|  |  |  |
| --- | --- | --- |
| **Spring 2024** | **Report #5 – 05/05/2024** | **Dingyi Nie** |

**——————————————————————————————————————**

**I. Task Achieved Last Week**

**——————————————————————————————————————**

* Besides P19, experiments are also done on P12 and PAM datasets
* Ablation study (there’s a problem!)

**——————————————————————————————————————**

**II. Feedback and Interaction**

**——————————————————————————————————————**

* **Prof. Kuo’s Feedback**
  + Placeholder

**——————————————————————————————————————**

**III. Report**

**——————————————————————————————————————**

Since we started targeting at a publication, before we move on to more datasets, I thought it’d make sense to validate every module we applied contributes to the performance gain, instead of degrading it.

However, as I tried using **an XGBoost classifier directly on original input** (XGBoost imputed, padded 7-sensor, 336-length time series), it turns out it outperforms everything with a 91.36% AUROC score. Adding upstream processing modules such as SSL and DFT only degrades the performance. In other words, the XGBoost classifier at our pipeline’s last layer is where the main performance gain comes from.

I have to admit that it was my fault overlooking this case in the first place. However, previous DL works on irregularly sampled time series, such as [Raindrop](https://arxiv.org/pdf/2110.05357.pdf) (ICLR22) and [ViTST](https://proceedings.neurips.cc/paper_files/paper/2023/file/9a17c1eb808cf012065e9db47b7ca80d-Paper-Conference.pdf) (NeurIPS23), generally did not address this either. They neither mentioned boosting method in related works, nor included it as one of the baseline competitors. I have a feeling that it is almost a consensus that boosting methods perform better than NN on **tabular data**, given the fact that they are leading Kaggle competitions in this domain.

I have also implemented codes so far for P12 and PAM datasets as well, and similar performance is observed – XGBoost classifier itself is dominating. So I had a few attempts on combining GL tools with XGBoost classifier to obtain further performance gain, but the only trial that seemed working was: instead of input original data directly, we create an ensemble of original data + 1 layer of Saab-transformed original data, with another DFT module to select out the most useful AC components (3 out of 20, and drop all others). This merely increases performance by 0.01%.

|  |  |
| --- | --- |
| Method | AUROC |
| Lin. imp. + SSL (config.1) + LR | 81.9% |
| XGB imp. (strat.1) + SSL (config.1) + LR | 84.1% |
| XGB imp. (strat.2) + SSL (config.1) + LR | 84.0% |
| XGB imp. (strat.1) + SSL (config.2) + LR | 85.3% |
| XGB imp. (strat.1) + SSL (config.2) + XGB Classifier | 87.2% |
| XGB imp. (strat.1) + SSL (config.3) + XGB Classifier | **88.6%** |
| XGB Classifier (on original data) | **91.36%** |
| XGB Classifier (on simply ensembled data) | **91.37%** |
| XGB imp. (strat.1) + SSL (config.3) + LR | **86.96%** |
| Transformer | 83.2% |
| Transformer-mean | 84.1% |
| GRU-D | 83.9% |
| SeFT | 78.7% |
| mTAND | 80.4% |
| Raindrop | 87.0% |
| ViTST | **89.2%** |

**——————————————————————————————————————**

**IV. Next Steps**

——————————————————————————————————————

* DL vs GL vs Boosting – how do we justify ourselves?

**——————————————————————————————————————**

**V. Milestone**

——————————————————————————————————————

* Refer to I and III.
* Codes are available here: <https://github.com/d9sus4/GL-TS>