On the implementation of baselines and Lightweight Conditional Model Extrapolation (LIMES) for class-prior shift

Paulina Tomaszewska[1][0000−0001−6767−1018] and Christoph H. Lampert[2][0000−0001−8622−7887]

[1] Warsaw University of Technology, Faculty of Mathematics and Information Science, Warsaw, Poland

[2] Institute of Science and Technology Austria (ISTA), Machine Learning and Computer Vision Group, Klosterneuburg, Austria

Abstract. This paper focuses on the implementation details of baseline methods and a recent lightweight conditional model extrapolation algorithm *LIMES* [5] for streaming data under class-prior shift. *LIMES* achieves superior performance over the baseline methods, especially concerning the average-of-minimum accuracy metric, which is important for the users of the system. In this work, the key measures to facilitate reproducibility and enhance the credibility of the results are described.

Keywords: continual learning · non-stationary data · class-prior shift.

1 Introduction

Offline Deep Learning based solutions work well in production as long as the input at the prediction time has similar characteristics to the data used for training. Otherwise, if the discrepancy is substantial, the model may need to be updated by fine-tuning using new data or changing the model architecture by adding new outputs which also requires some training. Another approach is to replace the previous model with a new one.

The relatively new paradigm is to train a model in a continuous manner on streaming data. However, in such a case, often the data drift problem arises. It may happen that at the production time, data from the new class occurs, features within the classes change or the class-prior shift happens. In the work, the latter issue is discussed mainly from the implementation standpoint. Following [5], the following algorithms are analyzed: *LIMES*, *incremental*, *random*, *ensemble* and *restart*. The key aspects of the recently proposed *LIMES* method are adaptation at the training time (inspired by meta-learning solutions) and extrapolation of incoming data class distribution. Their implementation is publicly available at github repository `https://github.com/ptomaszewska/LIMES` to facilitate results reproduction.

2 Dataset

In Machine Learning data is a start point of any research. That is why this is so crucial to distribute it (even in an anonymized manner) especially if the data is not artificially generated [2, 4]. Following this good practice, the exemplary `geo-location` dataset used in [5] to compare performance of different methods, was planned to be shared in an original format. However, Twitter's free Streaming API[3], which was used for data collection, states in its Terms of Service that it is not allowed to make large amounts of raw Twitter data available on the Web. In such a case, only the so called *dehydrated* dataset was placed at the website[4] provided in the paper. The dehydrated dataset means that json files with text of tweets and metadata (downloaded using API) are compressed to only tweet ids. Later, the original dataset in a form of json files can be restored using *Twarc* tool[5]. To get started with the tool, the configuration need to be done where the user has to provide information about Twitter application API keys and grant access to one or more Twitter accounts.

The process of restoring the tweets from their ids can be done with a single line of code: `twarc hydrate tweet_ids.txt > tweets_hydrated.jsonl`. This process is called *rehydration*. The dehydrated dataset on the webpage is stored in two zip files, separately for each time subset: 1-5 days and 11-15 days. Each zip contains files in a format `geo-YYYY_MM_DD_HH.id` where tweet ids from one hour HH of particular day YYYY-MM-DD are available. The size of each zip file is of about 500 MB. However, please note that after rehydration, the size increases about 200 times.

3 Implementation

The source code available at github repository[6] is under the Permissive MIT license (following guidelines in [4]) that puts only very limited restriction on reuse. While writing the implementation we followed the good practices described in [4] like modularity - there are classes and functions that facilitate understanding by using informative variable names. The repository contains python scripts for preprocessing of json files containing Twitter data, creating embeddings and implementation of the methods for class-prior shift. In addition, `requirements.txt` to facilitate the setting up of environment for the sake of experiments and the configuration scripts to run the experiments on SLURM queuing system[7] are provided. This is a step forward the high level of reproducibility defined in [4]. Due to large size of the dataset, it may be necessary to use cluster for an increased efficiency.

