Contrastive Learning for Event Sequences with Self-Supervision on multiple domains

Denis Grankin [1] **Ivan Apanasevich** [1] **Ekaterina Andreichuk** [1] **Irena Gureeva** [1] **Mikhail Konenkov** [1]

Abstract

This is the replication study of original work by (Babaev et al., 2022). This document contains results of the final project of the Machine Learning course at Skoltech in 2023. Pytorch-lifestream library was implemented for event sequences data extraction and CoLES method was used for embeddings. Data Fusion 2022 datasets were investigated and prepared for testing. Downstream tasks were implemented for data embeddings extracted with CoLES, Agg Baseline and Random Encoder. Results were obtained for different methods and datasets, their performance was compared.

Github repo: ClosedAI github
Project presentation: ClosedAI presentation

1. Introduction

Data embeddings are an important tool in machine learning, providing low-dimensional representations of high-dimensional data that can be used for a variety of tasks, including supervised and self-supervised learning, exploratory data analysis, and data visualization. Data embeddings can be used to express different objects with the same semantics (Zhuang et al., 2019), user search history (Zhu et al., 2021) and other real-world users data (Wang et al., 2022).

One way to construct embeddings is by extracting event sequences from real-world data. Event sequences are a form of sequential data that capture the behavioral actions of individuals over time, making them valuable for a variety of applications. Examples of sequential data include DNA sequences of genes, purchase histories of online customers, click-streams. The latter two are also an example of *lifestreams* — event sequences that are attributed to a person and capture theirs regular and routine behavioural actions of a certain type.

Extracting event sequences, mining over sequential data and constructing embeddings from it are well-known machine learning problems with various approaches for each topic. However, constructing embeddings from event sequences can be challenging, as not all types of sequential data are created equal. For example, log entries, IoT telemetry, industrial maintenance, and other industrial and financial event sequences typically consist of interleaved relatively independent sub-streams.

One of the self-supervised sequence embedding methods, Contrastive Learning for Event Sequences (CoLES), showed strong performance on four publicly available lifestreams datasets in extracting embeddings. Self-supervised methods like CoLES allow us to mine data without labeling data. CoLES proposes both novel contrastive learning representation and novel augmentation algorithm, which generates sub-sequences of observed event sequences and uses them as different high-dimensional views of the object (sequence) for contrastive learning. The idea of CoLES is that low-dimensional embeddings of such sub-sequences should be closer to each other.

In our research we compared CoLES with two other methods for learning embeddings: **random features encoder** and **aggregate baseline encoder**. We implemented downstream machine learning task on two real-world datasets: transactions dataset and clickstream dataset from VTB and got embeddings for each method for both datasets. We have shown that COLeS is an effective embedding technique, better than random features encoder, which one is COLeS technique based for, but still has lower perfomance than aggregate baseline encoder technique.

The main contribution of this report are as follows:

- Data Fusion 2022 datasets were investigated and prepared for testing using standard *Pandas* preprocessing and *PySpark* preprocessing;
- *pytorch-lifestream* library was implemented for event sequences data extraction and CoLES method was used for embeddings extraction. Also aggregation baseline

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Denis Grankin <Denis.Grankin@skoltech.ru>.

Final Projects of the Machine Learning 2023 Course, Skoltech, Moscow, Russian Federation, 2023.

method and random encoder method were used as baseline methods;

- Downstream tasks were implemented for data embeddings extracted with CoLES, aggregate baseline encoder and random encoder;
- Results were obtained for different data extraction methods and both datasets, methods' performance were evaluated;
- Results were compared, CoLES method was found to be the best algorithm for data extraction among three investigated methods.

2. Related work

Sequential data usually requires a careful design of its embedding before being fed to algorithms. Sequence embedding is one of the feature learning problems on sequential data (Cho et al., 2014), (Gehring et al., 2017), where the goal is to transform a sequence into a fixed-length embedding. We are interested in applying self-supervised learning for sequential data. The sequential recommendation aims at predicting the next items in user behaviors, which can be solved by characterizing item relationships in sequences. Due to the data sparsity and noise issues in sequences, a SSL paradigm is proposed to improve the performance, which employs contrastive learning between positive and negative views of sequences (Liu et al., 2021).

