Записки научных
семинаров ПОМИ
Том 552, 2026 г.

R. Garipov, A. Zabashta, A. Filchenkov

DEEP LEARNING ARCHITECTURE FOR PROCESSING TABULAR DATA TO EXTRACT META-FEATURES

Abstract. This paper study tabular data representation in the field of meta-learning. For this purpose, a deep learning architecture was proposed that is capable of processing tabular data by extracting information from it to a single vector, similar to traditional meta-features extraction approaches. This architecture is based on deep set model and the dataset table is represented as a set of sets. This approach allows to process tabular data in a more natural way because it takes into account the invariance of the dataset with respect to changing the order of rows or columns. Several modifications of this architecture have also been proposed and studied, which can be applied to the task of multi-class classification. The considered architectures were experimentally compared in a traditional algorithm selection problem. The results of the experiment showed that the models that directly processes the dataset outperforms models based only on meta-features, but together these approaches work even better. However, in some scenarios, the convolutional network-based model was superior to the deep set-based model.

§1. Introduction

In tabular datasets, the inherent relationships within the data remain consistent despite the permutation of rows and columns. This characteristic ensures that the outcomes of conventional machine learning algorithms remain unchanged, irrespective of the order in which the data is presented. This property is important to consider when developing meta-learning algorithms. The classical meta-learning approach [2] extracts features, called meta-features, from a dataset without regard to the order of rows and columns. However, this approach extracts fixed statistical and structural features.

Key words and phrases: artificial intelligence and machine learning and deep learning and meta-learning and tabular data.

This paper was carried out as part of ITMO University project No. 623097 "Development of libraries of promising machine learning methods".

Deep neural networks have demonstrated the ability to extract complex dependencies from data and generate features that are sometimes difficult to interpret for humans.

In LM-GAN [8] paper convolutional neural network was used as discriminator for generative adversarial network [5]. Which does not allow working with datasets with a variable number of features, classes and objects of each class. The problem of dataset invariance to permutations of rows and columns is solved by stabilization, which brings invariant datasets to a general form.

In dataset2vec [7] a dataset is perceived as a set of sets. This solves the dataset variable size issue. And the resulting architecture, as required, is invariant to permutations of rows, columns and classes. That is, two identical datasets with different row and column order will still be treated as the same. However, two different datasets can now be considered the same if they have the same set of values in different columns.

A similar idea is used in an paper [3] where a dataset is proposed to be represented as a graph. Since graph architectures are also invariant to permutations of vertices.

In this paper a new deep set-based architecture is developed so that a deep set-based architecture can separate different datasets with the same set of values in different columns.

§2. Extracting information from tabular datasets

2.1. Invariants of tabular datasets.

2.1.1. *Classical meta-feature approach.* This method forms a vector representation of a tabular dataset using so-called meta-features [14]. As mentioned earlier, a tabular dataset is invariant with respect to the permutation of rows and columns. This implies that changing the order of rows and/or columns does not alter the information, dependencies, and patterns within the dataset. Most machine learning algorithms support this invariant to some extent. For instance, if you train a linear regression model on a tabular dataset, rearrange the rows and columns, and then train the model again on the modified dataset, the linear regression will learn the same function for the relationship between features and the target variable, except for a possible permutation of features/coefficients. This is applicable when learning linear regression analytically.

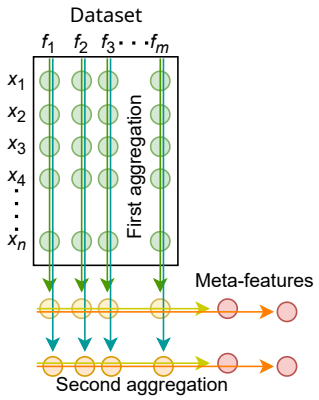

Figure 1. Scheme for constructing meta-features.

2.1.2. *Basic meta-features.* Basic meta-features of tabular datasets are the number of objects, the number of features, the proportion of categorical features, etc.

