Discrete & Continuous Models
& Applied Computational Science

2020, 28 (2) 105–119
http://journals.rudn.ru/miph

Computer Science and Computer Engineering

Research article

UDC 519.872:519.217
DOI: 10.22363/2658-4670-2020-28-2-105-119

Comparative analysis of machine learning methods by the example of the problem of determining muon decay

Migran N. Gevorkyan[1], **Anastasia V. Demidova**[1], **Dmitry S. Kulyabov**[1,2]

[1] *Department of Applied Probability and Informatics*
Peoples' Friendship University of Russia (RUDN University)
6, Miklukho-Maklaya St., Moscow, 117198, Russian Federation

[2] *Laboratory of Information Technologies*
Joint Institute for Nuclear Research
6, Joliot-Curie St., Dubna, Moscow region, 141980, Russian Federation

(received: May 21, 2020; accepted: June 30, 2020)

The history of using machine learning algorithms to analyze statistical models is quite long. The development of computer technology has given these algorithms a new breath. Nowadays deep learning is mainstream and most popular area in machine learning. However, the authors believe that many researchers are trying to use deep learning methods beyond their applicability. This happens because of the widespread availability of software systems that implement deep learning algorithms, and the apparent simplicity of research. All this motivate the authors to compare deep learning algorithms and classical machine learning algorithms.

The Large Hadron Collider experiment is chosen for this task, because the authors are familiar with this scientific field, and also because the experiment data is open source. The article compares various machine learning algorithms in relation to the problem of recognizing the decay reaction $\tau^- \to \mu^- + \mu^- + \mu^+$ at the Large Hadron Collider. The authors use open source implementations of machine learning algorithms. We compare algorithms with each other based on calculated metrics. As a result of the research, we can conclude that all the considered machine learning methods are quite comparable with each other (taking into account the selected metrics), while different methods have different areas of applicability.

Key words and phrases: muon decay, machine learning, neural networks

1. Introduction

Machine learning is a branch of mathematical modeling related to the construction of surrogate statistics models. Recent years this area has been experiencing really intensive growth, related to the development of computer technology and the ability to analyze grate amount of data (Big Data). Nowadays machine learning approaches, in particular deep learning, demonstrate their high efficiency in data science. Particularly significant results are obtained in classification and cluster analysis of data with unknown structure. The most popular tendency in machine learning is deep learning. It became mainstream area in machine learning and other areas were pushed aside.

In this paper, the authors try to study if deep learning is really superior to all other machine learning methods. Previously, the authors conducted a comparative analysis of the most popular software products for working with neural networks networks [1], and also tried to generalize the methodology for working with machine learning models [2].

1.1. Paper structure

This paper has following structure. In section 2 we describe the problem of the decay reaction recognition $\tau^- \to \mu^- + \mu^- + \mu^+$. A brief introduction to the physics of the process is given.

The section 3 briefly describes the software we use.

The 4 section briefly describes the classification task, provides the terminology from the field of machine learning, we also consider metrics that are used to evaluate efficiency of classifiers.

We apply the Python language and the modules described to the problem in the section 3 . We use metrics to evaluate the effectiveness of various machine learning methods.

2. The violations of the Standard model

Currently, the main model that describes particle physics is a Standard model formulated in 1960–1970 [3]. Standard model it has passed many experimental tests. However, with from a methodological point of view, this theory is not satisfactory [4]. For example, the Standard model does not describe a number of phenomena, such as explanation of matter–antimatter asymmetry. Research in the field of theoretical and experimental physics that try to expand the standard model and describe phenomena that are not available to it have a collective name: physics beyond the standard model.