Also self-supervised learning have demonstrated effectiveness in different machine learning domains, such as Natural Language Processing (Kachuee et al., 2020) and computer vision (Cao & Wu, 2021). In the field of natural language processing, contrastive learning is usually used to improve the quality of embedding representation by comparing feature vectors, bringing semantically similar and same label embeddings closer, and distancing semantically dissimilar and different label embeddings.

3. Algorithms and Models

3.1. Datasets

For our research of methods for data embeddings extraction we have used datasets with industrial data of transactions and clickstreams from competition Data Fusion 2022. The goal of this competition is to predict whether a client has a higher education or not, according to the data.

The contents of the datasets are described below:

- transactions (Table 1) — 1.98 million rows, which includes bank customer ID, mcc-code of transaction, transaction currency, amount in transaction currency, date and time of the transaction. Here customer ID column and data and time of the transaction column are shortened in order to show the content of the table.

Table 1. Transactions dataset

	user_id	mcc_code	currency_rk	transaction_amt	transaction_dttm
0	0009...	5411	48	-361.07	20-08-03 08:05
1	0009...	5499	48	-137.31	20-08-05 01:27
2	0009...	5499	48	-138.85	20-08-05 03:28
3	0009...	4829	48	-309.48	20-08-06 00:36
4	0009...	5411	48	-133.47	20-08-09 00:30

- clickstream (Table 2) — 126.75 million rows, which includes ISP client ID, visited categories, date and time of page visit, device ID. Here ISP client ID column and data and time of page visit column are shortened in order to show the content of the table.

Table 2. Clickstream dataset

	user_id	cat_id	timestamp	new_uid
0	000143...	165	2021-01-30 20:08	1873448
1	000143...	165	2021-01-31 20:06	1873448
2	000143...	308	2021-01-31 20:12	1873448
3	000143...	931	2021-01-31 22:12	1873448
4	000143...	931	2021-02-01 16:57	1873448

- target (Table 3) — 8509 rows, which includes bank customer ID (from transactions) and customer ID (from clickstream).

Table 3. Target data

	bank	higher_education
0	3755b59782464456bac1aec1a44e0db3	0.0
1	604a550439d644718ea6e1693fbf03dc	0.0
2	542d4776ebe5454fb8ab36f1c276fe0e	1.0
3	ee37fecea44d475ca030cde7ff7d545d	0.0
4	18617a1100f44a99b3a0772341fec3db	0.0

3.2. Preprocessing

For preprocessing we used *Pandas* Python library with the *PandasDataPreprocessor* for transactions dataset and *PySpark* Python library for clickstream dataset. PySpark was chosen because the clickstream dataset contains too many rows, and standard preprocessing methods were overloading RAM. For transactions dataset it was possible to preprocess it with Pandas and further use this data for learning. The preprocessed transactions dataset can be seen in Table 4.

PySpark is a Python interface to Spark, a tool for data analytics, data engineering, and machine learning on local machines or clusters of machines. For this reason, unlike

Table 4. Transactions dataset, preprocessed with Pandas library

	user_id	event_time	mcc_code	currency_rk	transaction_amt
0	00093...	tensor(15964...	tensor(1,...	tensor(1,...	tensor(-361.07,...
1	0009e...	tensor(15962...	tensor(1,...	tensor(1,...	tensor(-44.52, ...
2	000b2...	tensor(15963...	tensor(3,...	tensor(1,...	tensor(-334.58, ...
3	000c5...	tensor(15964...	tensor(74,...	tensor(1,...	tensor(-1923.65, ...
4	000e0...	tensor(15962...	tensor(6,...	tensor(1,...	tensor(-728.38, ...

Pandas, which does not scale well to clusters without third-party libraries, PySpark supports data analysis at scale "out of the box".

