eHealth-KD Challenge 2020

Docs, data and development scripts for the eHealth-KD Challenge at IberLEF 2020

Submission details

The challenge will be graded on Codalab.

A fully working evaluation script is provided to participants, that exactly matches the evaluation formulas used in Codalab. This way participants will have the possibility to evaluate their systems offline and perform hyper-parameter tuning with respect to the same evaluation metrics as used in the competition.

Baseline implementation

A baseline system is provided for participants to compare their results. If necessary, feel free to use the baseline as a starting point for developing your own solution, since the baseline already covers parsing the input and generating output in the correct format.

The baseline implementation is an extremely basic strategy that simply stores all the training, and at test time outputs keyphrases and relations if they exactly match something found in the training.

We recommend all participants to first run the baseline implementation (with the training and development sets) and upload it to Codalab, to get acquainted with the submission process. The following instructions detail this process.

Running the baseline implementation on the development set

The first step consists in downloading the project and running the baseline implementation.

Clone the `ehealthkd-2020` project from Github:

```
$ git clone https://github.com/knowledge-learning/ehealthkd-2020.git
```

Run the baseline implementation for the main scenario. The baseline implementation is in the `scripts/baseline.py` file. The arguments are:

- `--train` to run the baseline on the training collection.
- `--dev` to run the baseline on the development collection.
- `--test` to run the baseline on the test collection.
- `--custom GOLD MODE SCENARIOS` to run the baseline on the `GOLD` collection, across a list of comma-separated `SCENARIOS`, and labeling the output in the submit directory as `MODE`.
  - The configuration `--train` is equivalent to:

    ```
    --custom data/training/scenario.txt train \
        scenario1-main,scenario2-taskA,scenario3-taskB
    ```

  - The configuration `--dev` is equivalent to:

    ```
    --custom data/development/main/scenario.txt dev    \
        scenario1-main,scenario2-taskA,scenario3-taskB  \
    --custom data/development/transfer/scenario.txt dev  \
        scenario4-transfer
    ```

  - The configuration `--test` is equivalent to:

    ```
    --custom data/testing/{0}/scenario.txt test \
        scenario1-main,scenario2-taskA,scenario3-taskB,scenario4-transfer
    ```

  - Subtask A and Subtask B will not be run at scenarios labeled as `scenario3-taskB` and `scenario2-taskA` respectively (nor any other scenario that ends with `-taskB` and `-taskA` respectively).

> The module `scripts.submit` contains the utilities used by the baseline to handle the submission format.

In this case we will train with the `training` set (800 sentences) and evaluate on the `development` set (200 sentences) using the same sentences for the 3 evaluation scenarios. However, in the final TEST phase you will train with both training and development and evaluate on the corresponding test sets (different for each scenario).

Here is a baseline execution example:

```
$ cd ehealthkd-2020
# Inside the root folder ehealthkd-2020
$ python3 -m scripts.baseline --dev
```

Then, you can go to `data/submissions/baseline/dev/` and check the corresponding files were generated:

```
ls -lR data/submissions/baseline/dev/
data/submissions/baseline/dev/:
total 4
drwxr-xr-x 6 user user 4096 Jan 29 15:42 run1

data/submissions/baseline/dev/run1:
total 16
drwxr-xr-x 2 user user 4096 Jan 29 15:42 scenario1-main
drwxr-xr-x 2 user user 4096 Jan 29 15:42 scenario2-taskA
drwxr-xr-x 2 user user 4096 Jan 29 15:42 scenario3-taskB
drwxr-xr-x 2 user user 4096 Jan 29 15:42 scenario4-transfer

data/submissions/baseline/dev/run1/scenario1-main:
total 84
-rw-r--r-- 1 user user 64650 Jan 29 15:42 scenario.ann
-rw-r--r-- 1 user user 19060 Jan 29 15:42 scenario.txt

data/submissions/baseline/dev/run1/scenario2-taskA:
total 72
-rw-r--r-- 1 user user 53050 Jan 29 15:42 scenario.ann
-rw-r--r-- 1 user user 19060 Jan 29 15:42 scenario.txt

data/submissions/baseline/dev/run1/scenario3-taskB:
total 72
-rw-r--r-- 1 user user 52414 Jan 29 15:42 scenario.ann
-rw-r--r-- 1 user user 19060 Jan 29 15:42 scenario.txt

data/submissions/baseline/dev/run1/scenario4-transfer:
total 0
-rw-r--r-- 1 user user 0 Jan 29 15:42 scenario.ann
-rw-r--r-- 1 user user 0 Jan 29 15:42 scenario.txt
```

> ⚠️ Make sure that your files are named *exactly* as the files above, since the evaluation script in Codalab will expect these filenames.