2.1.3. *Statistical meta-features.* Statistical meta-features are calculations made from the columns of a dataset using functions that do not depend on the order of the data. For example average, median, mode, and variance are suitable functions. A primary aggregation function is applied to all columns of the dataset, and then a secondary aggregation function is applied to the results of the primary aggregation. This secondary function is also independent of the order of the data and produces a single value, known as a meta-feature. By using different combinations of primary and secondary aggregation functions, a description of the dataset in terms of meta-features is created. It is essential that these functions are independent of the order of the data, so that datasets which are the same except for the order of rows and/or columns are represented by the same meta-features.

2.1.4. *Structural meta-features.* Structural meta-features are parameters of a model trained on a tabular dataset. For example, the parameters of a decision tree trained on a tabular dataset. In this case, the structural features will be the tree parameters: the depth of each leaf, the number of vertices at a certain height of the tree, the number of objects in each leaf and any other characteristics of the constructed tree. So these could be,

for example, parameters of a trained linear/logistic regression, parameters of a neural network trained on a tabular datasets, etc.

Based on the constructed meta-features, a model is trained that predicts either the most accurate algorithm from the set A or the accuracy of each algorithm in the set. To achieve this, the meta-features are calculated for each tabular dataset in the training set. To obtain the target variable, all the algorithms from A are needed to run on each tabular dataset. Therefore, the general scheme involves calculating a meta-feature and calculating a target variable for each tabular dataset, and then training a model to solve the meta-learning problem.

One disadvantage of this approach is its simplicity and the use of fixed aggregation functions to construct meta-features. However, it will be demonstrated experimentally that the prediction accuracy of the most accurate algorithm can be increased by using learnable functions for feature extraction.

2.2. Convolution neural network. Since the table is also a two-dimensional matrix, like an image, a convolutional neural network can be used to process table, as was proposed in the LM-GAN paper [8]. However, local pixel relationships are important in an image. Therefore, the convolutional network is not invariant to the permutation of rows and columns in the input matrix.

To solve this problem, the same paper proposed the idea of bringing the dataset into a common form by rearranging the rows and columns to highlight the largest values on the diagonal. This idea is further developed in another paper [15], where it is proposed to reorder rows and columns by proximity or correlation.

This architecture is trained in a generative adversarial setting [5]. It combines the ideas of CGAN [10], since it is a so-called conditional model, and DCGAN [13], since convolutional layers are used.

Also, the discriminator in the LM-GAN architecture has a large number of parameters and can only process tabular datasets of a fixed size.

§3. Deep tables architecture

3.1. Representing table as set of sets. The general idea of deep table architecture is that a dataset can be represented as a set of features, and a feature as a set of values. Thus the data set will be a set of sets.

For processing sets by neural networks, there is a model proposed in the article Deep Sets [18]. The layers proposed in this work will be used in the developed architecture, so it is necessary to describe them.

In this work, 2 layers of neural network are proposed.

$$\text{Invariant Model:} \qquad f(X) = p\left(\sum_{x \in X} \phi(x)\right)$$

where X is a set, ϕ is a transformation function of the set element e.g linear function and p is a final transformation, e.g normalization function.

$$\text{Equivariant Model:} \qquad f(x) = \sigma(\phi(x) + \gamma \cdot \texttt{maxpool}(X))$$

where x is an element of the set, ϕ is a transformation function of the set element, γ is a pooling coefficient and σ is an activation function.

The Invariant model provides a method for aggregating a set into a single number by summing the values obtained from transforming the elements of the set and then applying the function p, which can, for example, normalize the resulting value. For N sets each of size M, or a matrix of size (N, M), the application of the Invariant model function results in a column vector of size N. If each element of the set is represented as a vector of size H_0, that is, we are dealing with a tensor of size (N, M, H_0), then the application of the function yields a matrix of size (N, H_0).

The Equivariant model is a method for transforming the elements of a set. The transformation ϕ is applied to an element of the set, and then the maximum element of the set, multiplied by the parameter γ, is added to the result. An activation function is then applied to this sum. As a result, when the Equivariant model function is applied to N sets of size M, or a matrix of size (N, M), a matrix of the same size is obtained. When this function is applied to N sets of size M, where each element of the set is represented as a vector of size H_0, or a tensor of size (N, M, H_0), the result is a tensor of size (N, M, H_1).

It should be noted that the transformation ϕ can be a linear operation, such as multiplication by a scalar when the elements of the set are numbers, or multiplication by a matrix when the elements of the set are represented as vectors. In these cases, the coefficients used in the multiplication will be the parameters of the model.

