Assignment 1: Fine-grained Name Tagging

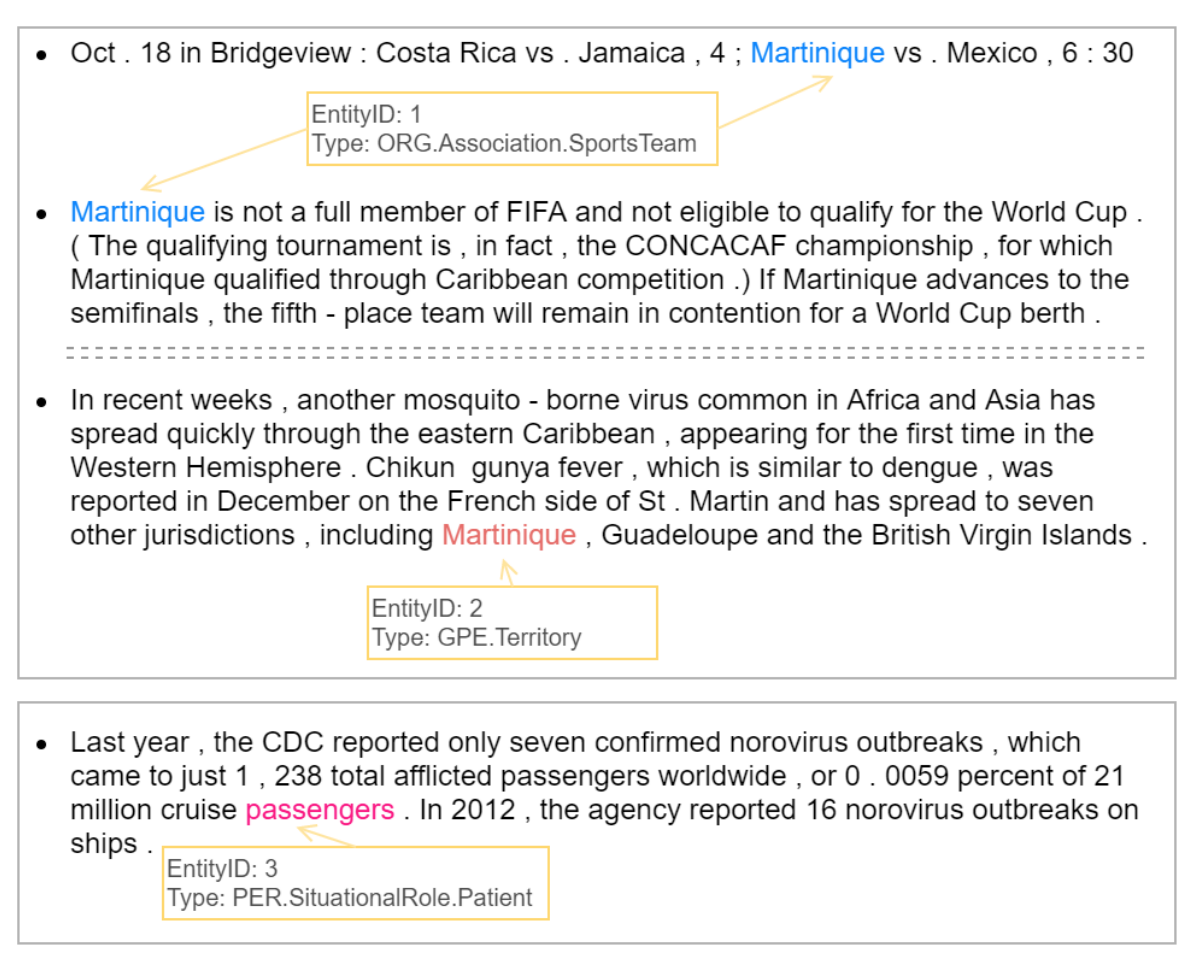
Due: September 29, 2021

Version History

* (current) 0.3:
  + Added: baseline code for the supervised setting
  + To be updated: where to submit
* 0.2:
  + Added: An evaluation script for self-evaluation. Please USE THIS SCRIPT instead of the one in their Github for reporting scores because the computation for few-shot setting is different. Added clarification on the leaderboard.
  + To be updated: where to submit, starting code for supervised setting, grading criteria for ranking
* 0.1:
  + Added: introduction, tasks and resources.
  + To be updated: evaluation scripts, where to submit, starting code for supervised setting, grading criteria for ranking

1. Introduction

The goal of name tagging is to extract mentions referring to named entities in a given textual document. This task is particularly challenging for fine-grained entity types. Input/Output example:



In this assignment we will aim to develop a name tagger for a target fine-grained entity ontology. In addition to the supervised learning setting, we will also explore a “N-way K~2K-shot” setting for few-shot name tagging. The dataset is available at:

* <https://ningding97.github.io/fewnerd/>

The Github repository <https://github.com/thunlp/Few-NERD> also includes helpful scripts for you to download and process the data, as well as baseline models you can start with. Please follow the training/development/test split as specified in the dataset.

1. Tasks and Grading (9 points + 2 bonus points):

(1) [Required; 3 points with complete submission] Few-NERD (SUP): supervised model, training data available for the target ontology.

