Experiment Replication report for Detecting Anomalies with Autoencoders on Data Streams by Lucas Cazzonelli et al.

Himanshu Choudhary

Eindhoven university of technology(Tu/e)
h.choudhary@student.tue.nl

Abstract. The report contains a detailed description over the replication of results from the paper Detecting Anomalies with Autoencoders on Data Streams [1]

Keywords: Data streams · Autoencoders · Anomalies · Data contamination

1 Introduction

Autoencoders (AEs) is one of the advanced neural network technique which sparked multiple areas of research in the field of machine learning. Although AEs and their variants are used significantly in offline settings, the online settings are still not sufficiently explored. In this paper, the authors proposed an approach for online anomaly detection using AEs and compared the performance with the current state of art methods on three real-world datasets. In summary, the major contributions of the paper are -

1. A formalization of streaming AD.
2. A competitive approach for AE-based anomaly detection on data streams.
3. An alternative optimization technique to improve the performance of AEs anomaly detectors under the influence of anomalous training examples.
4. An in-depth evaluation of our approaches with established real-world datasets.
5. A Python package[1] extending River[2] that facilitates the reproducibility and reuse of our work.

The authors evaluated their approach using various experiments over three different data streams i.e. Covertype, Shuttle, and Creditcard. They performed online anomaly detection over these data streams using AE, Denoising AE (DAE), and probability Weighted AE (PW-AE). The results produced by these techniques were compared against the previous state-of-the-art methods. The experiments also included testing over different contamination proportions (anomaly proportions) and parameter tunning to check the model's robustness.

[1] Available at https://github.com/online-ml/river

Results showed that Autoencoders and their variants performed significantly better than the previous approaches for online anomaly detection. Moreover, AEs took only a fraction of the processing time with respect to the other models.

This report aims to reproduce the results mentioned in the original paper along with all the experimentation. It contains a detailed explanation of how results are been generated, and what modifications one needs to make in order to replicate the results properly. Although the replication code is available for the above experiments[2], it lacks of proper documentation and contains some small bugs. Moreover, the authors have not described which systems and settings (module versions) were used to perform the tests and how much time it will take to run the scripts.

I tried to replicate the experiments by creating a new python environment on my system (Apple M1) and running the code as mentioned in the author's readme. I was unable to run the code because of the module's dependency errors. I contacted the authors of the paper and based on their suggestions changed the module versions and reduced the data stream size along with other experimentation parameters. Other than that, I fixed some small bugs in the code in order to run the scripts on my system.

Running the above scripts generated benchmarking scores for AEs and previous state-of-the-are models. It generated five raw and five processed csv(comma-separated values) files for benchmark, Capacity, Contamination, HST Baseline, and Learning Rate. The results showed higher overall accuracy for the Autoencoders and their variants with respect to the other models. Higher differences in processing time can also be observed between AEs and previous state-of-the-art models. Moreover, Generated plots showed the anomaly score distribution between anomalous and non-anomalous data points. Plots for learning rate, latent ratio, and data contamination showed the robustness of AE and its variants.

Since I ran the scripts on a smaller data set, the exact score can not be compared with the original results. I ran the above scripts with 50000 samples instead of 500000, due to the lack of computational resources. Although the exact numbers do not match I observe a similar trend, where AEs outperform the other models both in terms of score and runtime. The anomaly distribution plot matched with the original plot, as it was generated using the first 12000 examples. Other reproduced plots were also found to be almost similar to the one mentioned in the paper. Overall the results showed similar trends as mentioned in the paper. The original results can be easily reproduced with the default parameters in a high computational environment.

The basic layout of the report is as follows. Section 2 describes all the key terms that are necessary to understand the experiment, the original research problem, and the method used by the authors. Section 3 provides a detailed description of my replication approach and a comparison with the original methodology. In section 4 I provided the reproduced results, key notable things about the experimentation, and a comparison with the original results. Finally, the 5 section concludes the overall findings and the threats to validity.

[2] https://github.com/lucasczz/DAADS

2 Background

The following terms and their associated short descriptions summarize all the important factors that are necessary to understand the experimental setup.

The following list concludes the models that are used for anomaly detection.