In summary, the dimensions of the tensors after processing by these layers can be described by the following scheme:

- Invariant model: $(N, M) \to (N)$ and $(N, M, H_0) \to (N, H_1)$
- Equivariant model: $(N, M) \to (N, M)$ and $(N, M, H_0) \to (N, M, H_1)$

§4. Proposed deep table solution

The developed architecture is called Deep Table. As can be understanded from the description of the Invariant Model and Equivariant model layers, by combining them a deep neural network can be constructed. The Invariant Model allows to aggregate a set into a number, and the Equivariant model allows to apply a transformation to all elements of the set. Thus, while studying the article in which these layers were proposed, the idea arose to use these layers to aggregate a tabular in the same way as is done in the classical meta-learning approach.

First, each column is transformed as a set and the Equivariant Model operation is applied to it, increasing the size of the vector that represents each element of the column. Next, the Invariant Model is used to aggregate each set into a vector. Each column is represented by several aggregates. Next, these aggregate vectors are transformed using the Equivariant Model operation. And then the Invariant Model again is appled to aggregate many vectors into one final vector, which will be a vector of deep meta-features. After each transformation of the Equivariant Model the ReLU activation function [4] is used in the best traditions of deep learning. Invariant Model and Equivariant Model are just linear, fully connected layers.

General diagram of the proposed architecture:

(1) Meta-features are calculated from the dataset.
(2) The dataset is fed into a neural network block, which aggregates the tabular dataset into a vector of deep meta-features.
(3) The vector of deep meta-features concatenated with the vector of meta-features is supplied as input to the classifier.

The model is trained in a classification problem setting.

Since, for training, it is necessary to go through all tabular datasets from the training set in several epochs, and for each as many times as the number of epochs allocated for training, it is logical to calculate meta-features once in advance for each and save them to a file, so as not to calculate again every time.

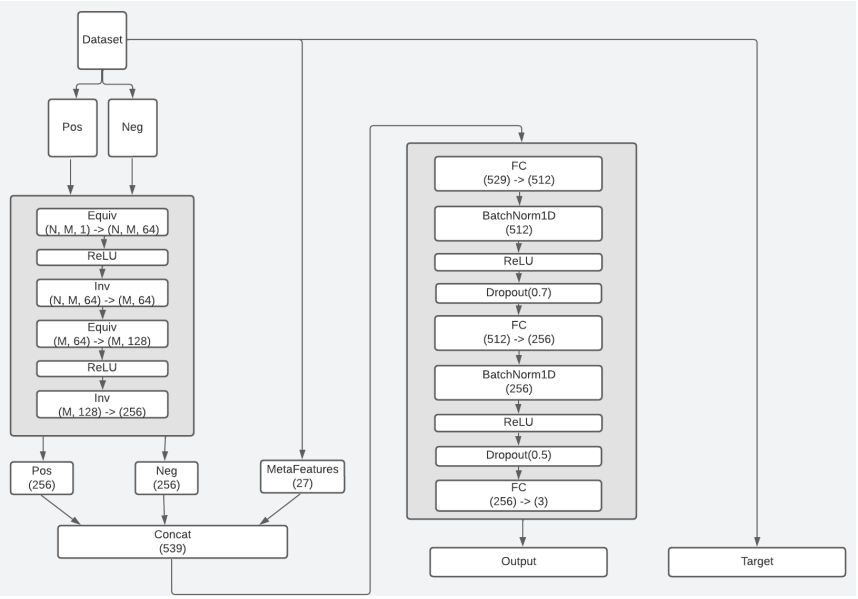

Figure 2. Deep Table architecture for the described meta-learning task.

4.1. Deep table neural network. In general, a tabular dataset is fed individually to the part of the neural network that is responsible for aggregating the tabular dataset into a deep meta-feature vector. Thus, a vector of deep meta-features is formed for each class, they are all concatenated together with the vector of meta-features extracted in the classical way. Then they are fed to the input of the classifier, which looks like a classic neural network classifier, consisting of alternating linear layers, batch normalization [6], activation function ReLU [4] and Dropout layer [16]. At the same time, the sizes of hidden states decrease as we move through the layers of the classifier.