2.1. Preservation of lepton numbers

The Large Hadron Collider (LHC) is the main tool for studying physics beyond the Standard model. At the LHCb detector (LHG beauty experiment) experiments are being performed [5] whose purpose is the detection of phenomena that contradict theoretical settings of standard model. In particular, one of these phenomena is associated with violation of preserving the lepton

number (L) and the lepton flavor (L_e, L_μ, L_τ). For leptons, the heuristic division into three generations, which is necessary for existence asymmetries of matter and antimatter:

— the first generation consists of an electron and an electron neutrino (e^-, ν_e),
— second generation — muon and muon neutrino (μ^-, ν_μ),
— third generation — τ-lepton (tau) and tau neutrino (τ, ν_τ).

As we can see from the Table 1, according to the standard model each lepton has four numbers L_e, L_μ, L_τ and L and for every reactions between particles the sum of the numbers on the right side of the reaction equation must be equal to the sum of the numbers on the left side (Lepton number conservation).

Table 1

Reactions between particles in the standard model

Particle	e^-	e^+	μ^-	μ^+	τ^-	τ^+	ν_e	$\bar{\nu}_e$	ν_μ	$\bar{\nu}_\mu$	ν_τ	$\bar{\nu}_\tau$
L	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
L_e	+1	-1	0	0	0	0	+1	-1	0	0	0	0
L_μ	0	0	+1	-1	0	0	0	0	+1	-1	0	0
L_τ	0	0	0	0	+1	-1	0	0	0	0	+1	-1

This rule holds, for example, in the following tau decay reaction:

$$\tau^- \to e^- + \nu_\tau + \bar{\nu}_e,\ 1_\tau = 1_e + 1_\tau - 1_e.$$

However, there is a hypothetical tau decay reaction of the following type:

$$\tau^- \to \mu^- + \mu^- + \mu^+,\ 1_\tau \neq 1_\mu + 1_\mu - 1_\mu.$$

Ultrahigh energies proton collisions are performed at the LHC. On average the collision generates about 80 various particles, most of which are unstable and fast disintegrate. Among them, there are tau that can occur in one of the the next five reactions:

— Prompt $D_s^- \to \tau$,
— Prompt $D^- \to \tau$,
— Non-prompt $D_s^- \to \tau$,
— Non-prompt $D^- \to \tau$,
— $X_b \to \tau$.

The task is to build a classification model that must be trained to recognize the decay reaction $\tau^- \to \mu^- + \mu^- + \mu^+$. For training the classifier one [6] provides real data from LHC (background events) with the addition of signal data (signal events). The signal data is a simulation of the reaction $\tau^- \to \mu^-\mu^-\mu^+$.

The classifier requires the following two properties.

— Small discrepancy between real data and simulation. For Estimation of discrepancy data `check_agreement.csv` is provided This data relates to the reaction $D_s^+ \to \phi(\to \mu^-\mu^+)\pi^+$ which is topologically very similar to the desired response of the decay τ^-. Also the value of the Kolmogorov–Smirnov test coefficient must be less than 0.09.
— Also the classifier should have weak correlation with the mass τ^-. Data in a file is provided to evaluate the correlation `check_correlation.csv` and the Kramer–von Mises test (CvM).

3. Software

To apply all the described classification methods, we use Python language and a number of modules: SciKit Learn [7], Keras [8], `XGBoost` [9] and `hep_ml` [10]. Let's give a brief description here for each of them.

SciKit Learn [7] is library for data processing, which implements various methods of classification, regression analysis, clustering, and other algorithms related to machine learning training that does not use neural networks. The library is written in Python and uses a number of libraries from the SciPy stack to accelerate calculations. The current version has the number 0.22.2, but the project is quite mature.

SciKit Learn implements almost all of classifications algorithms we described. So the Logistic Regression method is implemented in a submodule `linear_model`, Gaussian Naive Bayes method is in the submodule `naive_bayes`, the `ensemble` submodule implements Random Forest and Gradient Boosting Classifier methods. In the submodule `sklearn.metrics` there are functions that calculate various metrics for estimation of quality of the classifier.

The XGBoost library is considered the best implementation of gradient boosting. It has API for many languages, including Python. We use it along with SciKit Learn to apply Gradient Boosting Classifier. Also due to the specifics of the task we use `hep_ml` module because it is specially designed for physics problems.

