارد مذکور check_count نیز به عنوان ورودی دریافت می‌شود که مشخص

می‌کند چه تعداد از لیست training را نیاز داریم که با استفاده از آن‌ها تست را انجام دهیم. در این تابع، به ازای هر یک از استایل‌های نوشته شده، هر یک از روش‌های گفته شده را برای مدل امتحان می‌کنیم.

- [zero,one,few]_shot_evaluation: این توابع توسط تابع بالا صدا زده می‌شوند. در این توابع در صورتی که هدف one یا few باشد، به تعداد ۱ و ۵ داده از داخل داده‌های تست به صورت تصادفی استخراج می‌شود و سپس تابع بعدی صدا زده می‌شود.
- evaluate_model: این تابع توسط تابع بالای صدا زده می‌شود. این تابع وظیفهٔ بدست آوردن پرامپت مورد نظر، دادن آن به مدل مورد نظر و در نتیجه محاسبهٔ دقت مدل است.

```python
def evaluate_model(output_generator, message_generator, inital_questions, pairs, check_count, user_add_prompt, need_system_prompt):
    correct = 0
    check_pairs = pairs[:check_count]
    for row in tqdm(check_pairs):
        messages, expected = message_generator(list(inital_questions) + [row], user_add_prompt, need_system_prompt)
        prediction = output_generator(messages)
        if expected == prediction:
            correct += 1
    return correct / check_count

def zero_shot_evaluation(output_generator, message_generator, check_count, user_add_prompt, need_system_prompt):
    inital_questions = []
    accuracy = evaluate_model(output_generator, message_generator, inital_questions, train_pairs, check_count, user_add_prompt, need_system_prompt)
    return accuracy


def one_shot_evaluation(output_generator, message_generator, check_count, user_add_prompt, need_system_prompt):
    inital_questions = np.random.choice(test_pairs, size=1, replace=False)
    accuracy = evaluate_model(output_generator, message_generator, inital_questions, train_pairs, check_count, user_add_prompt, need_system_prompt)
    return accuracy

def few_shot_evaluation(output_generator, message_generator, check_count, user_add_prompt, need_system_prompt):
    inital_questions = np.random.choice(test_pairs, size=5, replace=False)
    accuracy = evaluate_model(output_generator, message_generator, inital_questions, train_pairs, check_count, user_add_prompt, need_system_prompt)
    return accuracy

def evaluate_model_on_message_generators(output_generator, model_id, check_count=20, user_add_prompt='', need_system_prompt=True):
    print(f"Evaluating {model_id}")

    for i, mg in enumerate(message_generators):
        print(f"Checking Style {i + 1}")
        zero_shot_results[model_id][i + 1] = zero_shot_evaluation(output_generator, mg, check_count, user_add_prompt, need_system_prompt)
        one_shot_results[model_id][i + 1] = one_shot_evaluation(output_generator, mg, check_count, user_add_prompt, need_system_prompt)
        few_shot_results[model_id][i + 1] = few_shot_evaluation(output_generator, mg, check_count, user_add_prompt, need_system_prompt)

    print()
    print(f"Result for {model_id}:")
    print("\tZero-shot results:", zero_shot_results[model_id])
    print("\tOne-shot results:", one_shot_results[model_id])
    print("\tFew-shot results:", few_shot_results[model_id])
```

برای تست کردن هر مدل در ابتدا نیاز است که هر یک را لود کنیم. برای امتحان کردن مدل از pipeline استفاده شده است چرا که رویه را ساده‌تر می‌کند. به عنوان مثال برای لود کردن مدل llama از قطعه کد زیر استفاده شده است:

```python
pipe0 = pipeline(
    "text-generation",
    model=model_names[0],
    model_kwargs={"torch_dtype": torch.bfloat16},
    device_map="auto",
)
```

به منظور حل کردن مشکلات ناشی از حجم بالای مدل نیاز بر این بود که از دیتاتایپ bfloat16 استفاده کنیم که نصف فضای اصلی جا می‌گیرد. در ادامه برای هر مدل نیاز است که تابع output_generator مناسب را بسازیم. در این توابع ورودی که

پیام‌ها باشند را دریافت می‌کنیم و با اپلای کردن template مربوط به چت بر روی آن‌ها، هر یک را از داخل پایپ‌لاین گذر می‌دهیم تا خروجی مناسب بدست آید.

```python
def generate_llama_output(messages):
    prompt = pipe0.tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )

    terminators = [
        pipe0.tokenizer.eos_token_id,
        pipe0.tokenizer.convert_tokens_to_ids("<|eot_id|>")
    ]

    outputs = pipe0(
        prompt,
        max_new_tokens=256,
        eos_token_id=terminators,
        do_sample=True,
        temperature=0.6,
        top_p=0.9,
    )

    return outputs[0]["generated_text"][len(prompt):]
```

برای مدل llama به صورت بالا تابع را می‌نویسیم. add_generation_prompt برای آن است که مدل ادامهٔ کار را از طرف assistant انجام دهد و جواب اصلی را بدهد. همچنین سه پارامتر نهایی برای پایپ‌لاین برای تولید تصادفی است که در صورت کامنت شدن نیز تفاوتی در خروجی نهایی به دست نخواهد آمد. در پایان جواب اصلی داده شده توسط مدل را جدا کرده و به عنوان خروجی بیرون می‌دهیم. رویهٔ کار output generator های دیگر نیز مشابه هستند و تفاوتی چندانی با این تابع ندارند.