For datasets with a fixed number of classes, as in this work, it is possible to concatenate the vectors of each classes, although then the architecture will not be invariant to class permutation. In general, another deep set model can be applied to the resulting vectors for each class. Thus, a dataset will be a set of sets of sets.

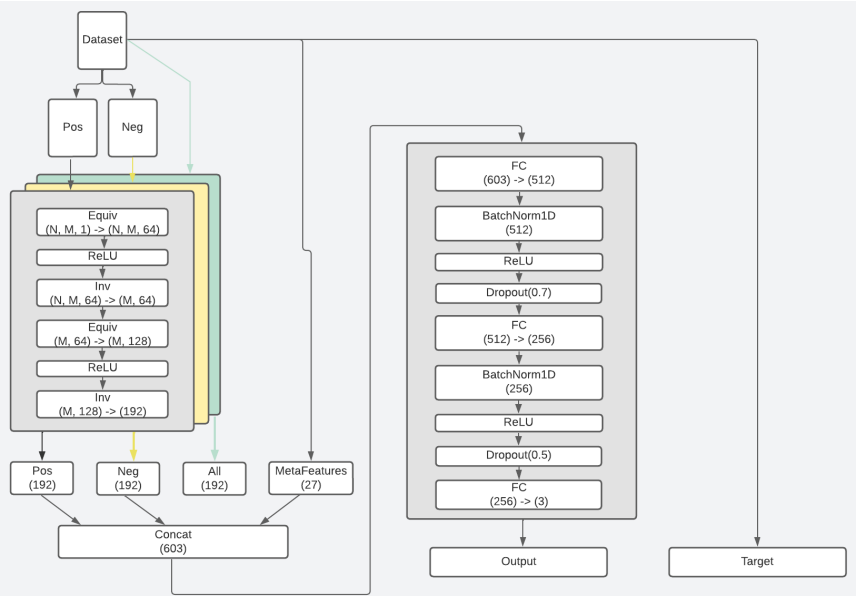

Figure 3. Deep table architecture, multiple extractors.

This architecture satisfies the following requirements: it aggregates a tabular dataset in a way that is invariant to permutation of the rows and/or columns of the tabular dataset, can process datasets of arbitrary size, since all operations with a tabular are performed as if it were a set.

However, such an architecture will be too invariant to bad permutations. If the values of features in one part of the data set were swapped, then in the general case the encoded hidden dependence in it will change, but according to the considered architecture, this dataset will not change.

4.2. Neural network deep table multiple extractors. This architecture differs from the previous one in that for each class, as well as for the entire dataset, separate neural network blocks are used that aggregate the tabular dataset into a vector of deep meta-features. Moreover, architectures of these blocks are absolutely identical, but the parameters are not shared, so they will all learn different dependencies during the neural network training process. Similarly, the deep meta-feature vectors are concatenated all together.

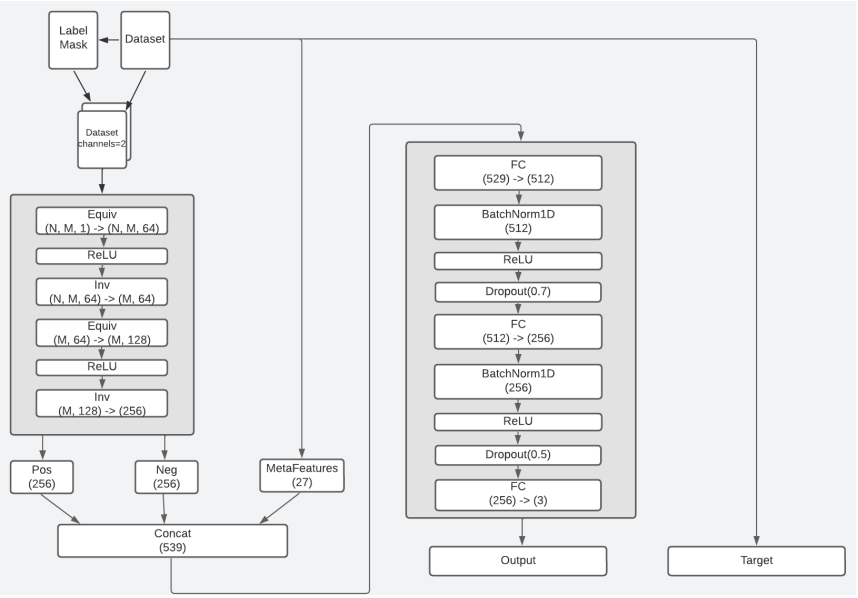

Figure 4. Deep table architecture, label channel.