- Autoencoders (AE): A feed-forward neural network architecture that learns in an unsupervised manner.
- Denoising Autoencoders (DAE): An extension of AE to avoid perfect reconstruction/overfitting.
- Probability Weighted Autoencoders (PW-AE): A novel extension with improved optimization objective for AE, proposed by the authors to address the issue of contamination in an online anomaly detection setting.
- Previous models for comparison -
  o iLOF: incremental local outlier factor [3]
  o HST: Half space trees [4]
  o RRCF: Robust Random Cut Forest [5]
  o xStream [6]
  o kit-net [7]

The below list contains information regarding key terms used for method evaluations and experimental settings.

- Contamination: contamination implies impurities in data. Here it denotes the different proportions of anomalies. It is used to check model robustness over the number of anomalies.
- Performance: To measure performance below metrics are used -
  o ROC-AUC: Receiver Operating Characteristic curve, It represents the degree of measure of separability.
  o PR-AUC: Precision-Recall curve, shows the trade-off between precision and recall for different thresholds.
  o Runtime: model execution time
- Parameters: Parameter tuning is also performed for AEs to ensure the robustness of the models. Below parameters are tuned in the neural network architecture -
  o latent space: proportion of hidden units in bottleneck layer of AEs with respect to the input layer
  o optimizer: functions that are used to update the model weights.
  o learning rate: The pace at which we perform adjustments to model weights.
- Mean Absolute Error (MAE): MAE is used to measure the reconstruction error(anomaly score) for autoencoders.
- seeds: seed values are used to fix the random initialization of model weights and other random values if needed for reproducibility.

In this paper, the authors researched, if autoencoders can be used for anomaly detection in online settings. Autoencoders are a state-of-the-art approach for detecting anomalies in offline settings, where AEs are trained on a large amount of non-anomalous data points, reducing reconstruction errors for clean data. Since there are no anomalies in training data the reconstruction loss remains higher for anomalies and can easily distinguish normal data from the encountered anomalies in test datasets. However, in online settings, there are multiple issues that are important to consider. For example, data may not be available beforehand and will arrive in chunks for an infinite amount of time, storing all the data and training the model will not be feasible in that case. Moreover, the stream may encounter distributional changes (concept drifts) which need to be tackled, as the definition of anomaly can change over time. Moreover, anomalies can concentrate in a small window degrading the model's performance. Keeping the above observations in mind authors find out based on different properties of AEs[1], they are still well-suited for online settings and are worth performing experimentation with.

In order to test the approach, the authors chose three real-world datasets, that is - Covertype, Shuttle, and Creditcard fraud detection [3]. Due to the properties of these data sets, such as different percentages of anomalies, presence of concept drifts, and sizes, the authors found these to be well-suited to check the model's robustness. AEs and their variants were used with SGD optimization function and other default settings. The Mean Absolute Error (MAE) metric is used to measure reconstruction error. The other well-established online anomaly detection methods were used with their default settings on similar datasets for comparison. Models were evaluated in terms of performance (ROC-AUC score, PR-AUC score and runtime), contamination robustness, and the effects of parameter tuning.

3 Experiment Setup

I reproduced the experimentation in order to verify the results as stated in the paper. Moreover, the reproduction of experimentation was performed in order to understand the source code and to analyze, how convenient it is to modify, so that I can extend the suggested approach for my thesis.

To run the experiment, I used my system, that is, Mac M1 (arm64 architecture) with 16 GB of ram. After getting some dependency errors during replication, I contacted the authors of the paper and based on their suggestion I used python 3.9 on a newly created anaconda[4] environment. After setting up the system I performed the following steps -

1. Cloned the GitHub repository[5] as provided in the paper.
2. Installed dependencies using requirements file from the project repository.
3. Modified the python scripts and reduced the sample and experimentation size in order to execute them in an appropriate amount of time. A detailed description of modifications is given in the section 4.

[3] all datasets are taken from - https://archive.ics.uci.edu/ml/datasets.php

[4] Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

[5] https://github.com/lucasczz/DAADS

4. Executed all python scripts one by one as suggested in the readme file. All experiments can be run together as well, but it could take a significant amount of time, so running separately is rather more appropriate.
5. Plotted the figures using plotting.ipynb notebook from the notebook folder.