> ⚠️ Also make sure that you have the file `scenario.txt` with the input sentences in your submission folder. This is the *exact* same file you processed as input, so you can just copy and paste it. The baseline script already handles this. This is necessary for the evaluation script to guarantee that you have the right sentences.

Evaluating a single scenario

Now you can run the evaluation script offline just to check your results. The evaluation script is in the file `scripts/score.py` and the arguments are:

- The gold annotations (in this case, `data/development/main/scenario.txt`).

- Your system’s annotations (`data/submissions/baseline/dev/run1/scenario1-main/scenario.txt`)

The evaluation script outputs the total number of correct, incorrect, partial, missing and spurious matches for each subtask, and the final score as defined in the Task section.

```
$ python3 -m scripts.score \
    data/development/main/scenario.txt \
    data/submissions/baseline/dev/run1/scenario1-main/scenario.txt

correct_A: 813
incorrect_A: 75
partial_A: 65
spurious_A: 674
missing_A: 352
correct_B: 102
spurious_B: 256
missing_B: 1102
--------------------
recall: 0.3776
precision: 0.4773
f1: 0.4217
```

> **NOTE:** The exact numbers you see with the baseline may vary, as the evaluation script and/or the baseline implementation can suffer changes as we discover bugs or mistakes. These numbers are for illustrative purposes only. The actual scores are the ones published in Codalab.

The options `--skip-A` and `--skip-B` instruct the script to ignore the performance of the submission on subtask A and subtask B respectively (i.e. they will not directly impact the final score reported).

You can evaluate just scenario 2 with the evaluation script by passing `--skip-B` :

```
$ python3 -m scripts.score --skip-B \
    data/development/main/scenario.txt \
    data/submissions/baseline/dev/run1/scenario2-taskA/scenario.txt

correct_A: 813
incorrect_A: 75
partial_A: 65
spurious_A: 674
missing_A: 352
--------------------
recall: 0.6479
```

```
precision: 0.5197
f1: 0.5767
```

You can evaluate just scenario 3 with the evaluation script by passing `--skip-A`:

```
$ python3 -m scripts.score --skip-A \
    data/development/main/scenario.txt \
    data/submissions/baseline/dev/run1/scenario3-taskB/scenario.txt

correct_B: 107
spurious_B: 91
missing_B: 1097
--------------------
recall: 0.08887
precision: 0.5404
f1: 0.1526
```

Additionally, you can pass `--verbose` if you want to see detailed information about which keyphrases and relations were correct, missing, etc.

```
$ python3 -m scripts.score --verbose \
    data/development/main/scenario.txt \
    data/submissions/baseline/dev/run1/scenario1-main/scenario.txt

===================  MISSING_A   ===================

Keyphrase(text='enfrentar', label='Action', id=3)
Keyphrase(text='tubos', label='Concept', id=7)
Keyphrase(text='filtran', label='Action', id=10)
Keyphrase(text='limpian', label='Action', id=11)
Keyphrase(text='eliminando', label='Action', id=13)

... LOTS OF OUTPUT

===================  MISSING_B  ===================

Relation(from='producen', to='genes', label='subject')
Relation(from='producen', to='proteínas', label='target')
Relation(from='producen', to='correctamente', label='in-context')
Relation(from='trastorno', to='niño', label='target')
Relation(from='trastorno', to='genético', label='in-context')
Relation(from='producen', to='trastorno', label='causes')
Relation(from='producen', to='trastorno', label='causes')
--------------------
recall: 0.3776
precision: 0.4773
f1: 0.4217
```

Evaluating all scenarios

You can also evaluate all runs in every scenario. The evaluation script is in the file `scripts/evaltest.py` and the mandatory arguments are:

- The evaluation mode (`dev` or `test` for development and test evaluation respectively).
- The path to the submissions folder is set by default to `data/submissions` . This is the folder of all participants, or, if `--single NAME` is passed, directly the folder of the participant identified by `NAME` inside `data/submissions` . Each participant's folder contains subfolders with runs.

```
$ python3 -m scripts.evaltest --single baseline --mode dev --plain

==================================================
:::::::::::::::::::: BASELINE ::::::::::::::::::::
==================================================
---------------------[ run1 ]---------------------
> scenario1
     correct_A        = 813
     incorrect_A      = 75
     partial_A        = 65
     spurious_A       = 674
     missing_A        = 352
     correct_B        = 102
     spurious_B       = 256
     missing_B        = 1102
     recall           ~ 0.3776
     precision        ~ 0.4773
     f1               ~ 0.4217
> scenario2
     correct_A        = 813
     incorrect_A      = 75
     partial_A        = 65
     spurious_A       = 674
     missing_A        = 352
     correct_B        = 0
     spurious_B       = 0
     missing_B        = 1204
     recall           ~ 0.6479
     precision        ~ 0.5197
     f1               ~ 0.5767
> scenario3
     correct_A        = 1305
     incorrect_A      = 0
     partial_A        = 0
     spurious_A       = 0
     missing_A        = 0
     correct_B        = 107
     spurious_B       = 91
     missing_B        = 1097
     recall           ~ 0.08887
     precision        ~ 0.5404
     f1               ~ 0.1526
> scenario4
     correct_A        = 0
     incorrect_A      = 0
     partial_A        = 0
     spurious_A       = 0
     missing_A        = 0
     correct_B        = 0
     spurious_B       = 0
     missing_B        = 0
     recall           ~ 0.0
     precision        ~ 0.0
     f1               ~ 0.0
```

Running the baseline on the test set

Once the test set input files are released, you will be able to test the baseline implementation on the test set as well. Please read the details about the test set structure.

These are the necessary steps:

Run the baseline on all test scenarios (scenario1 may take a couple minutes):

```
$ python3 -m scripts.baseline --test
```

> ⚠️ Remember that for scenario 3 the file `scenario.ann` must contain **a copy** of the gold annotations provided in the file `data/testing/scenario3-taskB/scenario.ann`, plus your own relation annotations. The baseline already does this, but ensure your own implementation takes it into consideration.

> ⚠️ When submitting to subtask B, make sure to **reuse the keyphrase ID** provided in the `scenario.ann` from the gold annotations. The baseline implementation already takes care of this detail.

Once finished, you can submit your results to Codalab.

Remember that for the duration of the challenge the results for the test set will be hidden and only shown after the competition ends.

However, you will receive error notifications if your upload is invalid. You have up to **100** different submissions.

Submitting your results to Codalab

The eHealth-KD evaluation environment can be accessed at this link. Once you have all the corresponding outputs, please bundle the content of the submit folder in a `.zip` file:

```
$ cd data/submissions/<team>
$ zip -r <team>.zip *
```

Where `<team>` is the name of the folder where your submission is stored.

> ⚠️ Make sure you zip **the content** of the `submission/<team>` folder, and not the `submission` folder *itself*. When in doubt, `cd` into `data/submission/<team>` and run `zip` there. The idea is that the root of your `submission.zip` file should directly contain the two folders `dev` and `test` and **not** a `submission` folder or a folder with your team's name.

Structure of the submit folder

For recap here is the expected structure of the `submission.zip` file:

- **Folder** `dev`:
  - **Folder** `run1`:
    - **Folder** `scenario1-main`:
      - **File** `scenario.ann`: Your output for subtask A and B.
      - **File** `scenario.txt`: Sentences, copied verbatim from input.
    - **Folder** `scenario2-taskA`:
      - **File** `scenario.ann`: Your output for subtask A.
      - **File** `scenario.txt`: Sentences, copied verbatim from input.
    - **Folder** `scenario3-taskB`:
      - **File** `scenario.ann`: Output for subtask A, copied verbatim from input, and your output for subtask B.
      - **File** `scenario.txt`: Sentences, copied verbatim from input.
    - **Folder** `scenario4-transfer`:
      - **File** `scenario.ann`: Your output for subtask A and B.
      - **File** `scenario.txt`: Sentences, copied verbatim from input.
  - **Folder** `run2`: Optional additional run with the same format
  - **Folder** `run3`: Optional additional run with the same format
- **Folder** `test`:
  - **Folder** `run1`:
    - **Folder** `scenario1-main`:
      - **File** `scenario.ann`: Your output for subtask A and B.