Pyspark provides methods to read Parquet file into DataFrame and write DataFrame to Parquet files. Parquet is a columnar file format whereas CSV is row based. Columnar file formats are more efficient for most analytical queries. It is possible to speed up Panda DataFrame queries by converting CSV files and working off of Parquet files. Therefore, for clickstreams we have used *PySparkDataPreprocessor* method, saved the data as Parquet file and further used it as iterable dataset during learning our models. Preprocessed clickstream dataset can be seen in Table 5.

Table 5. Clickstream dataset, preprocessed with PySpark

	user_id	cat_id	event_time
0	018d95...	[29,...	[1611903458, 1612164353,...
1	01ef0e...	[4,...	[1610973443, 1612144860,...
2	0570e0...	[12,...	[1612968600, 1613295720,...
3	0677be...	[12,...	[1622604120, 1622936168,...
4	0914be...	[1,...	[1612744599, 1612756008,...

We must mention that preprocessed data is shortened in order to show the content of the columns.

3.3. Dataset split

For each dataset with different data (clickstreams and transactions) we set apart 20% persons from the labeled and unlabeled parts of the data as the test det, used for evaluation on the downstream task. The remaining 80% of labeled and unlabeled data was used as training set for learning . For the learning of semi-supervised and baseline algorithms we used all data from training sets, including labeled data. During training of the supervised models for downstream task the unlabeled data was ignored.

3.4. CoLES method overview

The CoLES method described in [REF] — the paper on which our replication study is based on. We focus on discrete sequences of events. Our goal is to learn an *encoder* M that maps event sequences into a feature space $\mathbf{R}^d$ in such a way that the obtained *embedding* $\{x_e\} \to c_e = M(\{x_e\}) \in \mathbf{R}^d$ encodes the essential properties of entities and disregards irrelevant noise contained in the sequence. That is, the embeddings $M(\{x'\})$ and $M(\{x''\})$ should be close to each other, if x' and x'' were paths generated by the same process, and further apart, if generated by distinct processes.

Algorithm 1 Random slices sub-sequence generation strategy

hyperparameters: m, M:minimal and maximal possible length of a sub-sequence; k:number of samples.
input: A sequence $S = \{z_j\}_{j=0}^{T-1}$.
output S: sub-sequences of S.
for $i = 1$ **to** k **do: do**
  Generate a random integer T_u uniformly from $[1, T]$;
  if $T_i \in [m, M]$ **then**
    Generate a random integer s from $0, T - T_i$;
    Add the slice $\hat{S}_i := \{z_{s+j}^{T_i-1}{}_{j=0}\}$ to $\mathbf{S}$;
  end if
end for

When there is no sampling access to the latent processes, one could employ synthetic augmentation strategies, which are akin to bootstrapping. Most augmentation techniques proposed for continuous domains, such as image displacement, color jitter or random gray scale in CV, are not applicable to discrete events. Thus, generating sub-sequences from the same event sequence could be used as a possible augmentation. The key property of event sequences that represent lifetime activity is periodicity and repeatability of its events. This motivates the Random slices sampling method applied in CoLES (presented in Algorithm 1, described in the original paper).

Contrastive loss. A classical variant of the contrastive loss, proposed in (Babaev et al., 2022), is considered in original paper, which minimizes the objective

$$\mathrm{L}_{uv}(M) = Y_{uv}\tfrac{1}{2}d_M(u,v)^2 + (1 - Y_{uv})\tfrac{1}{2}max\{0, \rho - d_M(u,v)^2\},$$

with respect to $M : \chi \to \mathbf{R}^n$, where $d_M(u,v) = d_M(c_u, c_v)$ is the distance between embeddings of the pair $(u,v), c_* = M(\{x_*(\tau)\})$, Y_{uv} is a binary variable identifying whether the pair (u,v) is positive, and ρ is the soft minimal margin between dissimilar objects.

