PyTorch-Lifestream: Learning Embeddings on Discrete Event Sequences

Artem Sakhno[1], **Ivan Kireev**[1], **Dmitrii Babaev**[2], **Maxim Savchenko**[1], **Gleb Gusev**[1], **Andrey Savchenko**[1,3]

[1]Sber AI Lab, Moscow, Russia,
[2]SaluteDevices, Moscow, Russia
[3]ISP RAS Research Center for Trusted Artificial Intelligence, Moscow, Russia

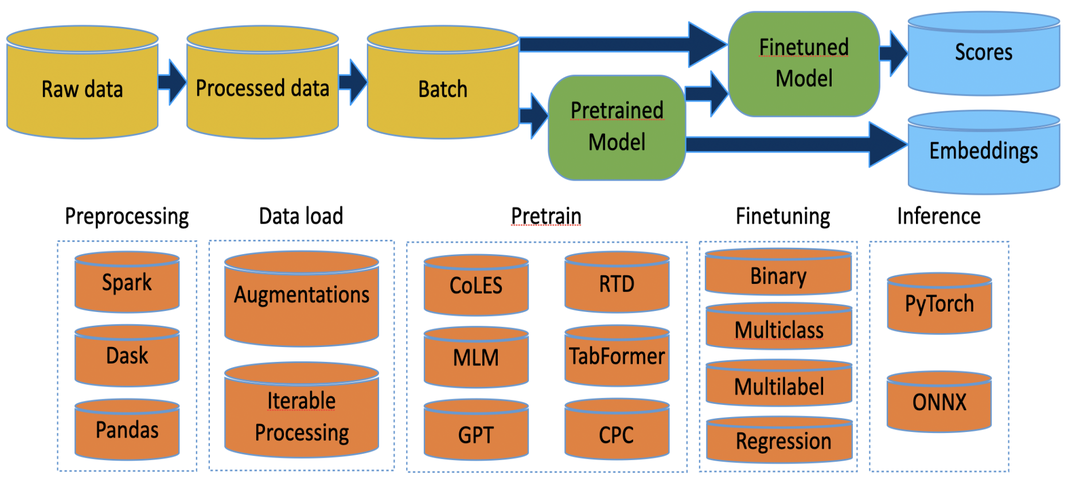
GitHub pytorch-lifestream

GitHub sb-ai-lab

Introduction

Event sequences appear across domains — finance, healthcare, e-commerce, telecommunications — each event defined by a timestamp and attributes like transactions, diagnoses, or interactions. Existing tools such as recommender systems and temporal point processes mainly predict the next event, while real-world tasks demand modeling full sequences: fraud detection, customer segmentation, behavior scoring. Unlike regular time series or text, event sequences are irregular and multimodal, mixing categorical and numerical features. To address this challenge, we present **PyTorch-Lifestream (PTLS)** — an open-source library for learning embeddings of event sequences, supporting data from transactions, clickstreams, and geolocation, and enabling downstream tasks from anomaly detection to forecasting.

PyTorch-Lifestream Overview

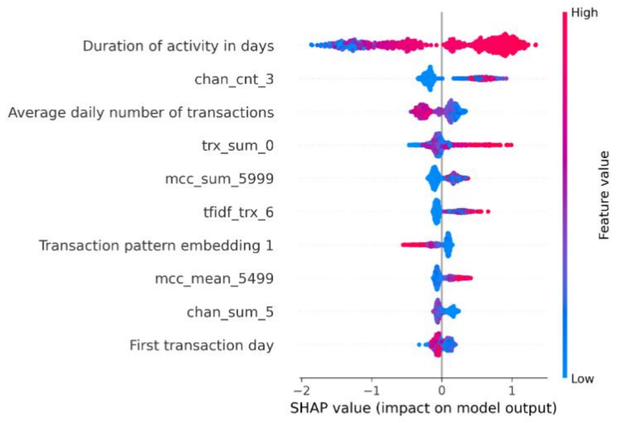


PyTorch-Lifestream provides a full pipeline for model development, enabling the extraction of embeddings or scores for downstream tasks directly from raw event data. Built on top of PyTorch Lightning, it is modular and extensible, which simplifies distributed training, ensures reproducibility, and supports integration of custom components such as checkpoints, evaluation metrics, or training logic.