The Keras [8] library provides a high-level software interface for building neural networks. It can work on top of TensorFlow, Microsoft Cognitive Toolkit (CNTK) [11] or Theano [12]. The library is written completely in Python and distributed under the MIT license. Current version is 2.3.1. The library is based on the following principles: simplicity usage, modularity, and extensibility. Our choice of this libraries about is justified in the article [1].

The modularity principle allows one to describe neural layers separately, optimizers, activators, and so on, and then combine them into one model. The model is fully described in Python. Created model one can save to disk for future use and distribution.

4. Classification models

The classification model is based on an array of data, presented in tabular form. The process of model construction is usually consists of fitting numeric parameters and is also called *model training*. The propose of the model is to predict the value *dependent variable*. In the case of a binary classifier

a dependent variable can only take two values: 0 or 1. In in this case, the dependent variable is most often called *binary response* or just *response*. One can also meet the terms: goal, outcome, label, and Y–variable [13]–[15].

The model parameters are adjusted based on independent parameters variables that are represented by columns of the table. The following terms are also used: *predictor variable*, attribute and X–variable.

There are two types of predictor variables *numeric* and *factorial* (another term — *categorical*) predictor variables. Numeric variables are continuous and can take any values from some interval on the numeric axis, and the factor variables are discrete, not necessarily numeric, and can take values from a finite set. A special type of factor variables are *indicator* variables. Such variables accept only two values (0 or 1).

Depending on the model, it may be necessary to convert factor variables to numeric values or numeric to factor. So when applying multiple linear regression to an array of data with factor variables we need to convert them to numeric type. For example, we can use logit conversion. On the contrary, using the naive Bayesian classifier to continuous data, this data must be converted to factor type.

4.1. Metrics for evaluating classification models

A number of numerical methods are used to evaluate the classifier's performance characteristics (metrics) that allow us to compare different classifiers with each other and choose the most optimal one for the given tasks [14].

The classifier is evaluated based on *control* sample (also called *test* or *verification* sample). This sample consists of already classified elements and allows one to measure the performance of the classifier.

Let's assume that the size of the control sample is N and the binary classifier detects the response Y and assigns it 1 or 0. Since this detection is performed on the basis of a control sample, the event class is already known and we can check classification results. All possible predictions fit into four case.

1. True-positive (TP) — classification result is 1 and true value is 1;
2. False-negative (FN) — classification result is 0 but true value is 1;
3. False-positive (FP) — classification result is 1 but true value is 0;
4. True-negative (FN) — classification result is 0 and true value is 0.

Let's describe the main metrics that are used for classifier evaluation and specify functions from the module `sklearn.metrics` [7], [16], which are used to calculate this metrics.

Let the total sample size is N, and the classifier has defined TP true-positive, FN false-negative, FP false-positive and TN of true-negative cases. We can calculate the following table 2 called the *confusion matrix*.

The classification of metrics is based on this matrix. It shows the number of correct and incorrect predictions grouped into categories by response type. Other names of this matrix are error matrix or confusion matrix. To calculate this matrix we use the `confusion_matrix` function from SciKit–Learn library.

Accuracy is calculated as the percentage of events that the classifier identified correctly. Calculated using the formula:

$$Acc = \frac{TP + TN}{N},$$

Table 2

The confusion matrix

		Prediction		
		True (1)	False (0)	Total
Data	True (1)	TP	FN	$TP + FN$
	False (0)	FP	TN	$FP + TN$
	Total	$TP + FP$	$FN + TN$	N

and using the `accuracy_score` function.

Recall is the percentage of correctly classified events of type 1. Calculated using the formula:

$$R_{TP} = \frac{TP}{TP + FN},$$

and using the `recall_score` function. Terms are also used are *sensitivity* or true-positive rate.

