Prompt Learning for Event Detection

Sharvi Tomar
stomar2@illinois.edu

October 8, 2022

1 Overview

This document reports on the sentence-level Event Detection System that I developed based on prompt learning. I experimented on designing different prompt templates, verbalizers, and loss functions to improve the event detection performance.

2 Introduction

2.1 Event Detection

Event Detection (ED) is a traditional Information Extraction task which includes two consecutive steps:

1. Identification: Given an 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.)

2.2 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.

2.3 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)

3 Dataset

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: link.

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: Catastrophe: which is triggered by data[“tokens”][19:20] -> “casualties”. 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.

4 Tasks

4.1 Designing the template

The first thing I did is to design the template for prompt learning. For event detection, a good template could be “[Sentence]. This sentence describes a [MASK] event.” (which is used by PILED).

4.2 Designing the 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 straightforward way is to directly use the event types’ names (Catastrophe, Conquering, etc.) as verbalizers. In addition to this, I added trigger words for each label as verbalizers. You can also design your own verbalizers.

4.3 Designing the 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. I have implemented the loss function similar to the one designed by the authors of PILED.

4.4 Writing the prediction function

The most straight-forward way for prediction is to select out the indices with maximum of logits. Since this is a multi-label classification problem and each sentence could have multiple predicted event indices, therefore, a suitable threshold value is to be chosen for prediction with good performance. I used the None event (index 0) as the threshold as what PILED does.

4.5 Training the model & dumping out the predictions on the test dataset

I wrote out all my 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": []

. . .

5 Performance Analysis

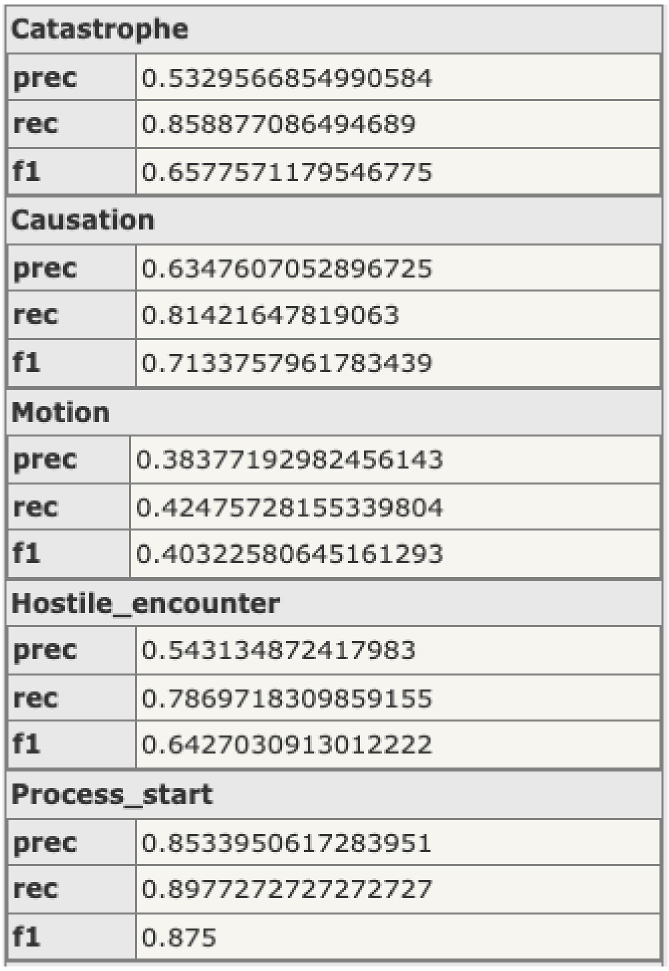
The following results were obtained after fine-tuning *openprompt's* ‘PromptForClassification‘ model for 5 epochs on the training set of MAVEN_DATA.

The output file has the same sentence order as the original test file.

Precision, Recall & Overall micro F1 scores.

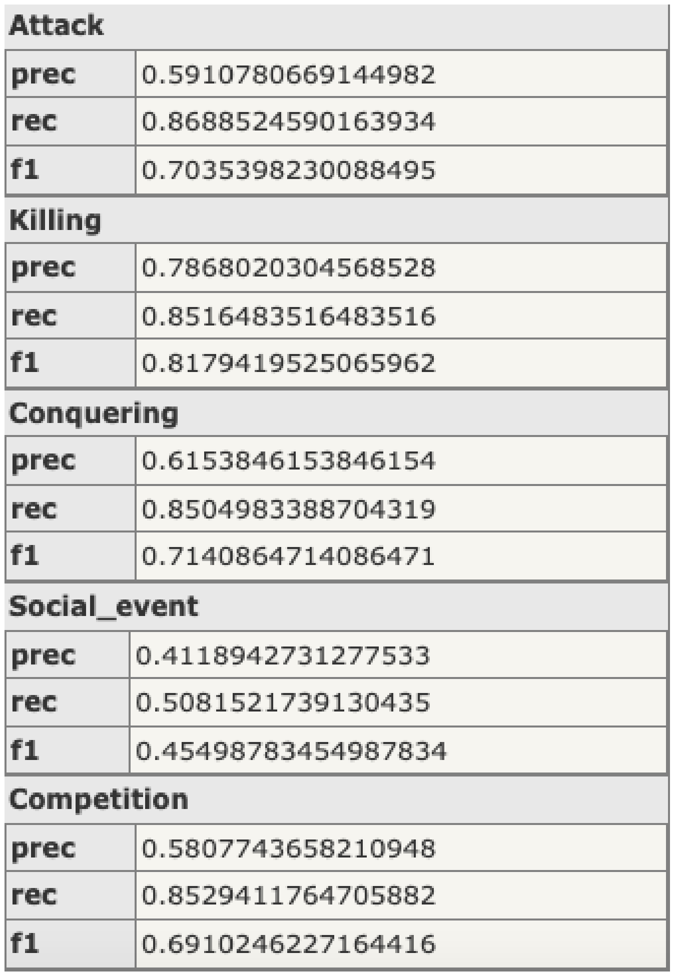
The following table shows the Label-wise performance of the model with precision, recall and F1-score per label.

Model performs the best for label: ’Process_start’ with the highest F1-score of 0.875 while it performs the least on the label: ’Motion’ with the lowest F1-score of 0.403. This gap in performance could be because of clear and distinct label characterisitc and label imbalance in dataset.



Catastrophe	
prec	0.5329566854990584
rec	0.858877086494689
f1	0.6577571179546775
Causation	
prec	0.6347607052896725
rec	0.81421647819063
f1	0.7133757961783439
Motion	
prec	0.38377192982456143
rec	0.42475728155339804
f1	0.40322580645161293
Hostile_encounter	
prec	0.543134872417983
rec	0.7869718309859155
f1	0.6427030913012222
Process_start	
prec	0.8533950617283951
rec	0.8977272727272727
f1	0.875

Figure 1: Label-wise results



Attack	
prec	0.5910780669144982
rec	0.8688524590163934
f1	0.7035398230088495
Killing	
prec	0.7868020304568528
rec	0.8516483516483516
f1	0.8179419525065962
Conquering	
prec	0.6153846153846154
rec	0.8504983388704319
f1	0.7140864714086471
Social_event	
prec	0.4118942731277533
rec	0.5081521739130435
f1	0.45498783454987834
Competition	
prec	0.5807743658210948
rec	0.8529411764705882
f1	0.6910246227164416

Figure 2: Label-wise results