خروجی‌های بدست آمده نیز به صورت‌های زیر هستند:

```
Evaluating NousResearch/Meta-Llama-3-8B-Instruct
Checking Style 1
100%|██████████| 20/20 [00:59<00:00,  3.00s/it]
100%|██████████| 20/20 [00:48<00:00,  2.43s/it]
100%|██████████| 20/20 [01:19<00:00,  3.99s/it]
Checking Style 2
100%|██████████| 20/20 [00:46<00:00,  2.32s/it]
100%|██████████| 20/20 [00:51<00:00,  2.59s/it]
100%|██████████| 20/20 [01:21<00:00,  4.08s/it]
Checking Style 3
100%|██████████| 20/20 [00:46<00:00,  2.34s/it]
100%|██████████| 20/20 [00:54<00:00,  2.75s/it]
100%|██████████| 20/20 [01:19<00:00,  3.95s/it]
Checking Style 4
100%|██████████| 20/20 [00:43<00:00,  2.19s/it]
100%|██████████| 20/20 [00:48<00:00,  2.41s/it]
100%|██████████| 20/20 [01:11<00:00,  3.56s/it]
Checking Style 5
100%|██████████| 20/20 [00:44<00:00,  2.22s/it]
100%|██████████| 20/20 [00:48<00:00,  2.41s/it]
100%|██████████| 20/20 [01:12<00:00,  3.64s/it]
Result for NousResearch/Meta-Llama-3-8B-Instruct:
        Zero-shot results: {1: 0.55, 2: 0.55, 3: 0.55, 4: 0.5, 5: 0.5}
        One-shot results: {1: 0.6, 2: 0.65, 3: 0.6, 4: 0.75, 5: 0.55}
        Few-shot results: {1: 0.65, 2: 0.6, 3: 0.75, 4: 0.65, 5: 0.6}
```

```
Evaluating microsoft/Phi-3-mini-4k-instruct
Checking Style 1
100%|██████████| 20/20 [00:08<00:00,  2.28it/s]
100%|██████████| 20/20 [00:14<00:00,  1.34it/s]
100%|██████████| 20/20 [00:37<00:00,  1.89s/it]
Checking Style 2
100%|██████████| 20/20 [00:08<00:00,  2.27it/s]
100%|██████████| 20/20 [00:13<00:00,  1.51it/s]
100%|██████████| 20/20 [00:40<00:00,  2.01s/it]
Checking Style 3
100%|██████████| 20/20 [00:09<00:00,  2.22it/s]
100%|██████████| 20/20 [00:16<00:00,  1.24it/s]
100%|██████████| 20/20 [00:46<00:00,  2.31s/it]
Checking Style 4
100%|██████████| 20/20 [00:08<00:00,  2.27it/s]
100%|██████████| 20/20 [00:15<00:00,  1.30it/s]
100%|██████████| 20/20 [00:35<00:00,  1.79s/it]
Checking Style 5
100%|██████████| 20/20 [00:08<00:00,  2.29it/s]
100%|██████████| 20/20 [00:13<00:00,  1.50it/s]
100%|██████████| 20/20 [00:33<00:00,  1.69s/it]
Result for microsoft/Phi-3-mini-4k-instruct:
        Zero-shot results: {1: 0.65, 2: 0.65, 3: 0.0, 4: 0.65, 5: 0.5}
        One-shot results: {1: 0.45, 2: 0.35, 3: 0.25, 4: 0.5, 5: 0.5}
        Few-shot results: {1: 0.45, 2: 0.55, 3: 0.4, 4: 0.5, 5: 0.7}
```

```
Evaluating universitytehran/PersianMind-v1.0
Checking Style 1
100%|██████████| 20/20 [00:16<00:00,  1.22it/s]
100%|██████████| 20/20 [00:24<00:00,  1.23s/it]
100%|██████████| 20/20 [00:39<00:00,  1.99s/it]
Checking Style 2
100%|██████████| 20/20 [00:15<00:00,  1.26it/s]
100%|██████████| 20/20 [00:23<00:00,  1.15s/it]
100%|██████████| 20/20 [00:46<00:00,  2.31s/it]
Checking Style 3
100%|██████████| 20/20 [00:15<00:00,  1.26it/s]
100%|██████████| 20/20 [00:23<00:00,  1.17s/it]
100%|██████████| 20/20 [00:43<00:00,  2.17s/it]
Checking Style 4
100%|██████████| 20/20 [00:15<00:00,  1.28it/s]
100%|██████████| 20/20 [00:22<00:00,  1.14s/it]
100%|██████████| 20/20 [00:44<00:00,  2.22s/it]
Checking Style 5
100%|██████████| 20/20 [00:15<00:00,  1.28it/s]
100%|██████████| 20/20 [00:22<00:00,  1.13s/it]
100%|██████████| 20/20 [00:38<00:00,  1.92s/it]
Result for universitytehran/PersianMind-v1.0:
        Zero-shot results: {1: 0.4, 2: 0.4, 3: 0.0, 4: 0.4, 5: 0.4}
        One-shot results: {1: 0.35, 2: 0.4, 3: 0.35, 4: 0.4, 5: 0.4}
        Few-shot results: {1: 0.4, 2: 0.4, 3: 0.4, 4: 0.35, 5: 0.35}
```