The idea behind this architecture is that an additional extractor captures information about the entire dataset so that the resulting architecture is not invariant to bad permutations.

4.3. Neural network deep table label channel. In this architecture, class label is represented in the additional input channel of the tabular dataset fed into the neural network.

This work considers only datasets for binary classification, so the additional layer directly encodes the class. In the case of multi-class classification, it is possible to one-hot encode the class and use an additional channel to indicate whether a given feature is a target or not.

4.4. Deep table flattened neural network. In this approach, a tabular dataset is represented as a single vector, that is, all its rows are concatenated. And the sequences Eqiuvariant Model, Invariant Model are supplied as input. And from them we get a vector of deep meta-features, calculated

from one set, where each element is represented by vectors whose size is equal to the size of the Equivariant Model hidden state.

Thus, this architecture does not take into account the tabular structure of the dataset at all so it is the baseline architecture for deep table approaches.

§5. Experiments

5.1. Meta-dataset of datasets. Tabular datasets for the binary classification problem were considered. The tabular datasets were the same as in the LM-GAN paper [8]. There, several datasets were selected for multi-class classification from the OpenML [17] archive. In each such dataset, pairs of classes were iterated to form a tabular dataset for binary classification. They were formed so that each dataset consists of 64 elements of the positive and 64 elements of the negative class. The training set consists of 8000 tabular datasets, the test set consists of 1911 tabular datasets. When splitting, all binary classification datasets from one multi-class classification dataset were added only to the training or only to the test part of meta-dataset. This was done to prevent overfitting.

The proposed architecture is potentially capable of processing datasets with a variable number of classes, features, and objects of each class. But for the experiments, fixed datasets were chosen so that the deep set-based model could be compared with the convolutional network-based model.

5.2. Algorithm selection problem. The experiments considered a portfolio of three following classification algorithms with specific hyperparameters:

(1) KNN(n_neighbors=3)
(2) SVM(kernel=rbf, gamma=2, C=1)
(3) Gaussian Naive Bayes(var_smoothing=10^{-9})

This choice was made in a previous study. It is due to the fact that such algorithms give a good distribution of meta-classes. Implementations of these algorithms were taken from the *scikit-learn* library [12].

For each dataset, each of the considered classification algorithms was evaluated by cross-validation with 3 splits. The target function in cross-validation was accuracy.

As a result, several algorithms could achieve the same accuracy and turn out to be the best. Therefore, multi-label classification was considered as the meta-prediction task. Accuracy was also used as quality measure for

this task. Thus, the target value for each is a vector of size 3, in which the element values are either zero or one, one corresponding to the algorithm that is most accurate on the given dataset.

5.3. Experiment design and results. The considered models were experimentally compared with the classical approach based on pure meta-features. For this purpose, implementations of classification algorithms from the *scikit-learn* library [12] were used. The implementation from previous paper similar to *PyMFE* library [1] was used to extract meta-features. Statistical, information theory and decision tree model-based meta-features were extracted. Basic meta-features were not used since the datasets were of a fixed size. A total of 27 meta-features were used.

Deep learning models were implemented with *pytorch* framework [11]. They trained using *BCEWithLogitsLoss* function and Adam [9] optimizer. Deep learning-based models are capable of working with multimodal data. Therefore, they were tested with only datasets and with a composition of dataset and its meta-features.

- **MF**: model takes meta-features as input.
- **SD**: model takes a stabilized dataset as input.
- **RD**: model takes a dataset as input.

The results of the experiments are presented in Table 1. The results of the experiment showed that the models that directly processes the dataset outperforms models based only on meta-features, but together these approaches work even better. However, in some scenarios, the convolutional network-based model was superior to the deep set-based model.

§6. Conclusion and future work

This paper proposes a deep learning architecture for working with tabular data, such as in meta-learning, where datasets are used as meta-objects. The considered architectures were experimentally compared in a traditional algorithm selection problem.