Event Sequence Data

- Temporally ordered data such as **transactions**, **app clicks**, or **clinical records**.
- Contain **numerical** and **categorical** features.
- **Irregular in time** — events occur with uneven intervals.
- **High-dimensional and heterogeneous**, combining diverse behavioral attributes.
- Often **sparse and long**, with thousands of events per entity.
- Used for **default prediction**, **churn detection**, **risk modeling**, and **what-if analysis**.

	client_id	trans_date	small_group	amount_rur
0	33172	6	4	71.463
1	33172	6	35	45.017
2	33172	8	11	13.887
3	33172	9	11	15.983
4	33172	10	11	21.341
...	...	...	...	...
26450572	43300	727	25	7.602
26450573	43300	727	15	3.709
26450574	43300	727	1	6.448
26450575	43300	727	11	24.669
26450576	43300	729	3	19.408



Model Architecture

PTLS models include two core components: **TrxEncoder** and **SeqEncoder**.
TrxEncoder embeds categorical features, normalizes numerical ones, and combines them into unified event vectors, supporting custom transformations like piecewise linear encodings.
SeqEncoder models event embeddings with RNNs or Transformers (including Hugging Face and x-transformers), producing either token-level or sequence-level embeddings for labeling, classification, or regression.
PTLS also supports **multimodal event streams**.

Learning Strategies

PTLS offers a broad set of strategies for learning event sequence embeddings.

•**Contrastive approaches**: CoLES[1], CPC[2], Barlow Twins[3], VicReg[4] — capture similarities and temporal dependencies between subsequences.
•**Masked modeling**: MLM[5], TabFormer[6], RTD[7] — inspired by language models, these methods mask or replace parts of a sequence and train the network to reconstruct them.
•**Generative methods**: GPT-based models[8] — predicting attributes of the next event.
All objectives are implemented modularly, allowing users to integrate custom losses or external libraries such as PyTorch Metric Learning.

For downstream tasks, the **SequenceToTarget** module provides fine-tuning for classification, regression, anomaly detection, and scoring, reusing pretrained embeddings while adapting them to domain-specific goals. This two-stage design — pretraining + task-specific adaptation — allows practitioners to combine the power of large-scale self-supervised learning with practical business applications.

Benchmark

Along with the library, we introduce a benchmark for event sequence modeling based on three banking datasets: **Default Scoring** (1.4M users), **Age prediction** (50K users), and **Churn prediction** (11K users). All datasets represent user transactional activity but differ significantly in scale.

We evaluate the methods implemented in PTLS and compare them with manually engineered features. For models such as GPT, MLM, and TabFormer, pooling was applied to generate sequence-level embeddings. Results show that self-supervised methods consistently perform on par with or outperform feature-engineered baselines. Moreover, combining embeddings from different models often improves performance further.

	Scoring	**Age**	**Churn**
Method	AUROC	Accuracy	AUROC
Hand made features	0.7792	0.629	0.827
Barlow Twins	0.7878	0.643	0.839
VicReg	0.7886	0.598	0.829
CPC	0.7919	0.602	0.792
CoLES	0.7921	0.640	0.841
MLM	0.7791	0.621	0.817
RTD	0.7910	0.631	0.771
Tabformer	0.7862	0.601	0.827
GPT	0.7737	0.589	0.824

References

[1] Babaev D., Ovsov N., Kireev I., Ivanova M., Gusev G., Nazarov I., Tuzhilin A. **CoLES: Contrastive Learning for Event Sequences with Self-Supervision.** SIGMOD, 2022.
[2] Oord A., Li Y., Vinyals O. **Representation Learning with Contrastive Predictive Coding.** arXiv:1807.03748, 2018.
[3] Zbontar J., Jing L., Misra I., LeCun Y., Deny S. **Barlow Twins: Self-Supervised Learning via Redundancy Reduction.** ICML, 2021.
[4] Bardes A., Ponce J., LeCun Y. **VicReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning.** arXiv:2105.04906, 2021.
[5] Devlin J., Chang M.-W., Lee K., Toutanova K. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.** NAACL, 2019.
[6] Padhi I., Schiff Y., Melnyk I., et al. **Tabular Transformers for Modeling Multivariate Time Series.** ICASSP, 2021.
[7] Clark K., Luong M.-T., Le Q., Manning C. **ELECTRA: Pre-training Text Encoders as Discriminators Rather than Generators.** arXiv:2003.10555, 2020.
[8] Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. **Language Models are Unsupervised Multitask Learners**. OpenAI Blog, 2019.