Specificity is percentage of correctly classified events of type 0 (also called zeros). Calculated using the formula:

$$R_{FP} = \frac{TN}{TN + FN},$$

and also using the `recall_score` function (for binary classifier this function returns both recall and specificity). The term false-positive rate is also used.

Precision is percentage of predicted units that are actually zeros. Calculated using the formula:

$$Pcn = \frac{TP}{TP + FP},$$

and also using the `precision_score` function.

One can create a classifier that will relate all events to class 1. For such a classifier, the recall will be equal to 1, and specificity 0. An ideal classifier should detect events from class 1, without incorrectly identifying events of class 0, as events of the 1 class. Thus a balance must be maintained between recall and specificity. To evaluate this balance, one uses a graphical method called *ROC-curve* — receiver performance curve.

The ROC curve is a graph of recall versus specificity. For plotting on one axis is delayed recall, and on the other specificity. The graph of an absolutely ineffective classifier will be represent a diagonal line. More effective classifiers will have a graph in the form of an arc. The stronger the arc pressed against the upper-left corner, the more effective it is classifier. The data required to build the curve is calculated with the `roc_curve` function.

For a more accurate estimation of the ROC curve, one uses a metric indicator AUC — *area under the ROC curve*. A classifier with a ROC curve as a diagonal line will have $AUC = 0.5$. The more effective the classifier, the closer the AUC value is to 1. AUC is calculated by the `auc` function.

4.2. Logistic Regression

Logistic regression [17], [18] is an analog of multiple linear regression, with the exception of binary response. To adapt multiple linear regression for this case is necessary to do fallowing steps:

— represent the dependent variable as a probability function, with values from segment $[0, 1]$ (probabilistic outcome);
— apply the cutoff rule — any outcome with probability, greater than the threshold is classified as 1.

If classical multiple regression models the response as linear function from predictor variables:

$$y = \beta_0 + \beta_1 x_1 \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n,$$

then the logistics response function is modeled using the logistics response function (*logit-function* or *sigmoid*):

$$p = \frac{1}{1 + \exp(\beta_0 + \beta_1 x_1 \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n)}.$$

The range of values of such function is the interval $(0, 1)$, we can interpret its values as the probability of the response.

To fit parameters, we consider not the function itself, but the log-odds function:

$$l = \ln \frac{p}{1-p} = \beta_0 + \beta_1 x_1 \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n,$$

which map the probability p from the interval $(0, 1)$ to real numbers set. Then one uses the maximum likelihood method to select parameters based on a training sample.

After selecting the parameters it remains to select cut-off threshold. For example, if one puts it equal to 0.5, then all the response with value $p < 0.5$ will be classified as 0, and with the value $p >= 0.5$ as 1.

In the `sklearn` library, the function that implements the logistic regression algorithm is located in the `linear_model` module and it is called `LogisticRegression`.

4.3. Gaussian Naive Bayes

Naive Bayesian classifier [15], [19] is a binary classifier. Assignment to a particular class is based on the conditional probability $p(y|x_1, ..., x_n)$ which is calculated based on the Bayes theorem:

$$p(y|x_1, ..., x_n) = \frac{p(y, x_1, ..., x_n)}{p(x_1, ..., x_n)} = \frac{p(x_1, ..., x_n|y)p(y)}{p(x_1, ..., x_n)}.$$

Next, we make the «naive» statement that all predictor variables are independent and, therefore, the joint probability is $p(y, x_1, ..., x_n)$ and it can be calculated using the formula:

$$p(y, x_1, \dots, x_n) = p(y) \prod_{i=1}^{n} p(x_i|y).$$

If predictor variables are assumed to be numeric (i.e. continuous values), then a second «naive» statement is made about the continuity of the distribution function and about the type of distribution. Most often, the normal distribution and a Gaussian Naive Bayes classifier are used.

The advantages of a Naive Bayes classifier include simplicity (there are only few hyperparameter settings) and speed.