[3] https://developer.twitter.com/en/docs/tutorials/consuming- streaming-data
[4] https://cvml.ist.ac.at/geo-tweets/
[5] https://twarc-project.readthedocs.io/en/latest/
[6] https://github.com/ptomaszewska/LIMES
[7] https://slurm.schedmd.com/documentation.html

3.1 Preprocessing of raw json files with Twitter data

At this stage, the raw data is processed to create files with a better defined structure so that the data will be easier for further use. The `process_raw_data.py` script is generic - it extracts more valuable information from metadata than for this research. Information about the country from which the particular tweet originates is extracted based on longitude and latitude coordinates. For this purpose, the `countries.geojson` file is required, which can be downloaded from this link[8]. Note that here the term country is considered in a broader sense following official codes in ISO 3166-1 where regions that are not independent but have their identity are also taken into account.

3.2 Embeddings

The following step in the algorithm is to use files generated in the previous stage to prepare input to the Machine Learning models. The jsonlines files are processed line by line. The two fields are considered: text and location. Each of them is passed to the pretrained `distiluse-base-multilingual-cased-v1` multilingual sentence embedding network[9] (suitable for social media texts) in order to create embeddings. At this stage, the train/validation split with ratio 80:20 is performed at random. Later, the concatenated features from two sources are appended to respected list with training or validation samples. Later, at training time, it is decided which data source to use. When all of records from the json file (regarding data from one hour) are processed, the lists of features are saved to files with the `.npy` extension. This file extension ensures binary format that results in smaller file size and more efficient data loading. In the script for generation of embeddings, the `country_codes.csv` file is used to decipher the code of the countries from the meta data. The file is provided in the repository. Later, the data stored in newly generated files is subsampled to create 10 realizations (script `subsample_realizations.py`)

3.3 Machine Learning models - training and evaluation

The implementation is written using *keras* framework with *tensorflow* backend. We prepared custom training loop were processing of streaming data is simulated - each sample is processed only once. The speed up the training, the training step was wrapped using `tensorflow.function` decorator. The default mode in Tensoflow 2 is eager execution which can slow down training but can be useful for the debugging. Instead, by applying the decorator, global performance optimizations can be performed because the model is treated as a static graph.

The algorithms (*LIMES*, *incremental*, *random*, *ensemble* and *restart*) are generic, however, the implementation is ready to use for the experiments on Twitter data to reproduce the results from the paper. The code can be adjusted

[8] https://github.com/datasets/geo-countries/blob/master/data/countries.geojson
[9] https://github.com/UKPLab/sentence-transformers

to other experiment settings like dataset mainly by modifying the dimensionality of the expected input and output of the respected layers. The first scenario is a reproduction and the second replication as specified in [4]. The pseudocode for the four methods is described in Algorithm 1. In order to implement a non-trainable bias correction term in a fully connected layer, the custom class was defined.

Algorithm 1 A generic training algorithm

```
$data\_source \leftarrow \{tweet, location, tweet + location\}$
$p \leftarrow U(0, 250)$
initiate model or models (collection of 24 models in case of *ensemble*)
**for** $counter \leftarrow 0$ to $all\_hours$ **do**
    **if** $counter > 0$ **then**
        $X\_val, y\_val \leftarrow load\_data(filenames\_val, counter)$
        **if** $LIMES$ **then**
            $p\_future \leftarrow extrapolation(all\_historical\_p)$
            $model \leftarrow model(p\_future)$                ▷ apply bias correction term
        **else if** $random$ **then**
            $p\_future \leftarrow random(all\_historical\_p)$
            $model \leftarrow model(p\_future)$                ▷ apply bias correction term
        **else if** $ensemble$ **then**
            $model \leftarrow models[counter]$
        **end if**
        $accuracy = validation(X\_val, y\_val, model)$
        **if** $restart$ **then**
            initiate *model*
        **end if**
    **end if**
    $X\_train, y\_train \leftarrow load\_data(filenames\_train, counter)$
    $y\_pred \leftarrow model(X\_train, y\_train)$
    $loss \leftarrow CrossEntropy(y\_pred, y)$
    update model weights using gradients based on loss
**end for**
```

To reproduce the paper results, it is recommended to run experiments with different combinations of parameters (subset, realization, data source, model).

3.4 Running experiments efficiently

Each of the steps (preprocessing, embedding generation, training) can be run with different parameters in automated way using bash scripts provided at the repository. One of those is `run_process_raw_data.sh` which is suitable to call a python script for data preprocessing iterating over different files. In analogous way, the python script for creating text embeddings can be executed. Running training of Machine Learning method can be done efficiently using two bash scripts from the repository (`run_sbatch.sh` and `run_training.sh`). In the first one, training with appropriate parameters is scheduled within loops iterating over different parameters of experiments. At this stage, the experiments are sent to the SLURM queue.