I followed and reproduced all the experiments as stated in the paper in a similar way. Performed all the experiments including raw models, contamination, and parameter tuning. The only difference is, that I executed raw experiments on 50000 samples instead of 500000, and rather than taking the average score by running each model 10 times I reduced the number of iterations to 2.

In order to evaluate the replication I manually checked the generated plots and their trends with the original plots. Moreover, I compared the generated PR-AUC and ROC-AUC results with the original benchmarking scores by re-arranging replicated results in a similar format. Since the replicated scores can not be exactly similar to the original scores due to the change in the number of samples along with some parameters (stated in section 4), I analyzed, if AEs follow similar behavior with respect to other models as mentioned in the paper. Also, the tests are performed on a different system so the processing time will not be exactly similar, but the comparative study will remain the same.

4 Experiment Execution and Results

There are multiple changes that I performed in order to execute the experiment in my system. Before discussing changes in the scripts there are some crucial points to be considered as indicated below.

- The experiment is conducted on a Mac M1 machine (arm64 architecture) with 16GB of ram.
- Python version 3.9 is used.
- Code is also tested on Ubuntu 16.04 and 22.04 system with python version 3.9.
- Running the scripts could take a significant amount of time based on your machine.
- if you are using a mac system specifically, rather than using a multiprocessing module, the multiprocess python module should be imported

The above system settings are important to consider for eliminating the basic python module errors. With the above system settings, I made the following changes in the code to run it in a feasible amount of time. The following settings can be modified to perform experiments based on your requirements. The code with all the changes is marked in my repo with **"#CHANGE"** comment and can be run directly from my Github repository[6]. Table 1 indicates the time each file took for execution.

- changed the seed size from 10 to 2

[6] https://github.com/himanshudce/DAADS

	File name	Execution time (in minutes)
1	benchmark_exp.py	48
2	capacity_exp.py	98
3	contam_exp.py	156
4	lr_exp.py	17
5	scores_exp.py	2
6	hst_exp.py	3

Table 1: Execution time of the scripts

- reduced sample size from 5000000 to 50000
- changed model list in the scores files, as configurations were only set for those models.
- changed visualization parameters in plotting notebooks
- It is not mentioned in the readme file by the author, but we also have to execute the hst_exp python file in order to get the baseline model results for generating comparative plots. So, do execute **hst_exp.py** file.

To achieve exactly similar results as paper, the parameters can be kept as it is. The code can be executed on the above-suggested environment which will just increase the execution time exponentially for the benchmark experiments.

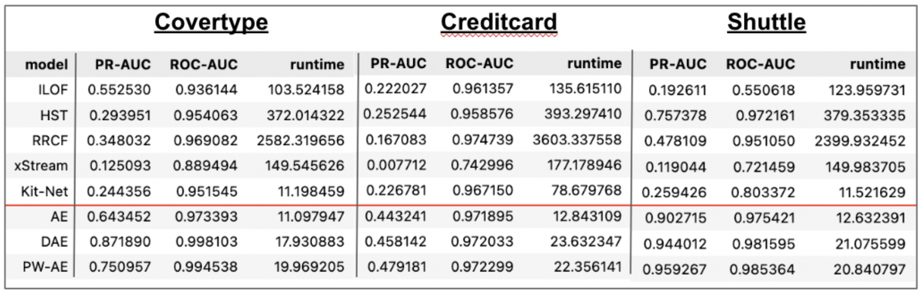
In the figure 1, replicated results, we can observe that overall runtime for AEs are quite low with respect to the other approaches. On average AEs outperforms the other models for all three datasets, i.e. Covertype, Creditcard and Shuttle. We can also observe, that PR-AUC score is quite low for covertype and creditcard dataset, for all the approaches, including AE and its variants. For Creditcard and Shuttle dataset, Probability weightage autoencoders(PW-AE) performed better than all the other models.

Fig 2, second image, plots the anomaly scores for non-anomalous and anomalous data points. Red dots indicates anomalies and grey dots are used to represent non anomalous data. These are plotted using the initial 12000 data points of shuttle dataset. The differentiation in anomaly score is on the higher end for PW-AE, resulting a better performance.