In the `sklearn` library, the function implementing the Gaussian Naive Bayes classifier algorithm is located in the `naive_bayes` module and is called `GaussianNB`.

4.4. Bagging and Random Forest

Tree models [20] or *decision trees* is a popular, relatively simple, and yet effective classification method.

Decision trees define a set of classification rules. The rules correspond to the sequential split of the data into segments. Each rule can be expressed as a «if-then» condition imposed on a predictor variable. For each predictor, *split value* is defined, which divides records into those where the value of the predictor variable is greater and those where it is less. A set of such rules forms a tree whose leaves correspond to one of the two required classes (for a binary classifier).

Tree models advantages is the simplicity of the results interpretation and the ability to reproduce the branching rules in natural language. However, one should avoid overtraining of these models. Overtraining means that branching rules start to take random noise into account. To prevent overtraining, one should limit the depth of tree branches.

Trees became particularly popular with the introduction of the *ensemble approach*. Its essence is to use a set of decision trees and train them on the same data with further taking the average or weighted average of their results.

Among the methods of training, a method called *bagging* or *bootstrap aggregation*. The bootstrap process involves repeatedly retrieving a random set of data from a sample. The number of extracted records is less than the sample size. The most common is bootstrap with replacement. Replacement means that the extracted data is returned to the sample after use, mixed, and used for subsequent retrievals. The begging process consists of training trees on multiple bootstrap samples with returns.

The *random forest* machine learning algorithm uses begging and selects predictor variables in addition to bootstrap. In other words, each new tree is built on a random subset of variables, rather than on all possible variables. There is empirical rule that it is most efficient to select only $\sqrt{n}$ predictor variables from n each time.

In the `sklearn` library, the function that implements the random forest algorithm is located in the `ensemble` module and is called `RandomForestClassifier`.

4.5. Gradient Boosting Classifier

Gradient boosting method [21] consists of combining a large number of simple models to produce one that is more accurate than each individual simple model itself. A set of simple models is called *ensemble*, and by boosting we mean the sequential process of building simple models.

The gradient boosting algorithm is one of the most commonly used machine learning algorithms. We will give only a brief qualitative description here, without going into mathematical details [22], [23].

At each step of gradient boosting, the selected loss function is minimized by gradient descent. The loss function is constructed for the selected base algorithm. Most often the underlying algorithm is the decision tree algorithm. When building each subsequent model, the errors of the previous one are taken into account. This is done by defining the data that does not fit into the previous simple model and adding the next model that processes this data correctly. When configuring the algorithm, the maximum number of models in the ensemble is specified, and when this number of iterations is reached, the algorithm stops. Each model from the ensemble is assigned a certain weight and their predictions are generalized.

In the `sklearn` library, the function that implements the gradient boosting algorithm is located in the `ensemble` module and is called `GradientBoostingClassifier`.

In addition to the implementation included in `scikitlearn`, Python also has the `XGBoost` [9] library, which is highly optimized and has interfaces for a large number of programming languages (C/C++, Java, Ruby, Julia, R).

In addition to the implementations from these two libraries, we used the gradient boosting implementation from the `hep_ml` [10] library, which contains machine learning methods used in the field of high-energy physics.

4.6. Neural Network

In the article [1], the authors compared various libraries for building neural networks [8], [12], [24]–[26]. The result of the speed and the accuracy tests show that Keras library provides the most optimal solution. Therefore, to solve the problem of recognizing the decay reaction $\tau^- \to \mu^- + \mu^- + \mu^+$ we build neural network using this library.

5. Application of the considered methods

We carry out a comparative analysis of classifiers from section 4 by applying them to the problem of determining muon decay. The problem is a binary classification problem and is based on data from the LCH and generated data for detecting muon decay. Training and test data are presented in csv files. The data contains the values of 40 analyzed parameters. The target attribute is the «signal» attribute, which takes the values 0 or 1. For training classifiers, the data set was divided into training and test samples in the ratio of 8 to 2, the number of records for training classifiers is 54042, and the number of records for testing is 13511.