Other method's features, like batch generation, encoder architecture and negative sampling described in the original paper. In Figure 1 you can see the general framework. However, our implementation consists only of Phase 1 and Phase 2.a steps. Also we've implemented aggregate baseline encoder and random feature encoder on both datasets in order to investigate more data embeddings' extraction methods and to compare the resulting scores of these three methods.

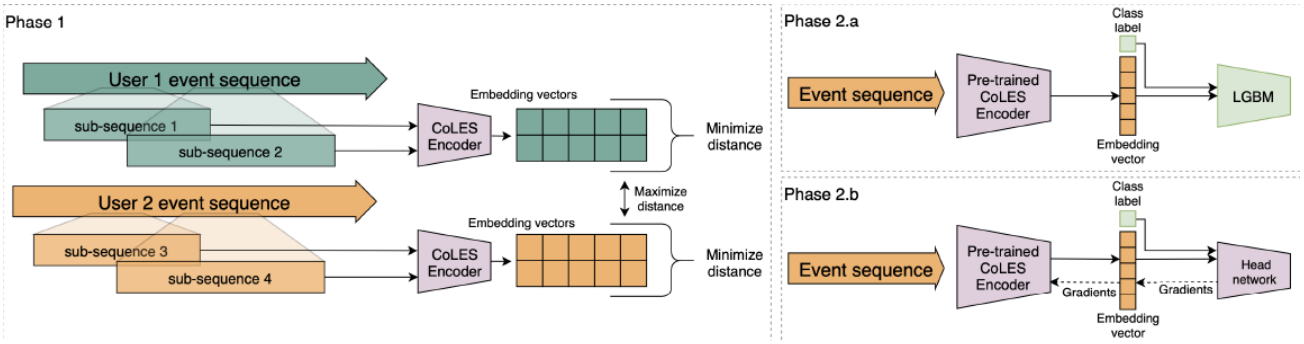

Figure 1. General framework. Phase 1: Self-supervised training. Phase 2.a Self-supervised embeddings as features for supervised model. Phase 2.b: Pre-trained encoder fine-tuning.

3.5. Other methods for data embeddings

In **aggregate baseline method** categorical features are decomposed into One-Hot Encoder. Numerical features are decomposed according to the resulting columns. A separate encoder is used to generate features. The encoder interface is the same as that of RNN CoLES. Aggregate baseline encoder doesn't take into account the sequential nature of data, unlike CoLES.

Random feature encoder has architecture which is similar to CoLES, but it is not pre-trained. Therefore, all weights are random.

4. Experiments and Results

4.1. General pipeline

4.1.1. Experiment stages description

We were following 4-stage pipeline which can be seen on Figure 2. In this Section of our report we provide the resulting performance of the models on downstream tasks and further analyze the results in the Section 5. More detailed description of each stage can be foung in Section 4.2.

4.1.2. Computational infrastructure

We were using Google Colab for conducting experiments on models' performances, best parameters search (including embedding size, sampling technique and encoder). For clickstream dataset preprocessing with PySpark we have been using personal laptop with the following parameters:

- GPU: NVIDIA GeForce 3070 RTX,
- RAM: 32 Gigabytes.

4.2. CoLES method

For first CoLES training we have chosen the parameters that are presented in the *pytorch-lifestream* library in the CoLES demo. The parameters can be seen in Table 6.

Table 6. Hyper-parameters for the first CoLES training

Model parameters	CoLES
Embedding size	256
Loss function	Contrastive loss
Learning rate	0.001
Resampling	Sample Slices
Size of subsequence	25-200
Encoder	GRU

Then we created the data embeddings using trained CoLES algorithm. For downstream task we implemented *Catboost*. Catboost parameters can be seen in Table 7.

Table 7. Catboost parameters

Model parameters	CoLES
Number of iterations	500
Loss function	Logloss
Learning rate	AUC
Logging level	Silent
Depth	5

We have chosen *accuracy, f1 score, precision score and ROC AUC score* as measures for evaluating model results. Metrics achieved on Catboost downstream task for both transactions and clickstreams datasets can be seen in table 8.