        - **File** `scenario.txt` : Sentences, copied verbatim from input.
      - **Folder** `scenario2-taskA` :
        - **File** `scenario.ann` : Your output for subtask A.
        - **File** `scenario.txt` : Sentences, copied verbatim from input.
      - **Folder** `scenario3-taskB` :
        - **File** `scenario.ann` : Output for subtask A, copied verbatim from input, and your output for subtask B.
        - **File** `scenario.txt` : Sentences, copied verbatim from input.
      - **Folder** `scenario4-transfer` :
        - **File** `scenario.ann` : Your output for subtask A and B.
        - **File** `scenario.txt` : Sentences, copied verbatim from input.
    - **Folder** `run2` : Optional additional run with the same format
    - **Folder** `run3` : Optional additional run with the same format

The `dev` folder is optional and will only be checked if present. During the **training** phase in Codalab (i.e., before the test set is released) you can skip the `test` folder. During the **test** phase in Codalab, both `dev` and `test` will be checked if present. The results for the `dev` folder will always be published in Codalab (during the **training** and **test** phases) but the results for the `test` folder will be kept hidden until the competition ends.

> ⚠️ **You can submit up to 100 times, but only the last submission sent will count towards the competition evaluation.**

You can submit up to **three (3) runs** inside each of the `dev` and `test` folders. More runs will be considered and error, and less runs will only show a warning. You can use these three runs to try different approaches. Do not use them for fine-tunning hyperparameters.

> ⚠️ Please double-check the files for all four scenarios, including the `scenario.txt` files. If you do not plan to participate in any given scenario, kindly reuse the baseline output then, to avoid the evaluation script from raising errors about missing files.

Uploading your results to the competition server

> **NOTE**: The Codalab competition is still **not available**. The instructions below are subject to possible changes once the competition opens.

Please also make sure to fill-in this Google Form to accept the license terms for the corpus.

Go to the Codalab competition page and register if you have not done so already. In Codalab, go to the Participate section and enter the details of your submission:

- **Team name.**
- **Method name:** a short, memorable name for the technique you are presenting.

- **Method description:** refers to the type of techniques used. Please write a summary (~200 words) of the techniques, algorithms or approaches used. Also specify if you use external sources (other corpora, knowledge bases, etc.). Finally, attach one or more of the following tags regarding techniques and/or resources used in your approach. These tags will help us better understand which approaches are more popular or perform better in this task in the future.
    - **K:** knowledge-bases
    - **S:** Shallow supervised methods (i.e., *logistic regression*, *SVM*, *Markov models*, *CRF*, …)
    - **D:** Deep supervised methods (e.g, *CNNs*, *LSTMs*, …)
    - **U:** Unsupervised methods (e.g. clustering or dimensionality reduction techniques, …)
    - **E:** Embeddings (e.g., *word2vec*, *BERT*, *ELMo*, …)
    - **N:** Standard NLP techniques (pos-tagging, *AMR* parsing, dependency parsing, *NER*, …)
    - **R:** Hand-crafted rules

Finally hit the submit button and attach you zip file. If everything is ok, after a few seconds hit the **Submit to leaderboard** button at the bottom of the page to see your results.

Final words

> **DISCLAIMER:** The scoring you achieve during the training phase is only for your own reference, and should not be taken as an indication that you will achieve a similar score in the test phase. Particularly, participants that achieve the highest scores in the training phase are **not guaranteed** to win in the test phase, since participating in the training phase is completely optional. Likewise, at any point we may decide to change the evaluation script, including during the blind test phase, if we discover any kind of bug or error. We will inform you if that's the case and provide an updated evaluation script.

Finally, if you discover a mistake in the evaluation script, please let us know at ehealth-kd@googlegroups.com or post an issue on our Issues Page in Github.