In fig 4 we can see the variation in anomaly score between anomalous and non-anomalous data points. Red dot represent the anomaly and grey dot represents the non-anomalous data. From this figure we can observe that there is a significant gap in reconstruction error (anomaly score) between anomaly and normal data for AE and its variants. Whereas this gap is not clearly visible in Kit-Net and RRCF models. Although, in HST we can still differentiate anomalies from the normal data. Moreover, DAE and PW-AE are able to differentiate anomalies from the starting with respect to AE.

When share of anomaly percentage increases from 2 to 10%, the accuracy, on average goes down for for DAE, PW-AE and the baseline model HST. The accuracy remains above than 95% for DAE and PW-AE, even with the higher

	Covertype			Creditcard			Shuttle		
Model	ROC-AUC	PR-AUC	Runtime	ROC-AUC	PR-AUC	Runtime	ROC-AUC	PR-AUC	Runtime
iLOF [20]	0.934	0.339	7.44	0.922	0.127	8.96	0.551	0.193	1.52
HST [26]	0.905	0.141	21.07	0.931	0.171	21.47	0.975	0.78	3.92
RRCF [2]	0.968	0.301	240.82	**0.950**	0.103	354.46	0.947	0.461	43.54
xStream [15]	0.754	0.033	13.85	0.711	0.005	15.38	0.724	0.118	2.35
Kit-Net [16]	0.905	0.205	**0.95**	0.943	0.141	6.02	0.803	0.259	**0.17**
AE	0.956	0.266	1.56	0.940	0.234	**1.67**	0.973	0.886	0.26
DAE	**0.984**	**0.501**	2.60	0.943	0.247	2.66	0.981	0.922	0.42
PW-AE	0.982	0.451	2.87	0.945	**0.258**	2.95	**0.986**	**0.955**	0.47



	Covertype			Creditcard			Shuttle		
model	PR-AUC	ROC-AUC	runtime	PR-AUC	ROC-AUC	runtime	PR-AUC	ROC-AUC	runtime
ILOF	0.552530	0.936144	103.524158	0.222027	0.961357	135.615110	0.192611	0.550618	123.959731
HST	0.293951	0.954063	372.014322	0.252544	0.958576	393.297410	0.757378	0.972161	379.353335
RRCF	0.348032	0.969082	2582.319656	0.167083	0.974739	3603.337558	0.478109	0.951050	2399.932452
xStream	0.125093	0.889494	149.545626	0.007712	0.742996	177.178946	0.119044	0.721459	149.983705
Kit-Net	0.244356	0.951545	11.198459	0.226781	0.967150	78.679768	0.259426	0.803372	11.521629
AE	0.643452	0.973393	11.097947	0.443241	0.971895	12.843109	0.902715	0.975421	12.632391
DAE	0.871890	0.998103	17.930883	0.458142	0.972033	23.632347	0.944012	0.981595	21.075599
PW-AE	0.750957	0.994538	19.969205	0.479181	0.972299	22.356141	0.959267	0.985364	20.840797

Fig. 1: first table **(original results)** shows the average benchmarking results for 10 random seeds. Second table **(replicated results)** shows the average benchmarking results for 2 random seeds.

proportion of anomalies. Accuracy of the baseline model (HST) fluctuate significantly, implicating non-uniform behaviour with respect to anomaly proportion. The above phenomena can be observed from fig 3.

In fig 5b we can see the performance of different optimization functions along with varying learning rates. SGD and HD-SGD produced better results with respect to the Adam optimizer. Best performance is achieved between 10^{-2} to 10^{-1} for SGD and HD-SGD. As we move towards bigger learning rate values, performance of SGD and HD-SGD increases whereas performance of Adam optimizer decreases. In fig 6b we can see an overall average increase in ROC-AUC accuracy for AE and its variants. Accuracy almost remains constant or improves for DAE and PW-AE if we increase latent layer ratio. Overall, all AE models performs better then the base model (HST).

Since reproduced results are generated for 50000 examples instead of 5000000, the exact scores can not be compared and won't be similar. I checked similarities in overall trends, and comparative study to get the intuition. Anomaly distribution graphs were plotted for initial 12000 samples of the data streams, which are similar to the original results and can be compared point to point. Similar to score tables, the contamination and parameter tunning experiments are not exactly similar, because of the randomly choose data points without defining seeds.