After that, we have conducted an experiment to find the best model performance depending on embedding size. To do that, we have run general pipeline for embeddings sizes of $[32, 64, 96, 128, 256, 512]$.

In the result of the experiment we found out, that the best performance model shows on embedding size of 512. The metrics for both datasets on this size can be seen in Table 9.

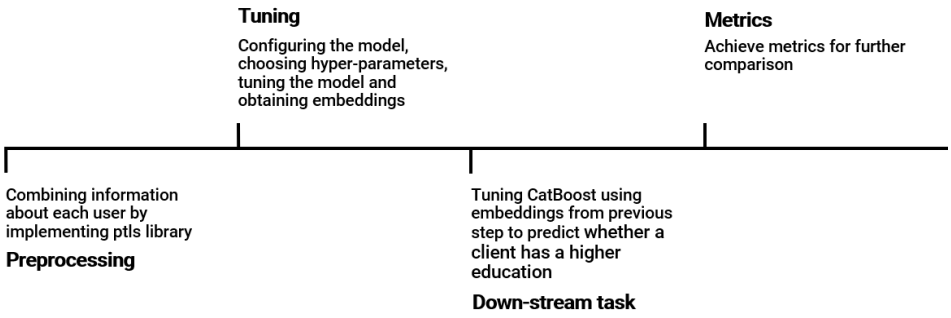

Figure 2. General pipeline for finding model quality.

Table 8. CoLES model quality on transactions and clickstream datasets with different metrics

Metrics	Transactions	Clickstream
F1 score	0.8615	0.8334
ROC-AUC score	0.7583	0.5936
Precision	0.9605	1.0000
Accuracy	0.7708	0.7145

Also we show the plot of ROC-AUC score depending on the embedding size in Figure 3. We have chosen the ROC-AUC score as the main metric based on the two facts:

1. Since we have unbalanced datasets, ROC-AUC score is more relevant among all chosen metrics;
2. ROC-AUC score was the metric in the original Data Fusion 2022 competition. Since we are using the datasets for this competition, usage of the score from this competition is justified.

Table 9. ROC-AUC score for embedding dimensionality of 512 (CoLES method)

Metrics	Transactions	Clickstream
F1 score	0.8541	0.8331
ROC-AUC score	0.7661	0.6137
Precision	0.9250	0.9946
Accuracy	0.7654	0.7152

Also we have conducted experiments on searching best parameters for our datasets. However, since our computing infrastructure were not capable of *GridSearch* on the whole parameters grid in the face of time constraints, we were experimenting with parameters one-by-one and 'by hand'. As a result, we determined GRU as the best encoder, *SampleRandom* as the best sampling technique for transactions dataset and *SplitRandom* as the best sampling technique for clickstream dataset. Other parameters were equal to such in Table 6.

The performance of the model with each tested encoder and each sampling technique is shown in Tables 10 and 11, respectively.

Table 10. ROC-AUC score for different CoLES encoders

Methods	Transactions	Clickstream
GRU	0.7606	0.5978
LSTM	0.7528	0.5766

Table 11. ROC-AUC score for different CoLES sampling techniques

Methods	Transactions	Clickstream
SampleSlices	0.7605	0.5978
SplitRandom	0.7493	0.6003
SampleRandom	0.7678	0.5902

Our results are consistent with the findings of the original paper: the larger the embedding size, the better score the CoLES model shows. It is required to find a reasonable bias-variance tradeoff so that the model shows the best performance.

4.3. Metrics for other methods

4.3.1. Aggregate baseline encoder

Following the general pipeline, we trained aggregate baseline encoder for data embeddings extraction. We chose base parameters from the pytorch-lifestream demo Jupyter notebooks. Then extracted data embeddings we used in downstream task using Catboost with the same parameters as in Table 7. The quality of the model on both datasets is presented in Table 12.

