Assignment 2

Credits: Federico Ruggeri, Eleonora Mancini, Paolo Torroni

Keywords: Sexism Detection, Multi-class Classification, LLMs, Prompting

Contact

For any doubt, question, issue or help, you can always contact us at the following email addresses:

Teaching Assistants:

- Federico Ruggeri -> federico.ruggeri6@unibo.it
- Eleonora Mancini -> e.mancini@unibo.it

Professor:

- Paolo Torroni -> p.torroni@unibo.it

Relevant Material

- Tutorial 3
- Huggingface documentation
- Huggingface hub

Introduction

You are tasked to address the EDOS Task B on sexism detection.

Problem definition

Given an input text sentence, the task is to label the sentence as non-sexist or one of these four sexist categories: (1) threats, (2) derogation, (3) animosity, (4) prejudiced discussion.

Examples:

Text: ``*Schedule a date with her, then don't show up. Then text her "GOTCHA B___H".*''

Label: Threats

Text: ``*That's completely ridiculous a woman flashing her boobs is not sexual assault in the slightest.*''

Label: Not sexist

Approach

We will tackle the five-class classification task with LLMs.

In particular, we'll consider zero-/few-shot prompting approaches to assess the capability of some popular open-source LLMs on this task.

Preliminaries

We are going to download LLMs from Huggingface.

Many of these open-source LLMs require you to accept their "Community License Agreement" to download them.

In summary:

- If not already, create an account of Huggingface (~2 mins)
- Check a LLM model card page (e.g., Mistral v3) and accept its "Community License Agreement".
- Go to your account -> Settings -> Access Tokens -> Create new token -> "Repositories permissions" -> add the LLM model card you want to use.
- Save the token (we'll need it later)

Huggingface Login

Once we have created an account and an access token, we need to login to Huggingface via code.

- Type your token and press Enter
- You can say No to Github linking

In [ ]:
```
!hf auth login
```

```
    _|    _|  _|    _|    _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|
_|_|_|_|    _|_|      _|_|_|  _|_|_|_|
    _|    _|  _|    _|  _|        _|          _|    _|_|    _|  _|
_|        _|    _|  _|        _|
    _|_|_|_|  _|    _|  _|  _|_|  _|  _|_|    _|    _|  _|  _|  _|  _|_|
_|_|_|    _|_|_|_|  _|        _|_|_|
    _|    _|  _|    _|  _|    _|  _|    _|    _|    _|    _|_|  _|    _|
_|        _|    _|  _|        _|
    _|    _|    _|_|      _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|
_|        _|    _|    _|_|_|  _|_|_|_|
```

```
    A token is already saved on your machine. Run `huggingface-cli whoami`
to get more information or `huggingface-cli logout` if you want to log ou
t.
    Setting a new token will erase the existing one.
    To login, `huggingface_hub` requires a token generated from https://hu
ggingface.co/settings/tokens .
Enter your token (input will not be visible):
```

After login, you can download all models associated with your access token in addition to those that are not protected by an access token.

Data Loading

Since we are only interested in prompting, we do not require a train dataset.

We have preparared a small test set version of EDOS in our dedicated Github repository.

Check the `Assignment 2/data` folder. It contains:

- `a2_test.csv` → a small test set of 300 samples.
- `demonstrations.csv` -> a batch of 1000 samples for few-shot prompting.

Both datasets contain a balanced number of sexist and not sexist samples.

Instructions

We require you to:

- **Download** the `A2/data` folder.
- **Encode** `a2_test.csv` into a `pandas.DataFrame` object.

[Task 1 - 0.5 points] Model setup

Once the test data has been loaded, we have to setup the model pipeline for inference.

In particular, we have to:

- Load the model weights from Huggingface
- Quantize the model to fit into a single-GPU limited hardware

Which LLMs?

The pool of LLMs is ever increasing and it's impossible to keep track of all new entries.

We focus on popular open-source models.

- Mistral v2
- Mistral v3
- Llama v3.1
- Phi3-mini
- TinyLlama
- DeepSeek-R1
- Qwen3

Other open-source models are more than welcome!

Instructions

In order to get Task 1 points, we require you to:

- Pick 2 model cards from the provided list.
- For each model:
  - Setup a quantization configuration for the model.
  - Load the model via HuggingFace APIs.

Note

There's a popular library integrated with Huggingface's `transformers` to perform quantization.

[Task 2 - 1.0 points] Prompt setup

Prompting requires an input pre-processing phase where we convert each input example into a specific instruction prompt.