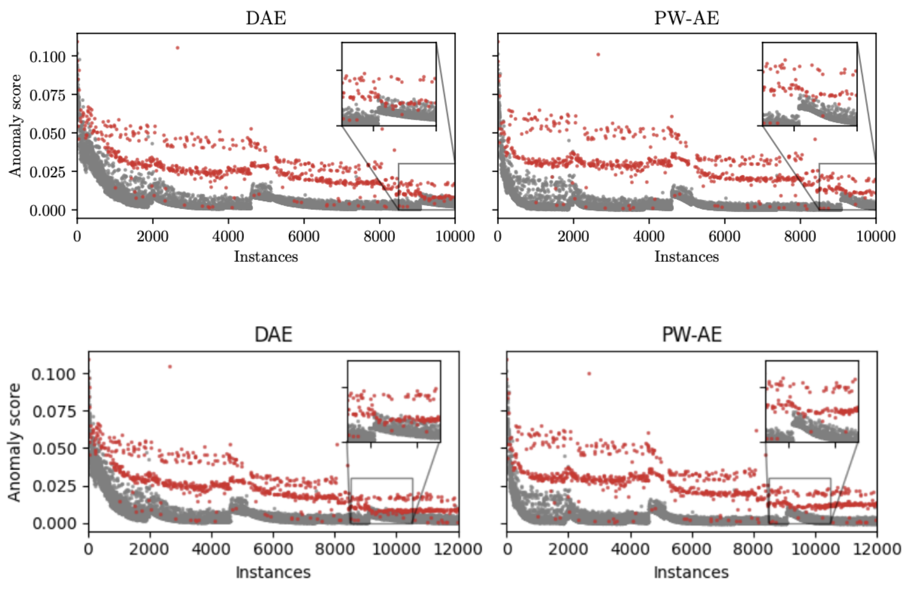


Fig. 2: first image **(original results)** and second image **(replicated results)** shows distribution of unscaled AE anomaly scores concerning the number of processed instances on Shuttle. Anomalies are drawn in red.

If I compare the generated score with the original scores from fig 1, we can see similar patterns in performances. AEs outperform on each of the datasets for both PR-AUC and ROC-AUC scores. Similar to the original results DAE performed best for the Covertype, RRCF performed best for Creditcard dataset and PW-AE performed the best for shuttle dataset. We can observe almost similar trends for all the results. Moreover, PR-AUC score remains on the lower end for both creditcard and covertype dataset. Distribution of anomaly scores in fig 2 and fig 4 is similar as the original paper. The only difference is that, replicated results are plotted for initial 12000 data points whereas original results were plotted for 10000 data points. Otherwise, we can see almost similar distribution as the original for anomalous and non-anomalous data points for different models.

Fig 3 also indicates almost similar trend in the generated graph as compared to the original one. DAE and PW-AE scores are on the higher end as compared to HST for both the plots. Although, the trend is similar but there is significant gap between scores in original and replicated results for covertype dataset. The ROC-AUC accuracies for the covertype are on the higher end as compared to the original one.

We can observe similar behaviour in learning rate plots except the one with the adam optimizer. These can be seen from fig 5a and fig 5b. Similar to this, experiments with latent ratio are also not differentiable in original and replicated