4 Reproducibility

The experimental process should be repeatable, yield consistent results and conclusions [3]. Therefore, in the source code, the random seed was set at different stages of the pipeline: when creating a training/validation split and training of the Machine Learning model (especially important for weight initialization). To facilitate the reproduction, the detail instructions on how to get the dataset, run the code and comment in the scripts are provided. Not only can the results of the experiments be reproduced but also the figures presented in the paper because the appropriate script is available at the repository.

To facilitate the reproduction of the results described in the papers, the authors sometimes provide Google Colaboratory notebooks. In our case, it is not done for several reasons. First of all, the data cannot be publicly shared in an original form, instead, it has to be rehydrated, which is not a straightforward task. It requires creating an account on Twitter application API required and providing personal keys to configure the *Twarc* tool. Moreover, the dataset is of significant size. Lastly, the computations take significant time so it may be better to run it on the cluster if possible.

5 Credibility of results

As stated in [1, 3], one of the common problems with credibility originates in selective reporting of results and overclaiming of the results, by drawing conclusions that go beyond the evidence presented (e.g. insufficient number of experiments). We have taken measures to eliminate these problems. We generated many dataset subsets - 10 different realizations of data, for 2 different time subsets (*"early"* and *"late"*) using 3 different data sources (*"tweet"*, *"location"* and concatenation of both). In addition, to analyze whether there is a correlation between the task difficulty and the performance boost of the proposed method, we investigated three different data sources. In order to reduce bias that could be introduced by the choice of the metric and show the broader spectrum of advantages and limitations of *LIMES*, two different metrics are computed - *avg-of-avg* and *avg-of-min* of accuracy values aggregated for 24 hours periods (in both cases mean and standard deviations are reported). The exact way in which the metrics are computed is provided in the code. It turned out that in case of minimum-across-the-day accuracy, the *LIMES*'s boost of performance is higher than in case of average-across-the-day accuracy. To figure the randomness nature of the experimental results, we applied the Wilcoxon statistical test to validate the performance of *LIMES* over the *incremental* baseline method which reaches the closest results to *LIMES*. Such an analysis is combined with the results visualization provided in the paper.

Note that the parameters of the models were left at default values. It may be possible that doing a careful hyperparameter search could improve the model performance.

6 Conclusions

We prepared a repository with implementation of baseline methods and *LIMES* dealing with the class-prior shift. To answer the need of reproducibility and reliability of results, the following measures were undertaken: we provided detailed information on how to recreate the dataset used in the paper, we made the source code publicly available and payed attention to setting the random seed and crated many different variants of the datasets. We believe that this repository can be valuable for other researchers working on the topic of modeling on streaming data under class-prior shift. Hopefully, it would fasten their work by using already implemented baselines.

References

1. Baker, M.: 1,500 scientists lift the lid on reproducibility. Nature pp. 452–454 (2016). https://doi.org/https://doi.org/10.1038/533452a
2. Liu, C., Gao, C., Xia, X., Lo, D., Grundy, J., Yang, X.: On the Reproducibility and Replicability of Deep Learning in Software Engineering. ACM Transactions on Software Engineering and Methodology **31**(1), 1–46 (jan 2022). https://doi.org/10.1145/3477535, https://doi.org/10.1145%2F3477535
3. Pineau, J., Vincent-Lamarre, P., Sinha, K., Lariviere, V., Beygelzimer, A., d'Alche Buc, F., Fox, E., Larochelle, H.: Improving Reproducibility in Machine Learning Research(A Report from the NeurIPS 2019 Reproducibility Program). Journal of Machine Learning Research **22**(164), 1–20 (2021), http://jmlr.org/papers/v22/20-303.html
4. Tatman, R., Vanderplas, J., Dane, S.: A practical taxonomy of reproducibility for machine learning research. In: Reproducibility in Machine Learning – Workshop at ICML (2018)
5. Tomaszewska, P., Lampert, C.H.: Lightweight Conditional Model Extrapolation for Streaming Data under Class-Prior Shift. 26th International Conference on Pattern Recognition (2022)