We choose MLP (multi-layer perceptron) architecture for the neural network. The network consists of fully connected layers with Batch Normalization layers

between them that prevent overtraining. Each of the fully connected layers contains a different number of neurons. The input layer consists of 28 neurons, the hidden layers contain 100, 120, 60, and 20 neurons, and the output layer contains 2, according to the number of classes in the data.

All classifiers were tested on a small discrepancy between real data and simulation (Kolmogorov–Smirnov test, the test value for the classifier should be less than 0.09) and a weak correlation with ground test (Cramer–von Mises (CvM), the test value for the classifier should be less than 0.002) In the table 3 lists the values of these tests.

Table 3

Results of the Kolmogorov–Smirnov and the Kramer–von Mises tests

Classification	Kolmogorov–Smirnov test	Kramer–von Mises test
Random Forest	0.03682	0.00092
Logistic Regression	0.03309	0.00103
Gaussian Naive Bayes	0.04722	0.00113
Gradient Boosting Classifier	0.05162	0.00089
Xgboost	0.06327	0.00089
UGradient Boosting Classifier	0.05587	0.00102
MLP	0.01139	0.00079

For all classifiers, the main metrics are calculated (table 4) for test data, and the results of comparing the classifiers are presented as a diagram (Figure 1).

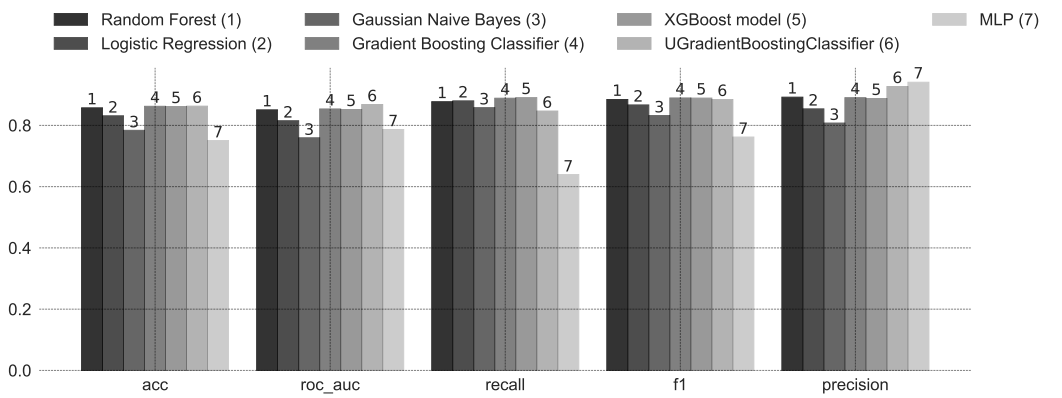

Figure 1. The results of the comparative analysis of classifiers

Table 4

The metric values on the test data

Classification	Accuracy	ROC-AUC	Recall	F1	Rrecision
Random Forest	0.857	0.851	0.877	0.885	0.892
Logistic Regression	0.831	0.815	0.880	0.867	0.854
Gaussian Naive Bayes	0.784	0.759	0.858	0.832	0.808
Gradient Boosting Classifier	0.862	0.854	0.889	0.889	0.890
xgboost	0.861	0.853	0.890	0.889	0.888
UGradient-BoostingClassifier	0.863	0.868	0.847	0.885	0.927
MLP	0.750	0.787	0.640	0.762	0.941

6. Discussion

The paper presents a comparative analysis of various machine learning algorithms on the example of the problem of determining the decay reaction $\tau^- \to \mu^- + \mu^- + \mu^+$ at the LHC. We study following algorithms: Logistic Regression, Gaussian naive Bayes classifier, gradient boosting classifier, bootstrap aggregating (bagging) and random forest, neural network model (machine learning algorithm — MLA). For each of the algorithms, we build a classifier using Python libraries and calculate metrics calculated that can be used to determine the most effective model.