(2) [Required; 2 points with complete submission] Few-NERD (INTRA): Few-shot setting (N-way K~2K-shot), you can explore transfer learning by using the training data within coarse-grained types.

(3) [Optional; 2 extra points with complete submission] Few-NERD (INTER) Few-shot setting (N-way K~2K-shot), you can explore transfer learning by using the training data across coarse-grained types.

(4) [Required; 2 points] After you finish developing the system, submit it to the leaderboard, 0-2 points will be assigned based on the system's relative rank in the class.

(5) [Required; 2 points] Write an informative report about what you have done, and your findings. 0-2 extra points will be assigned to informative analysis (what has worked and remaining challenges).

1. Detailed Instructions

General submission requirements:

* For all tasks, please submit **your source code including a shell script named “run\_task\_{task\_number}.sh”** that can directly train a model and run evaluation on the test set. After running the script, you should have an output file generated (details specified below), which will be compared with the corresponding test file for the evaluation. All the tasks are evaluated using the F1 score, however we recommend you to compute precision and recall as well in order to better understand your outputs.
* For all tasks, you need to submit the output files and also report the score yourself in order to get familiar with how to evaluate the models for these tasks.
* Please submit a report (MS Word Document or PDF). In addition to describing what you have implemented, it is important to include analysis on your results.
* Since the test set is already public available, it seems not making sense to hold a leaderboard. Therefore, we provide a script for you to evaluate the performance of your models. We will be using the same script to grade your assignment.

An example submission folder should contain:

* File: report.pdf/docx, including the scores for all tasks
* File: src/run\_task\_{i}.sh, i=1,2, (optional) 3
* File: src/outputs/sup/test.txt
* File: src/outputs/intra/test\_{N}\_{K}.jsonl, N=5, 10, K=1, 5
* (Optional) File: src/outputs/inter/test\_{N}\_{K}.jsonl, N=5, 10, K=1, 5
* Other source code: src/\*

Don’t include your trained models for submission, but do save copies of them. We may ask for your checkpoints if we cannot reproduce your results.

1. Few-NERD (SUP):

This is the common supervised learning setting. You need to download Few-NERD (SUP) from the website or run the downloading scripts in the Github repository

|  |
| --- |
| bash data/download.sh supervised |

You will get three .txt files for training, validation and evaluation respectively. Your script should generate a “test\_output.txt” file with the same format as the “text.txt” file you downloaded.

Although the supervised setting is the simplest setting, the official Github repository doesn’t include the implementation for the supervised models yet. You need to setup your own system yourself for this task.

1. Few-NERD(INTRA):

In this task, we will develop a few-shot tagger for the N-way K~2K-shot setting. We will only give a brief explanation for the setting here. If you are not familiar with this setting, please refer to the dataset paper for more details. In brief, instead of training on each individual sentences for the tagging, you need to train a model on a set of episodes. In each episode, your model is given a few sentences, including K~2K mentions for each of the N entity types as the support set. Then your model needs to tag a few query sentences. During evaluation, your model runs on another set of episodes and the average F1 score is reported. Note that in this assignment, we require you to report the average F1 score over episodes, instead of over mentions as in the official repository.

For both this task and the INTER task, You need to download the “Sampled Few-NERD (568M)” from the website, and report results on the test set “test\_{N}\_{K}.jsonl” for fair comparison. However, you don’t have to use the sampled training set. For example, you can download “Few-NERD(INTRA)” and sample the training set yourself. Your final outputs should be of the same format as “test\_{N}\_{K}.jsonl”.

In the Github repository, there are already three baseline models provided for this setting. “train\_demo.py” is the training script and also runs evaluation in the end of training. We recommend you using their script for evaluation. You can find their model architectures under the “. /model” directory. You may also create your model there and integrate it into the training script, which will make it easier to run the training and evaluation of your own model.

1. Few-NERD(INTER):

This task is almost identical to the second task, except that the dataset splits are different. So please follow the same instructions for the INTRA task.

3. Some baseline systems that you can consider using as a starting point:

(1) Code for this leaderboard paper is available in the leaderboard: <https://arxiv.org/abs/2105.07464>

(2) [Lin et al., ACL2019]: you will need to map the types to the target ontology:

<https://github.com/limteng-rpi/fet>

(3) Use OneIE to acquire coarse-grained mentions:

https://blender.cs.illinois.edu/software/oneie/

4. Suggestions for possible enhancement:

(1) Leverage more training data from other shared tasks, such as RUFES: <https://blender.cs.illinois.edu/paper/kbp2020overview.pdf>

A list of publicly available data sets is summarized at: <https://tac.nist.gov/2020/KBP/RUFES/guidelines/KBP2020TaskSpec_V1.0.pdf> .

We also have prepared a dataset here: <https://drive.google.com/file/d/1lAmwEKuEwZv1ZlVyAE6-F9BHx7rloSIW/view?usp=sharing>

(2) Adding linguistic features based on gazetteers, knowledge-empowered embedding, etc.

(3) Adding architecture improvement into sequence-to-sequence models.