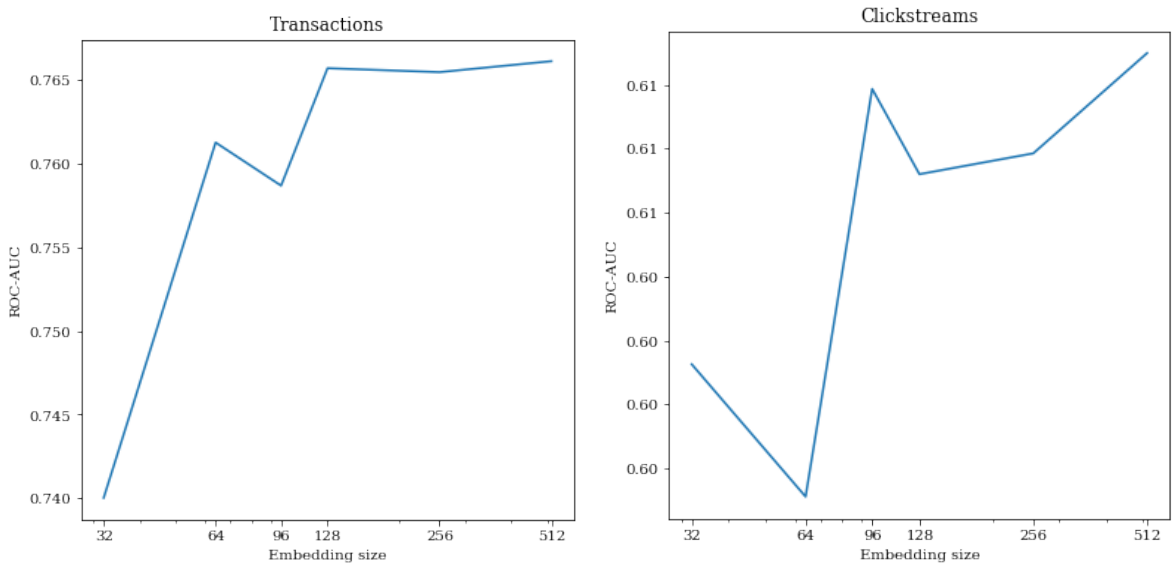

Figure 3. Embedding dimensionality vs. quality

Table 12. Aggregate baseline encoder quality on transactions and clickstream datasets with different metrics

Metrics	Transactions	Clickstream
F1 score	0.8650	0.8424
ROC-AUC score	0.7999	0.5564
Precision	0.9194	1.0000
Accuracy	0.7870	0.7278

4.3.2. Randon feature encoder

We also followed our experiment design to achiever random feature encoder performance on both datasets. Our results are listed in Table 13

Table 13. Random feature encoder performance on downstream task on both datasets

Metrics	Transactions	Clickstream
F1 score	0.8504	0.8424
ROC-AUC score	0.5970	0.5516
Precision	0.9927	1.0000
Accuracy	0.7408	0.7278

5. Conclusion

CoLES is a effective embedding technique. In comparison with Random feature encoder,embeddings achieved using CoLES showed better results for Clickstream and Transactions datasets, which can be seen in higher accuracy, ROC-AUC score and F1 score. But in comparison with aggregate feature encoder,which is a powerful baseline, CoLES achieved lower score. But yet, further investigation of CoLES hyperparameters is needed as score still may be improved with best hyperparameters. The dependency of the embedding size on the quality of supervised task was also demonstrated . It was found that with the growing embedding size , the quality on the downstream task also increases for both transactions and clickstreams datasets. Different types of encoders were evaluated on the downstream task. It was found that *GRU* encoder shows best quality for both of the datasets. The quality of CoLES on downstream task with different sampling techniques were also obtained. *SampleRandom* technique was best for Transactions dataset, while *SplitRandom* was better for Clickstreams dataset.

References

Babaev, D., Ovsov, N., Kireev, I., Ivanova, M., Gusev, G., Nazarov, I., and Tuzhilin, A. Coles: Contrastive learning for event sequences with self-supervision. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1190–1199, 2022.