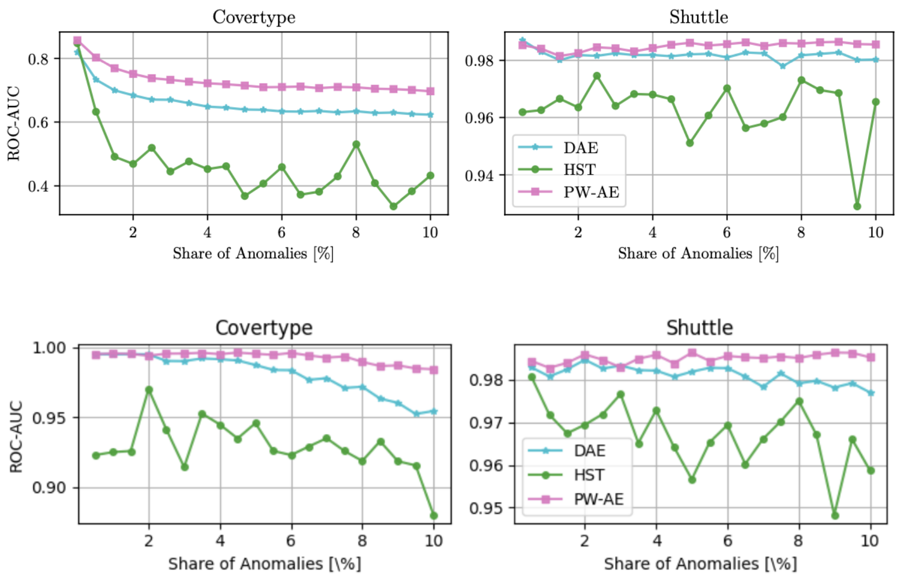


Fig. 3: first image **(original results)** and Second image **(replicated results)** shows ROC-AUC scores for varying anomaly percentages, which we generated by randomly sampling and inserting anomalies into the data. To allow for larger anomaly percentages, we used sampling with replacement. Each dataset created in this manner consisted of 50,000 examples.

results except some small fluctuations. These can be observed from the fig 6a and fig 6b.

5 Discussion

Overall the reproduced results were found to be almost similar to the results mentioned in the paper. Since the replication was done on a subset of the original data, the numbers were not supposed to be similar, but as expected the trends and comparative performance of the models with respect to each other came out similar to those on the larger dataset. Moreover, running over a small sample (initial 50000 examples) helped in identifying the important characteristic of autoencoders, that is, AEs can achieve better accuracy with a relatively lower number of data points as well. Also, in general, the variation of basic autoencoder (DAE and PW-AE) performs relatively better than AE on the cost of higher execution time. One important thing to consider is that the results are majorly generated for Covertype and Shuttle datasets only, whereas, Creditcard data, which is a better representation of the real-life scenario, and is not filtered/modified by the authors is not used to analyze extensively. Handpicking the analysis can originate doubts over the methodology. Also, the contamination and anomaly percentage plots were generated by sampling over random 50000 examples without setting up seeds. Because of which we can not get the similar plots, and this makes the plots non-reproducible (every code run will generate new plots depending on samples). It can be observed as well, that there is no benefit of using a complex optimizer (Adam, HD-SGD) over simple SGD.

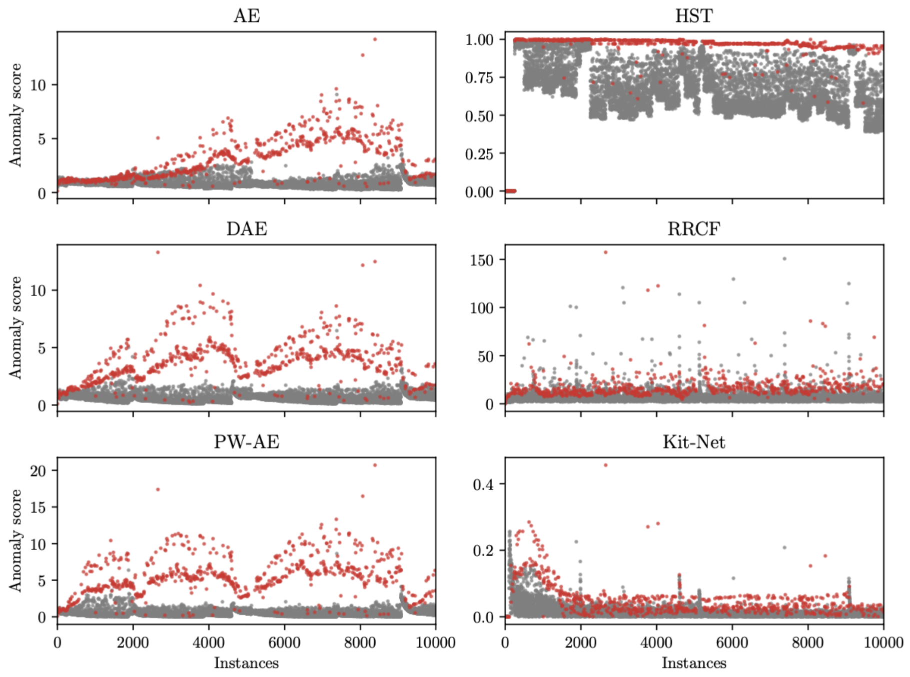
Since I replicated the results using first 50000 examples, results produced by randomly chosen data points instead of the initial once won't provide the similar results. Moreover, I set up seeds for random function in my modified code for reproducibility. The results may not be similar if the same seed value were not used. These small threats are important to consider in order to reproduce the similar results as mine.