Prompt Template

Use the following prompt template to process input texts.

In [ ]:
```
prompt = [
    {
        'role': 'system',
        'content': 'You are an annotator for sexism detection.'
    },
    {
        'role': 'user',
```

```
        'content': """Your task is to classify input text as non-sexist
         or sexist. If sexist, classify input text according to one
         of the following four categories: threats, derogation,
         animosity, prejudiced discussion.

         Below you find sexist categories definitions:
         Threats: the text expresses intent or desire to harm a woman.
         Derogation: the text describes a woman in a derogative manner.
         Animosity: the text contains slurs or insults towards a woman.
         Prejudiced discussion: the text expresses supports for
         mistreatment of women as individuals.

         Respond only by writing one of the following categories:
         non-sexist, threats, derogation, animosity, prejudiced.

        TEXT: {text}

        ANSWER:
        """
    }
]
```

Instructions

In order to get Task 2 points, we require you to:

- Write a `prepare_prompts` function as the one reported below.

In [ ]:
```
def prepare_prompts(texts, prompt_template, tokenizer):
  """
    This function format input text samples into instructions prompts.

    Inputs:
      texts: input texts to classify via prompting
      prompt_template: the prompt template provided in this assignment
      tokenizer: the transformers Tokenizer object instance associated
      with the chosen model card

    Outputs:
      input texts to classify in the form of instruction prompts
  """
  pass
```

Notes

1. You are free to modify the prompt format (**not its content**) as you like depending on your code implementation.

2. Note that the provided prompt has placeholders. You need to format the string to replace placeholders. Huggingface might have dedicated APIs for this.

[Task 3 - 1.0 points] Inference

We are now ready to define the inference loop where we prompt the model with each pre-processed sample.

Instructions

In order to get Task 3 points, we require you to:

- Write a `generate_responses` function as the one reported below.
- Write a `process_response` function as the one reported below.

In [ ]:
```python
def generate_responses(model, prompt_examples):
  """
    This function implements the inference loop for a LLM model.
    Given a set of examples, the model is tasked to generate
    a response.

    Inputs:
      model: LLM model instance for prompting
      prompt_examples: pre-processed text samples

    Outputs:
      generated responses
  """
  pass
```

In [ ]:
```python
def process_response(response):
  """
    This function takes a textual response generated by the LLM
    and processes it to map the response to a binary label.

    Inputs:
      response: generated response from LLM

    Outputs:
      parsed classification response.
      Use the following mapping:
      {
        'not-sexist': 0,
        'threats': 1,
        'derogation': 2,
        'animosity': 3,
        'prejudiced': 4
      }
  """
  pass
```

Notes

1. According to our tests, it should take you ~10 mins to perform full inference on 300 samples on Colab.

[Task 4 - 0.5 points] Metrics

In order to evaluate selected LLMs, we need to compute performance metrics.

We compute **macro F1-score** and the ratio of failed responses generated by models (**fail-ratio**).

That is, how frequent the LLM fails to follow instructions and provides incorrect responses that do not address the classification task.

In summary, we parse generated responses as follows:

- **0** if 'not-sexist'
- **1** if 'threats'
- **2** if 'derogation'
- **3** if 'animosity'
- **4** if 'prejudiced'
- **0** if the model does not answer in either way

Instructions

In order to get Task 4 points, we require you to:

- Write a `compute_metrics` function as the one reported below.
- Compute metrics for the two selected LLMs.

In [ ]:
```python
def compute_metrics(y_pred, y_true):
  """
    This function takes predicted and ground-truth labels and compute
    metrics. In particular, this function compute accuracy and
    fail-ratio metrics. This function internally invokes
    `process_response` to compute metrics.

    Inputs:
      y_pred: parsed LLM responses
      y_true: ground-truth binary labels

    Outputs:
      dictionary containing desired metrics
  """
  pass
```

[Task 5 - 1.0 points] Few-shot Inference

So far, we have tested models in a zero-shot fashion: we provide the input text to classify and instruct the model to generate a response.

We are now interested in performing few-shot prompting to see the impact of providing demonstration examples.

To do so, we slightly change the prompt template as follows.