Cao, Y.-H. and Wu, J. Rethinking self-supervised learning: Small is beautiful. *arXiv preprint arXiv:2103.13559*, 2021.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin,

Y. N. Convolutional sequence to sequence learning. In *International conference on machine learning*, pp. 1243–1252. PMLR, 2017.

Kachuee, M., Yuan, H., Kim, Y.-B., and Lee, S. Self-supervised contrastive learning for efficient user satisfaction prediction in conversational agents. *arXiv preprint arXiv:2010.11230*, 2020.

Liu, Z., Chen, Y., Li, J., Luo, M., Philip, S. Y., and Xiong, C. Self-supervised learning for sequential recommendation with model augmentation. 2021.

Wang, L., Lim, E.-P., Liu, Z., and Zhao, T. Explanation guided contrastive learning for sequential recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 2017–2027, 2022.

Zhu, Y., Nie, J.-Y., Dou, Z., Ma, Z., Zhang, X., Du, P., Zuo, X., and Jiang, H. Contrastive learning of user behavior sequence for context-aware document ranking. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2780–2791, 2021.

Zhuang, Z., Kong, X., Elke, R., Zouaoui, J., and Arora, A. Attributed sequence embedding. In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1723–1728. IEEE, 2019.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Denis Grankin (24% of work)

- Coding the main algorithm (CoLES)
- Implementing PySpark preprocessing for clickstream dataset
- Transfering Parquet data as iterable dataset for further embeddings learning
- Experimenting with embedding size for CoLES on both datasets
- Preparing the GitHub Repo
- Experimenting with different downstream tasks

Ekaterina Andreichuk (19% of work)

- Coding the main algorithm (CoLES)
- Coding downstream tasks (*catboost* and others)
- Experimenting with sampling methods and encoders

Ivan Apanasevich (19% of work)

- Coding baseline embeddings extraction methods (aggregate baseline encoder, random feature encoder)
- Visualising data and results for the report and the presentation
- Preparing Section 5 and editing Section 1 of this report.

Irena Gureeva (19% of work)

- Reviewing literate on the topic (9 papers)
- Preparing presentation content
- Preparing Abstract, Section 2 and the Section 3.1 of this report
- Experimenting with clickstream dataset (embedding size) (150 epochs run, embedding sizes)

Mikhail Konenkov (19% of work)

- Reviewing literate on the topic (9 papers)
- Preparing Sections 1-5. (excluding Irena's work and Ivan's work) of this report
- Preparing presentation content
- Experimenting with clickstream dataset (150 epochs run, embedding sizes)

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: The main CoLES method was implemented in the original notebook demo. We have implemented:

   - data preprocessing with Pandas and PySpark,
   - experiments with embeddings' size and accuracy,
   - baseline methods implementation
   - other experiments.

   Our code percentage: $\approx 60\%$

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: The main features of the CoLES method described here; original paper is referenced for more detailed description.

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: We were using datasets from Data Fusion 2022 competition (provided in the task), therefore, data collection is not applicable for our project.

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Section 3.1

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: 80% train, 20% test.

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☐ Yes.
   ☑ No.
   ☐ Not applicable.

   Students' comment: We were not using GridSearch due to computational infrastructure problems and time condiditions; however, we have experimented with different parameters for CoLES model. Also we have tried different downstream tasks; however, Catboost was showing the best performance, that's why we included only this in our report.

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Total number of notebooks is 9. But since we tested different embedding sizes, sampling methods, etc., total general pipeline runs number is around 50.

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

Students' comment: Sections 4.2 and 4.3 of this report.

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: We have included different most applicable scores for our task; the main and most reasonable score in our case is ROC-AUC. ROC-AUC choice as the main metric is justified (Section 4.2).

12. Clearly defined error bars are included in the report.

    - [ ] Yes.
    - [ ] No.
    - [x] Not applicable.

    Students' comment: We don't use measurements to which error bars application is not reasonable.

13. A description of the computing infrastructure used is included in the report.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: Section 4.1.