Experimental Report on Robustness Task - ELOQUENT Lab @ CLEF 2024

Notebook for the ELOQUENT Lab at CLEF 2024

Annika Simonsen

The University of Iceland, Sæmundargata 2, 102 Reykjavík, Iceland

Abstract
The ELOQUENT Lab's Robustness Task for CLEF 2024 aimed to evaluate the robustness of language models against various input variations, such as dialectal, sociolectal, and cross-cultural differences. The task aimed to assess how well language models could maintain consistent, functionally equivalent output despite surface variations in input prompts.

Keywords
Language models, Robustness, CLEF 2024, Dialects, Cross-cultural variations

1. Introduction

The ELOQUENT Lab's Robustness Task for CLEF 2024 aimed to evaluate the robustness of language models against various input variations, such as dialectal, sociolectal, and cross-cultural differences. The task aimed to assess how well language models could maintain consistent, functionally equivalent output despite surface variations in input prompts.

2. Objective

The primary goal was to measure the robustness of language models to input variations by analyzing the consistency of their responses across different dialectal and cultural prompts. This would involve evaluating the output using measures of language variation, such as n-gram overlap and embedding similarity, to ensure that the responses maintained their topical and semantic content.

3. Method

3.1. Models used

I employed the following models for this task:

- gpt-4-turbo-2024-04-09 [1]
- gpt-sw3-20b-instruct [2]
- gpt-sw3-20b [2]
- gpt-sw3-40b [2]

The models were chosen for their advanced capabilities and availability. The GPT-4-turbo model was the state-of-the-art model at the time of the experiment, while the GPT-SW3 series from AI Sweden is lesser known and is therefore, studied less. The GPT-SW3 models are a series of Nordic models, primarily trained on Swedish text data, but also Norwegian, Danish, Icelandic and English text data.

CLEF 2024: Conference and Labs of the Evaluation Forum, September 09–12, 2024, Grenoble, France
ans72@hi.is (A. Simonsen)
https://github.com/AnnikaSimonsen (A. Simonsen)

3.2. Prompts and variations

The prompts were provided by the organizers and were written in in multiple languages, including Swedish, English, Arabic, and Finnish, to test the models' ability to handle linguistic diversity. The variations within these prompts included differences in dialect, sociolect, idiolect, and levels of formality. I kept the provided prompts unchanged and I did not add any system prompt.

3.3. Submission details

Two submissions were made:

1. **Initial Submission:** Included results from gpt-4-turbo-2024-04-09, gpt-sw3-20b-instruct, and gpt-sw3-20b. This submission faced issues where the GPT-SW3 models tended to repeat the prompts themselves.
2. **Second Submission:** Included results from gpt-sw3-40b base. This submission also encountered similar issues, and prompts were split into three batches due to computational constraints.

3.4. Technical Setup

The scripts are included in the appendix.

- **Temperature and Sampling Settings:** For both gpt-4-turbo and the GPT-SW3 models, the temperature was set to 0.7 and for the GPT-SW3 models the top-p sampling was set to 1 to ensure a balance between creativity and determinism.
- **Hardware:** The models were run on a single GPU (NVIDIA A100) on the university cluster. Due to computational limitations, prompts were processed in batches.

4. Results and Discussion

At the moment of writing, the evaluation of the output has not been released, so the following is the observations made during the experiment itself. The GPT-SW3 models struggled with prompt repetition, which could potentially be due to a coding error or model-specific behavior. If I were to run the experiments again, I would re-formulate the prompts in order to get GPT-SW3 to follow the instructions better.

I encountered that the output was not in UTF-8 encoding, even though the dataset was loaded in with encoding='utf-8'. Therefore I had to first process the generated responses and then map the responses back to items, which were saved into the appropriate structure with encoding='utf-8'. See the full scripts in the appendix.

I did not add any prompts to the prompt collection, although it would have been interesting to add prompts written in Faroese, a low-resource Nordic language that is my native language. Ultimately, this was not done due to time constraints.

4.1. Conclusion

The Robustness task was run on four different models, one GPT-4 model and three GPT-SW3 models. Future work could focus on improving prompt handling in models like GPT-SW3 and exploring additional languages and dialects to further test model robustness.

References

[1] OpenAI, GPT-4 Model, 2023. URL: https://www.openai.com/research/gpt-4.
[2] AI Sweden, GPT-SW3 Models, 2024. URL: https://www.ai.se/en/gpt-sw3.