The result showed that the deep set-based architecture is capable of extracting information from tables when they represented as a set of sets. However, the dataset for the classification task is more complex than the table, since it also contains information about the classes, which is invariant under their permutation.

Model	Input	Accuracy
Decision Tree	MF	0.186 ± 0.000
kNN	MF	0.286 ± 0.000
SVM	MF	0.228 ± 0.000
Logistic Regression	MF	0.173 ± 0.000
MLP	MF	0.300 ± 0.012
CNN	SD	0.463 ± 0.006
CNN	MF + SD	0.527 ± 0.005
Deep Table	RD	0.438 ± 0.007
Deep Table Flattened	RD	0.418 ± 0.010
Deep Table Multiple Extractors	RD	0.449 ± 0.010
Deep Table Label Channel	RD	0.441 ± 0.002
Deep Table	MF + RD	0.578 ± 0.006
Deep Table Flattened	MF + RD	0.576 ± 0.009
Deep Table Multiple Extractors	MF + RD	0.577 ± 0.002
Deep Table Label Channel	MF + RD	0.571 ± 0.008

Table 1. Comparison of models in the algorithm selection problem.

In addition to the algorithm selection problem, the proposed architecture can be applied to other meta-learning problems. For example, to generate new datasets using generative adversarial networks or diffusion models.

References

1. E. Alcobaça, F. Siqueira, A. Rivolli, L. Garcia, J. Oliva, and A. Carvalho, *MFE: Towards Reproducible Meta-Feature Extraction*, Journal of Machine Learning Research **21** (2020), 1–5.
2. P. Brazdil, J. van Rijn, C. Soares, and J. Vanschoren, *Metalearning: Applications to Automated Machine Learning and Data Mining*, Springer Nature, 2022.
3. G. Drozdov, A. Zabashta, and A. Filchenkov, *Graph Convolutional Network Based Generative Adversarial Networks for the Algorithm Selection Problem in Classification*, in: Proceedings of the 2020 1st International Conference on Control, Robotics and Intelligent System, 2020, pp. 88–92.
4. K. Fukushima, *Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements*, IEEE Transactions on Systems Science and Cybernetics **5** (1969), 322–333.

5. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative Adversarial Nets*, in: Advances in Neural Information Processing Systems **27**, 2014.
6. S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning, Vol. 37, 2015, pp. 448–456.
7. H. Jomaa, L. Schmidt-Thieme, and J. Grabocka, *Dataset2Vec: Learning Dataset Meta-Features*, Data Mining and Knowledge Discovery **35** (2021), 964–985.
8. I. Kachalsky, A. Zabashta, A. Filchenkov, and G. Korneev, *Generating Datasets for Classification Task and Predicting Best Classifiers with Conditional Generative Adversarial Networks*, in: Proceedings of the 3rd International Conference on Advances in Artificial Intelligence, 2019, pp. 97–101.
9. D. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2017.
10. M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, 2014.
11. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, in: Advances in Neural Information Processing Systems **32**, 2019, pp. 8024–8035.
12. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, *Scikit-Learn: Machine Learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.
13. A. Radford, L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 2016.
14. A. Rivolli, L. Garcia, C. Soares, J. Vanschoren, and A. Carvalho, *Meta-Features for Meta-Learning*, Knowledge-Based Systems **240** (2022), 108101.
15. I. Sahipov, A. Zabashta, and A. Filchenkov, *Stabilization of Dataset Matrix Form for Classification Dataset Generation and Algorithm Selection*, in: Intelligent Data Engineering and Automated Learning — IDEAL 2020, 2020, pp. 66–75.
16. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research **15** (2014), 1929–1958.
17. J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo, *OpenML: Networked Science in Machine Learning*, SIGKDD Explorations **15** (2013), 49–60.
18. M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, *Deep Sets*, in: Advances in Neural Information Processing Systems **30**, 2017.

ITMO University,
Kronverksky Prospekt 49, bldg. A,
St. Petersburg, 197101, Russia
E-mail:, R. Garipov devilgar@gmail.com
E-mail:, A. Zabashta azabashta@itmo.ru
E-mail:, A. Filchenkov aafil@mail.ru

Поступило 4 февраля 2026 г.