Reproducing results is one of the crucial task for the validity of the approach. It can become quite complex if limited information is provided by the authors. While working on the above replication, i learned some important lessons to take care of. Firstly, it is very important to provide the exact environment settings where the code is tested. From my experience, in data science projects, 90% of the errors are due to the environment mismatch. providing the exact details of the system and libraries/module used reduces the significant amount of time for replicating the code. Secondly, before providing the final code, it should be tested in a separate environment, which is built from scratch. This end to end testing will make the code bug free from user's perspective. It will provide the similar experience to what the replicating users will do and will definitely helps to get rid of basic errors. Third, the documentation/readme should be clear, tested and easy to understand. While replicating my code, I found that although readme file provided by the author was clear, but it had one missing link. The authors forgot to mention about executing one of the necessary script required for plotting. Although, it is a very basic error, but it can be complicated for someone, who haven't gone through the paper and just trying out to visualize the results. Fourth, Whenever we have used random functions in the code, it should always be with the seed values, in order to replicate the similar results.

References

1. Lucas, C., Cedric, K.: Detecting Anomalies with Autoencoders on Data Streams. In: Editor, ecmlpkdd 2022, 2022/09, sub_245LNCS
2. Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H. M., Read, J., Abdessalem, T., Bifet, A. (2020). River: Machine learning for streaming data in Python. ArXiv. https://doi.org/10.48550/arXiv.2012.04740
3. D. Pokrajac, A. Lazarevic and L. J. Latecki, "Incremental Local Outlier Detection for Data Streams," 2007 IEEE Symposium on Computational Intelligence and Data Mining, Honolulu, HI, USA, 2007, pp. 504-515, doi: 10.1109/CIDM.2007.368917.
4. Tan, Swee Ting, Kai Liu, Fei Tony. (2011). Fast Anomaly Detection for Streaming Data.. 1511-1516. 10.5591/978-1-57735-516-8/IJCAI11-254.
5. Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust random cut forest based anomaly detection on streams. In Proceedings of the 33rd

International Conference on International Conference on Machine Learning - Volume 48 (ICML'16). JMLR.org, 2712–2721.
6. Emaad Manzoor, Hemank Lamba, and Leman Akoglu. 2018. XStream: Outlier Detection in Feature-Evolving Data Streams. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining (KDD '18). Association for Computing Machinery, New York, NY, USA, 1963–1972. https://doi.org/10.1145/3219819.3220107
7. Chen, Jinghui Sathe, Saket Aggarwal, Charu Turaga, Deepak. (2017). Outlier Detection with Autoencoder Ensembles. 10.1137/1.9781611974973.11.