In [ ]:
```
prompt = [
    {
        'role': 'system',
        'content': 'You are an annotator for sexism detection.'
    },
    {
        'role': 'user',
        'content': """Your task is to classify input text as non-sexist
         or sexist. If sexist, classify input text according to one
         of the following four categories: threats, derogation,
         animosity, prejudiced discussion.

         Below you find sexist categories definitions:
         Threats: the text expresses intent or desire to harm a woman.
         Derogation: the text describes a woman in a derogative manner.
         Animosity: the text contains slurs or insults towards a woman.
         Prejudiced discussion: the text expresses supports for
         mistreatment of women as individuals.

         Respond only by writing one of the following categories:
         non-sexist, threats, derogation, animosity, prejudiced.

        EXAMPLES: {examples}

        TEXT: {text}

        ANSWER:
        """
    }
]
```

The new prompt template reports some demonstration examples to instruct the model.

Generally, we provide an equal number of demonstrations per class as shown in the example below.

In [ ]:
```
prompt = [
    {
        'role': 'system',
        'content': 'You are an annotator for sexism detection.'
    },
    {
        'role': 'user',
        'content': """Your task is to classify input text as non-sexist
         or sexist. If sexist, classify input text according to one
         of the following four categories: threats, derogation,
         animosity, prejudiced discussion.

         Below you find sexist categories definitions:
         Threats: the text expresses intent or desire to harm a woman.
         Derogation: the text describes a woman in a derogative manner.
         Animosity: the text contains slurs or insults towards a woman.
         Prejudiced discussion: the text expresses supports for
         mistreatment of women as individuals.

         Respond only by writing one of the following categories:
         non-sexist, threats, derogation, animosity, prejudiced.
```

```
        EXAMPLES:
        TEXT: **example 1**
        ANSWER: threats
        TEXT: **example 2**
        ANSWER: not-sexist

        TEXT: {text}

       ANSWER:
       """
    }
]
```

Instructions

In order to get Task 5 points, we require you to:

- Load `demonstrations.csv` and encode it into a `pandas.DataFrame` object.
- Define a `build_few_shot_demonstrations` function as the one reported below.
- Modify `prepare_prompts` to support demonstrations.
- Perform few-shot inference as in Task 3.
- Compute metrics as in Task 4.

In [ ]:
```
def build_few_shot_demonstrations(demonstrations, num_per_class=2):
  """
    Inputs:
      demonstrations: DataFrame wrapping demonstrations.csv
      num_per_class: number of demonstrations per class

    Outputs:
      list of demonstrations to inject into the prompt template.
  """
  pass
```

Notes

1. You are free to pick any value for `num_per_class`.
2. According to our tests, few-shot prompting increases inference time by some minutes (we experimented with `num_per_class` $\in [2, 4]$).

[Task 6 - 1.0 points] Error Analysis

We are now interested in evaluating model responses and comparing their performance.

This analysis helps us in understanding

- Classification task performance gap: are the models good at this task?
- Generation quality: which kind of responses do models generate?
- Errors: which kind of mistakes do models do?

Instructions

In order to get Task 6 points, we require you to:

- Compare classification performance of selected LLMs in a Table.
- Compute confusion matrices for selected LLMs.
- Briefly summarize your observations on generated responses.

[Task 7 - 1.0 points] Report

Wrap up your experiment in a short report (up to 2 pages).

Instructions

- Use the NLP course report template.
- Summarize each task in the report following the provided template.

Recommendations

The report is not a copy-paste of graphs, tables, and command outputs.

- Summarize classification performance in Table format.
- **Do not** report command outputs or screenshots.
- Report learning curves in Figure format.
- The error analysis section should summarize your findings.

Submission

- **Submit** your report in PDF format.
- **Submit** your python notebook.
- Make sure your notebook is **well organized**, with no temporary code, commented sections, tests, etc...

FAQ

Please check this frequently asked questions before contacting us.

Model cards

You can pick any open-source model card you like.

We recommend starting from those reported in this assignment.

Implementation

Everything can be done via `transformers` APIs.

However, you are free to test frameworks, such as LangChain, LlamaIndex LitParrot, provided that you correctly address task instructions.

Task Performance

The task is challenging and zero-shot prompting may show relatively low performance depending on the chosen model.

Prompt Template

Do not change the provided prompt template.

You are only allowed to change it in case of a possible extension.

Optimizations

Any kind of code optimization (e.g., speedup model inference or reduce computational cost) is more than welcome!

Bonus Points

0.5 bonus points are arbitrarily assigned based on significant contributions such as:

- Outstanding error analysis
- Masterclass code organization
- Evaluate A1 dataset and perform comparison
- Perform prompt tuning

Note that bonus points are only assigned if all task points are attributed (i.e., 6/6).

The End