A. Scripts

A.1. Script for gpt-4-turbo

```
import os
import json
from openai import OpenAI

# Replace 'MY-API-KEY-HERE' with your actual OpenAI API key
os.environ["OPENAI_API_KEY"] = "MY-API-KEY-HERE"

client = OpenAI()


def generate_response(prompt):
    try {
        # Generate a response using the chat completion method
        completion = client.chat.completions.create(
            model="gpt-4-turbo-2024-04-09",
            temperature=0.7,
            messages=[
                {"role": "user", "content": prompt}
            ]
        )
        # Correct extraction of the response text
        response_text = completion.choices[0].message.content
        return response_text.strip()
    } except Exception as e {
        print(f"Error generating response: {e}")
        return ""
    }
}

# Load the dataset from a JSON file with UTF-8 encoding
with open('task3.test1.json', 'r', encoding='utf-8') as file {
    data = json.load(file)
    prompts = data['eloquent-robustness-test']['items']

# Prepare the results dictionary
results = {"eloquent-robustness-results": {"source": "Annika-UOI", "year": "2024", "items": []}}

# Process each prompt in the dataset
for item in prompts {
    item_id = item['id']
    responses = []
    variants = item['variants']
    for variant in variants {
        prompt = variant['prompt']
        response_text = generate_response(prompt)
        responses.append({"response": response_text})
    }

    # Add the responses for each item to the results
    results['eloquent-robustness-results']['items'].append({"id": item_id, "responses": responses})
}

# Saving the data
with open('submission_fo.json', 'w', encoding='utf-8') as out_file {
    json.dump(results, out_file, ensure_ascii=False, indent=2)
}

# Reading back the saved data for verification
with open('submission_fo.json', 'r', encoding='utf-8') as in_file {
    loaded_data = json.load(in_file)
    print(loaded_data)  # Check how the data reads back
}
```

A.2. Script for gpt-sw3 models

```
import torch
import json
from transformers import AutoTokenizer, AutoModelForCausalLM
```

```
import os

os.environ['PYTORCH_CUDA_ALLOC_CONF'] = "max_split_size_mb:128"

gpu_id =  os.environ.get("CUDA_VISIBLE_DEVICES")

if gpu_id is not None {
    print(f"running gpu {gpu_id}")
} else {
    print(f"No gpu found")
}

# Setup for HuggingFace and Torch
model_name = "AI-Sweden-Models/gpt-sw3-40b"
# model_name = "AI-Sweden-Models/gpt-sw3-20b"
# model_name = "AI-Sweden-Models/gpt-sw3-20b-instruct"
device = "cuda:0" if torch.cuda.is_available() else "cpu"

# Initialize Tokenizer and Model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
model.eval()
model.to(device)

# Load dataset in batches
with open('task3.test1-5.json', 'r', encoding='utf-8') as file {
    data = json.load(file)
    prompts = data['eloquent-robustness-test']['items']
}

# Prepare the results dictionary and batch inputs
results_sw3 = {"eloquent-robustness-results-sw3": {"source": "Annika-UOI", "year": "2024", "items": []}}
batch_prompts = []
batch_info = []

# Accumulate prompts for batching
for item in prompts {
    item_id = item['id']
    variants = item['variants']
    for variant in variants {
        prompt = variant['prompt']
        batch_prompts.append(prompt)
        batch_info.append({"item_id": item_id, "prompt": prompt})
    }
}

# Tokenize the batch
input_ids = tokenizer(batch_prompts, return_tensors="pt", padding=True, truncation=True,
max_length=512)["input_ids"].to(device)

# Generate responses in a batch
generated_token_ids = model.generate(
    input_ids=input_ids,
    max_new_tokens=100,
    do_sample=True,
    temperature=0.7,
    top_p=1
)

# Process the generated responses
responses = [tokenizer.decode(ids, skip_special_tokens=True) for ids in generated_token_ids]

# Map responses back to items and prepare the result structure
for response, info in zip(responses, batch_info) {
    item_id = info["item_id"]
   if not any(item["id"] == item_id for item in results_sw3["eloquent-robustness-results-sw3"]["items"]) {
        results_sw3["eloquent-robustness-results-sw3"]["items"].append({"id": item_id, "responses": []})
    }
    item = next(item for item in
    results_sw3["eloquent-robustness-results-sw3"]["items"] if item["id"] == item_id)
    item["responses"].append({"prompt": info["prompt"], "response": response})
}
```

```
# Save the results to a JSON file
with open('submission_sw3_base40-1-5.json', 'w', encoding='utf-8') as out_file {
    json.dump(results_sw3, out_file, ensure_ascii=False, indent=2)
}
```