All classifiers successfully passed tests for a small discrepancy between real data and simulation (Kolmogorov–Smirnov test) and for a weak correlation with mass (Kramer–von Mises test), which indicates a good quality of the constructed classifiers for this problem.

To conduct a comparative analysis of machine learning methods, we calculate the most important metrics for each model: accuracy, ROC–AUC score, recall, F1-score, precision. In the aggregate of all metrics, the random forest and the gradient boosting method (and their modifications) have the best results. Logistic Regression, Gaussian Naive Bayes and a model based on a fully connected neural network show worse results. However, the neural network surpass other classifiers by the value of the precision metric. This means that the neural network can better distinguish classes from each other than other classifiers.

7. Conclusion

A comparative analysis of various machine learning algorithms is carried out on the example of the problem of determining the decay reaction $\tau^- \to \mu^- + \mu^- + \mu^+$ at the Large Hadron Collider. As the compared algorithms were chosen: Logistic Regression, Naive Bayesian approach with normal distribution, the method of gradient boosting (Gradient boosting classifier), bootstrap aggregation in combination with random forest, a model based on a neural network (machine learning algorithm—MLA). For each of the algorithms, using the libraries for the Python language, a classifier was built and metrics were calculated, based on which the most effective model can be determined.

Acknowledgments

The publication has been prepared with the support of the Russian Foundation for Basic Research (RFBR) according to the research project No 19-01-00645.

References

[1] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, and A. A. Sobolev, "Review and comparative analysis of machine learning libraries for machine learning," *Discrete and Continuous Models and Applied Computational Science*, vol. 27, no. 4, pp. 305–315, Dec. 2019. DOI: 10.22363/2658-4670-2019-27-4-305-315.

[2] L. A. Sevastianov, A. L. Sevastianov, E. A. Ayrjan, A. V. Korolkova, D. S. Kulyabov, and I. Pokorny, "Structural Approach to the Deep Learning Method," in *Proceedings of the 27th Symposium on Nuclear Electronics and Computing (NEC-2019)*, V. Korenkov, T. Strizh, A. Nechaevskiy, and T. Zaikina, Eds., ser. CEUR Workshop Proceedings, vol. 2507, Budva, Sep. 2019, pp. 272–275.

[3] P. Langacker, *The standard model and beyond*, ser. Series in High Energy Physics, Cosmology and Gravitation. CRC Press, 2009.

[4] I. Lakatos, "Falsification and the Methodology of Scientific Research Programmes," in *Criticism and the growth of Knowledge*, I. Lakatos and A. Musgrave, Eds., Cambr. University Press, 1970, pp. 91–195.

[5] R. Aaij *et al.*, "Search for the lepton flavour violating decay $\tau - \to \mu - \mu + \mu -$," *Journal of High Energy Physics*, vol. 2015, no. 2, p. 121, Feb. 2015. DOI: 10.1007/JHEP02(2015)121. arXiv: 1409.8548.

[6] (2018). "Flavours of Physics: Finding $\tau \to \mu\mu\mu$ (Kernels Only)," [Online]. Available: https://www.kaggle.com/c/flavours-of-physics-kernels-only.

[7] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] F. Chollet. (2020). "Keras," [Online]. Available: https://keras.io/.

[9] (2020). "XGBoost Documentation," [Online]. Available: https://xgboost.readthedocs.io.

[10] (2020). "Hep_ml," [Online]. Available: https://arogozhnikov.github.io.

[11] (2020). "CNTC official repository," [Online]. Available: https://github.com/Microsoft/cntk.

[12] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.0, 2016.

[13] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier, 2011. DOI: 10.1016/C2009-0-19715-5.

[14] A. Bruce and P. Bruce, *Practical Statistics for Data Scientists: 50 Essential Concepts*. O'Reilly Media, 2017.

[15] J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, 2016.

[16] (2020). "Scikit-learn home site," [Online]. Available: https://scikit-learn.org/stable/.

