On Guaranteed Optimal Robust Explanations for NLP Models - Code Reproducibility - IJCAI 2021

anonymous submission

January 2021

1 Preliminary

This code has been successfully tested on Linux (Fedora 30+32, Ubuntu 18.04). We use Python - at least 3.8 is suggested - and the *package installer for Python*, a.k.a. pip to manage dependencies - we tested our code with pip 19.3.1.

In order to run the code of the experiments, you need to download the supplementary material (.zip file attached to the submission) and unzip it. Once unzipped, you should find a folder named `OREs_anonym` that contains the code necessary to replicate a few simulations.

1.1 Python Dependencies

OREs code depends on Python (we use version 3.8). It also requires the following packages (the versions reported are what we use):

- numpy=1.18.5
- python-sat
- tensorflow=2.3.0
- Keras=2.3.1

In order to install the dependencies, you can run the command `pip install PACKAGENAME==X.Y.Z --user` where `PACKAGENAME` is the name of the missing dependency and `X.Y.Z` is the version, for example, `pip install numpy==1.18.5 --user`. If no specific version is specified in the previous list, you can run `pip install PACKAGENAME --user`.

1.1.1 External Dependencies

We used the version of Marabou with commit hash `228234ba` and its Python API[1]: just clone into `OREs_anonym` (i.e., at the same level of `Explanations`

[1]Please note that we couldn't insert Marabou directly as it is an external tool with its own specific Licence and it has to be installed on the system

folder) the files at the following link (`https://github.com/NeuralNetworkVerification/Marabou/`). Once installed (the Marabou's Github folder has instructions on how to install the package) you should have `Marabou` and `Explanations` as sub-directories of the `OREs_anonym` folder.

Run Experiments

To launch an Optimal Robust Explanation (ORE) experiment, go to the `Explanations\abduction_algorithms\experiments\{DATASET}\` folder, where {DATASET} is either 'SST' 'Twitter' or 'IMDB', and run one of the python files (`.py` extension). We report a few usage examples.

ORE for a Fully Connected Model on a Sample Review

Let's suppose that you want to extract an ORE from a sample review and you are using an FC model, trained on SST dataset, with 25 input words and using the k Nearest Neighbours bounding box technique with k=10. You have to navigate to `Explanations\abduction_algorithms\experiments\SST\` folder and run the following:

```
python3 smallest_explanation_SST_fc_knn_linf.py -k 10 -w 5 -n 25
 -i 'This is a very bad movie'
```

The algorithm identifies the words *is* and *bad* as an ORE. Additionally, results are logged and stored in a folder inside `results\HS-clear` (the complete path that is reported in the logs of the execution).

If you want to obtain the same result but using Minimum Satsfying Assignment algorithm:

```
python3 smallest_cost_explanation_SST_fc_knn_linf.py -k 10 -w 5
 -i 'This is a very bad movie'
```

Solving an hard instance with Adversarial Attacks

Let's suppose that you want to extract an ORE from a Twitter text and you are using a CNN model, trained on Twitter dataset, with 25 input words and using the k Nearest Neighbours bounding box technique with k=8. You can improve the convergence on hard instances with Adversarial Attacks by simply specifying the number of attacks that you want to launch (parameter -a/–adv). You have to navigate to the `Explanations\abduction_algorithms\experiments\Twitter\` folder and run the following:

```
python3 smallest_explanation_Twitter_cnn2d_knn_linf.py -k 8 -w 5
 -n 25 -a 500 -i 'Spencer is not a good guy'
```

The algorithm identifies the words *Spencer* and *not* as an ORE (alongside a *PAD* token that highlights a problem with the model robustness). Again,

results are stored in a folder inside `results\HS-clear` (the complete path that is reported in the logs of the execution).

Detecting Decision Bias Using Cost Functions

Let's suppose you want to check whether an explanation will always contain the name of a character, actor or director: something which ideally should not be used for classifying whether a review is positive or negative. To do this, you can run the following command:

```
python3 smallest_HScost_explanation_SST_fc_knn_linf.py -k 27 -w 5
-a 0 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end" -u False -x 'austin,powers,goldmember'
```

If the excluded word still appears in the ORE (which it does in this example, "powers" is present), you know that no explanation exists without it and therefore that there is a decision bias here.

If you want to do the same but using MSA algorithm:

```
python3 smallest_cost_explanation_SST_fc_knn_linf_alternate_cost.py
 -k 27 -w 5 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end"
```

If you want to permanently exclude the word from the explanation use:

```
python3 smallest_cost_explanation_SST_fc_knn_linf_alternate_cost_exclude.py
-k 27 -w 5 -i "Austin Powers in Goldmember has the right stuff for
summer entertainment and has enough laughs to sustain interest to
the end"
```

2 Run multiple instances in parallel

You can run several ORE instances in parallel (e.g., on a server) by launching the `run-exp_MODEL_DATASET_knn_linf.sh` script in each `Experiments\DATASET` folder, where DATASET is either 'IMDB', 'SST' or 'Twitter' and MODEL is either 'fc' or 'cnn'[2].

[2]This requires the `screen` bash command (available on any Linux distribution) to manage the various instances.