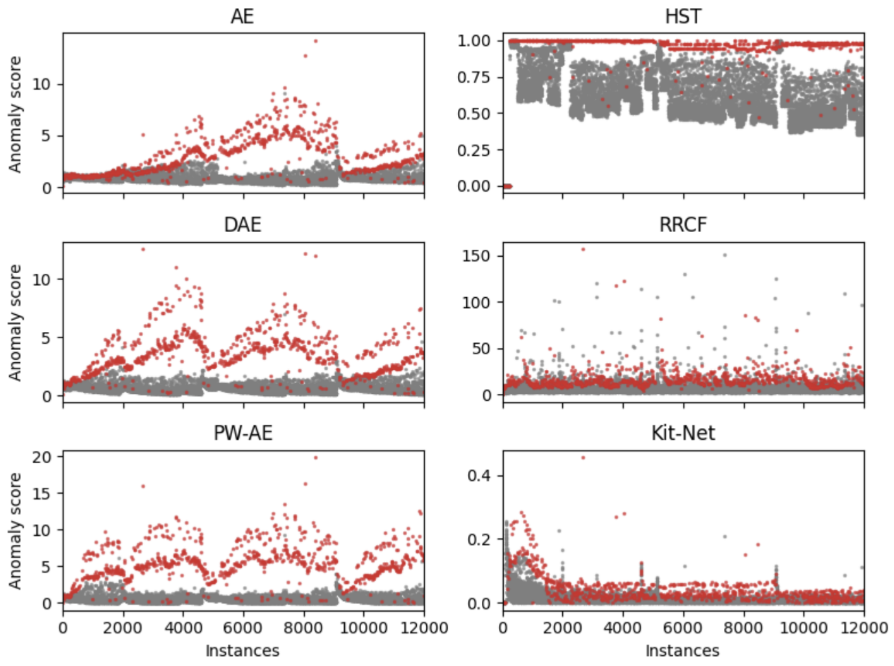
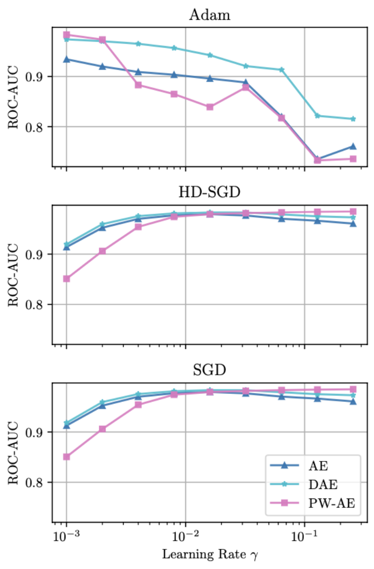
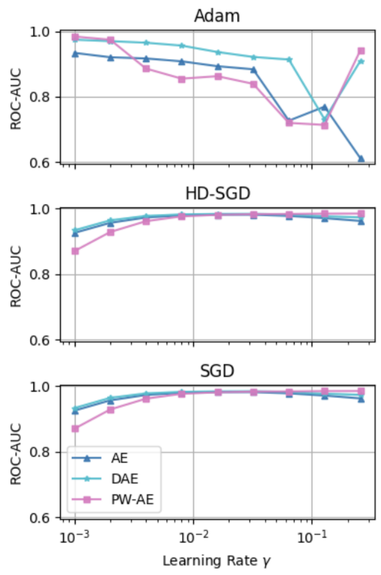


Fig. 4: first image **(original results)** and second image **(replicated results)** shows distribution of anomaly scores as a instances on Shuttle. Anomalies are drawn function of the number of processed in red.

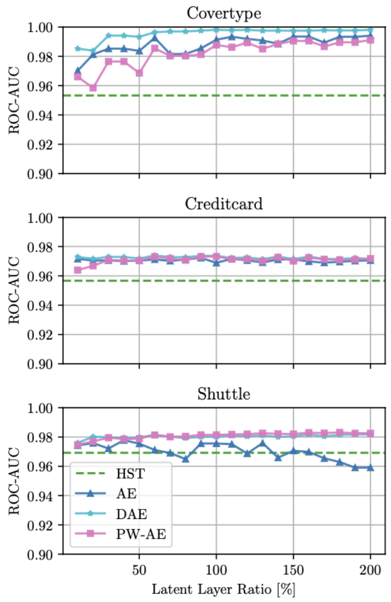


(a) **original results**

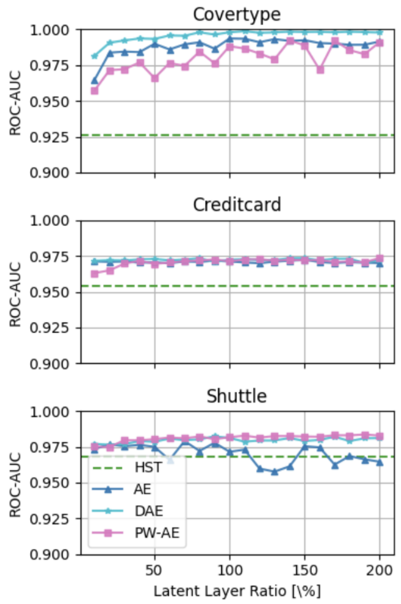


(b) **replicated results**

Fig. 5: ROC-AUC scores for variable learning rates averaged over the first 50,000 samples of each dataset



(a) **original results**



(b) **replicated results**

Fig. 6: ROC-AUC score for AE anomaly detectors with varying latent layer ratios on the first 50,000 samples of each dataset.