همانطور که مشخص است مدل اول بهترین نتیجه را داشته، به طوری که این بهترین نتیجه بر روی استایل‌های سوم و چهارم حاصل شده‌اند که به ترتیب برای ترکیب فارسی انگلیسی و کامل فارسی بوده‌اند.

مدل دوم نیز از طرفی بهترین جواب را برای few-shot برای استایل آخر که انگلیسی است، نتیجه می‌دهد. برای zero-shot نیز بهترین نتیجه در پرامپت‌های اول و دوم و چهارم بدست آمده‌اند. همچنین برای پرامپت سوم نتیجهٔ صفر داده شده است.

برای مدل سوم نیز نتایج تا حدودی یکسان بدست آمده‌اند. مگر پرامپت سوم برای zero-shot که مطابق مدل قبلی صفر بدست آمده است.

از جمله تفاوت‌های این سه مدل، در این است که مدل دوم از نقش system پشتیبانی نمی‌کند در حالی که دو مدل دیگر از این نقش استفاده می‌کنند تا بتوانند نتایج بهتری را حاصل کنند. هر چند همانطور که مشخص شد مدل آخر بدترین نتیجه را به ما داد. همچنین صرف نظر از پشتیبانی این مدل از زبان فارسی، تفاوتی میان پرامپت‌های فارسی و انگلیسی برای این مدل بدست نیامده است. و حتی مدل llama که بر روی داده‌های انگلیسی تمرین داده شده است، در مقایسه با این مدل در هنگام

دریافت دادهٔ فارسی بهتر عمل کرده است. علت بهتر بودن این مدل در داده‌های فارسی، می‌تواند این باشد که سوالات نیز خود فارسی بوده‌اند و هنگامی که کانتکست کاملاً در یک زبان بوده است، نتیجهٔ بهتری حاصل شده است.

همچنین بر خلاف مدل llama دو مدل دیگر در پرامپت استایل سوم، بدترین نتیجه را داشتند اما مدل llama بهترین نتیجه را در این پرامپت دریافت کرد.

بررسی مدل‌ها در سناریوهای مختلف:

	zero-shot	one-shot	few-shot
llama	با بیس ۰.۵ و نهایت ۰.۵۵ و تاثیرپذیری کم از پرامپت‌ها	نتیجهٔ بهتر نسبت به حالت بیس و تاثیرپذیری از پرامپت‌ها	نتیجهٔ بهتر نسبت به حالت بیس و تاثیرپذیری از پرامپت‌ها – این حالت نسبت به one-shot تاحدودی نتایج استیبل‌تر بودند.
Phi-3	بهترین بیس در میان مدل‌ها را داشته است. (به جز پرامپت سوم)	با در نظر گرفتن میانگین بدترین نتیجه را در میان one-shotها داشته است. به طوری که حتی نسبت به zero-shot هم بدتر عمل کرده است.	در این وضعیت با استفاده از پرامپت ۵، بهترین نتیجه برای این مدل حاصل شده است. اما در حالات دیگر همچنان عملکرد مدل پایین است. پس بهترین پرامپت برای این حالت مدل پرامپت ۵ است.
PersianMind	این مدل در هر سه سناریو تا حدودی رفتاری مشابه را از خود نشان داده است. (مگر پرامپت سوم که برای این سناریو نتیجهٔ صفر را داده است.)	این مدل در هر سه سناریو تا حدودی رفتاری مشابه را از خود نشان داده است.	این مدل در هر سه سناریو تا حدودی رفتاری مشابه را از خود نشان داده است.

بنابراین با در نظر گرفتن میانگین، بهترین پرامپت برای مدل llama، استایل سوم، برای Phi-3 استایل پنجم، و در مدل آخر نیز تفاوت چندانی وجود نداشته است.