Assignment 2:
Prompt Learning for Event Detection

Due: 11:59 pm Central Time on Oct 07, 2022.

# Overview

In this assignment, you will develop a sentence-level event detection system based on prompt learning. You will be experimenting on designing different prompt templates, verbalizers, and loss functions to improve the event detection performance.

# Introduction

1. **Event Detection:**

Event Detection (ED) is a traditional Information Extraction task which includes two consecutive steps: 1) **Identification:** Givenan input sentence, the model should identify how many types of events are mentioned in the sentence. 2) **Localization:** For all the event types mentioned in the sentence, the model should extract an event trigger (a text span in the sentence) that best indicates the occurrence of the event. An example of event detection is shown as follows.

**Input:** A Montana helicopter company transported 40 people out from the flooding.

**Output:** A Montana helicopter company transported [TRANSPORT] 40 people out from the flooding [DISASTER].

In this example, the model takes a sentence “A Montana helicopter company transported 40 people out from the flooding” as input. In the **identification** step, the model should identify that there are “Transport” and “Disaster” events happened in the sentence. In the **localization** step, the model should extract “transported” and “flooding” as the event triggers of the two events respectively.

In this assignment, we mainly focus on the **identification** step. The problem essentially becomes a multi-label classification problem. (Each sentence could contain multiple events, a single event, or no events.)

1. **Prompt Learning**

Prompt Learning is a new fine-tuning technique for pretrained language models (PLM), where prompt templates are designed and the model will make decisions based on the probabilities of the verbalizers of each class returned from the PLM. See the simple sentiment classification task as example in the lecture.

1. **Prompt Learning for Event Detection**

In this assignment, we aim to building a prompt-learning based model for event detection, which includes three main steps: 1) **Template:** We design a text template to ask the PLM which event types are included in the sentence 2) **Verbalizers:** We design the verbalizers (mappings from words to event types) for event identification (e.g., “good” and “bad” for the sentiment classification example). The PLM will make decisions based on the probabilities of these words. 3) **Loss function:** We design the loss function of training the model to maximize the probability of the words corresponding to the correct event types. A typical related paper is at: <https://arxiv.org/pdf/2202.07615.pdf> (PILED)

# Data

In this assignment, we will use the **MAVEN** dataset which is the largest dataset for event detection. MAVEN has 168 event types and we only focus on the top 10 frequent event types in this assignment:

['Catastrophe', 'Hostile\_encounter', 'Attack', 'Causation', 'Process\_start', 'Competition', 'Motion', 'Social\_event', 'Killing', 'Conquering']

We provide a pre-processed version of the MAVEN dataset at: <https://uofi.box.com/s/eo14g4aklh6sji2yw7la8p4a2dsurnc2> This folder includes two versions of the dataset: in *{train/valid/text}.json* we only keep the top 10 frequent event types; while in *{train/valid/text}\_full.json* we keep all 168 event types in the original dataset.

**Data Format:**

Each line of a .json file is a sentence with event annotations. The “tokens” dict key denotes the tokenized sentence, and the “events” dict key is a list of events in the sentence.

An Example:

data = {"tokens": ["though", "the", "catholic", "royalists", "were", "not", "entirely", "defeated", ",", "the", "fact", "the", "hussites", "were", "able", "to", "inflict", "such", "heavy", "casualties", "with", "so", "few", "men", ",", "and", "then", "escape", "soundly", "proved", "to", "be", "a", "great", "victory", "."], "events": [[19, 20, "Catastrophe"], [7, 8, "Conquering"]]}

There are two events in this sentence:

1. Catastrophe: which is triggered by data[“tokens”][19:20] -> “casualties”.
2. Conquering: which is triggered by data[“tokens”][7:8] -> “defeated”.

**Note that each test split of dataset doesn’t have any events since it is blind to public.**

# Tasks

We provide a code skeleton at: <https://github.com/zhangzx-uiuc/PromptED/blob/main/identification.py> based on OpenPrompt(<https://github.com/thunlp/OpenPrompt> ). Please also feel free to build your model based on other repos.

1. **Design your template. (***get\_template()***)**

The first thing you need to do is to design your template for prompt learning. For event detection, a good template could be “[Sentence]. This sentence describes a [MASK] event.” (which is used by PILED). But you can also design your own template which could perform better than that.

1. **Design your verbalizer. (***get\_verbalizer()***)**

The verbalizers are the mappings from the event types to some important words. The predictions of a prompt-based model is based on how likely (probabilities) these words fit into the [MASK] in your template. The most straight-forward way is to directly use the event types names (Catastrophe, Conquering, etc.) as verbalizers. But we can definitely do something better than that. In PILED model, the authors use the most likely trigger words as verbalizers. You can also design your own verbalizers.

1. **Design your loss function.**

When training a prompt-based model, the general idea for designing the loss function is to maximize the logits for correct labels and minimize the logits for the incorrect labels. You can also design your own loss function to test performances.

1. **Write your prediction function.**

The most straight-forward way for prediction is to select out the indices with maximum of logits. Note that this is a multi-label classification problem and each sentence could have multiple predicted event indices. Therefore, using what threshold for prediction is important for a better performance. You can also use the None event (index 0) as the threshold as what PILED does.

1. **Train the model and dump out the predictions on the test dataset.**

You need to write out all your predictions on the test set into an “*output.json*” file, where each line represents a sentence from the test set. Each sentence should use a “predictions” dict key which is a list of event names for the input sentence. (If no events, just use an empty list.)

For example,

{"predictions": ["Catastrophe", "Conquering"]}

{"predictions": ["Social\_event"]}

{"predictions": []}

…

Note that the output file must have the same sentence order of the original test file! Our grading will be based on the rankings of your overall micro F1 scores.

# 5 Code Resources

* + OpenPrompt <https://github.com/thunlp/OpenPrompt>
	+ Code Skeleton: <https://github.com/zhangzx-uiuc/PromptED/blob/main/identification.py>

# 6 Submission

Please name your submission as `netid\_assignment2.zip`, including a report named as `netid\_report.pdf` and the code `netid\_code`. The code should include a README.md with environment and running instructions and at least an “output.json” file for your predictions on the test set.

Please submit your assignment on Canvas Assignment 2.

# Grading

1. **(8pt)** Build an event detection (identification) model based on prompt learning on the top 10 frequent event types in MAVEN.
	1. (1pt) Design your template.
	2. (2pt) Design your verbalizers.
	3. (3pt) Implement your loss function, which should be bug-free for training the model.
	4. (2pt) Implement your prediction function.
	5. (1pt) Write code to dump out your results on the test set.
2. **(2pt)** Performance Ranking: this 2pts are based on the rankings of your performance on the blind test set among the class.
3. **(2pt)** Writeclear and informative written report about your methods, results and findings.
	1. **(2pt)** Error analysis: Looking into the prediction errors of validation set and analyze the possible reasons for the wrong predictions (and think of how to improve the model).
4. **(2pt)** Writeclear README of the submitted code, including environment and running instructions.
5. **(Bonus, 2pt)** Experimenting on **all 168 event types** in MAVEN and generate another prediction file “*output\_full.json*”.
6. **(Bonus, 5pt)** Thinking of possible improvements of your system and write or implement your new ideas. For example, using AutoPrompt(<https://arxiv.org/pdf/2010.15980.pdf> ) or soft prompts instead of manually created templates.