[17] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, ser. Wiley Series in Probability and Statistics. Wiley, 2013.

[18] J. M. Hilbe, *Logistic Regression Models*, ser. Chapman & Hall/CRC Texts in Statistical Science. Chapman and Hall/CRC, May 2009. DOI: 10.1201/9781420075779.

[19] D. Ruppert, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," *Journal of the American Statistical Association*, Springer Series in Statistics, vol. 99, no. 466, p. 567, 2004. DOI: 10.1198/jasa.2004.s339.

[20] R. Collins, *Machine Learning with Bagging and Boosting*. Amazon Digital Services LLC - Kdp Print Us, 2018.

[21] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. DOI: 10.2307/2699986.

[22] A. W. Kemp and B. F. J. Manly, *Randomization, Bootstrap and Monte Carlo Methods in Biology*. Ser. Chapman & Hall/CRC Texts in Statistical Science 4. CRC Press, Dec. 1997, vol. 53. DOI: 10.2307/2533527.

[23] O. Soranson, *Python Data Science Handbook: The Ultimate Guide to Learn How to Use Python for Data Analysis and Data Science. Learn the Essential Tools for Beginners to Work with Data*, ser. Artificial Intelligence Series. Amazon Digital Services LLC - KDP Print US, 2019.

[24] M. Abadi, A. Agarwal, Paul Barham, EugeneBrevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, and Jeffrey Dean. (2015). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," [Online]. Available: http://tensorflow.org/.

[25] (2020). "TensorFlow home site," [Online]. Available: https://www.tensorflow.org/.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*, 2017.

For citation:
M. N. Gevorkyan, A. V. Demidova, D. S. Kulyabov, Comparative analysis of machine learning methods by the example of the problem of determining muon decay, Discrete and Continuous Models and Applied Computational Science 28 (2) (2020) 105–119. DOI: 10.22363/2658-4670-2020-28-2-105-119.

Information about the authors:

Migran N. Gevorkyan (Russian Federation) — Candidate of Sciences in Physics and Mathematics, Assistant Professor of Department of Applied Probability and Informatics of Peoples' Friendship University of Russia (RUDN University) (e-mail: gevorkyan-mn@rudn.ru, phone: +7(495)9550927, ORCID: https://orcid.org/0000-0002-4834-4895, ResearcherID: E-9214-2016, Scopus Author ID: 57190004380)

Anastasia V. Demidova (Russian Federation) — Candidate of Sciences in Physics and Mathematics, Assistant Professor of Department of Applied Probability and Informatics of Peoples' Friendship University of Russia (RUDN University) (e-mail: demidova-av@rudn.ru, phone: +7(495)9550927, ORCID: https://orcid.org/0000-0003-1000-9650, ResearcherID: AAD-2214-2019, Scopus Author ID: 57191952809)

Dmitry S. Kulyabov (Russian Federation) — Docent, Doctor of Sciences in Physics and Mathematics, Professor at the Department of Applied Probability and Informatics of Peoples' Friendship University of Russia (RUDN University) (e-mail: kulyabov-ds@rudn.ru, phone: +7 (495) 952-02-50, ORCID: https://orcid.org/0000-0002-0877-7063, ResearcherID: I-3183-2013, Scopus Author ID: 35194130800)

УДК 519.872:519.217
DOI: 10.22363/2658-4670-2020-28-2-105-119

Сравнительный анализ методов машинного обучения на примере задачи определения мюонного распада

М. Н. Геворкян[1], **А. В. Демидова**[1], **Д. С. Кулябов**[1,2]

[1] *Кафедра прикладной информатики и теории вероятностей*
Российский университет дружбы народов
ул. Миклухо-Маклая, д. 6, Москва, 117198, Россия
[2] *Лаборатория информационных технологий*
Объединённый институт ядерных исследований
ул. Жолио-Кюри, д. 6, Дубна, Московская область, 141980, Россия

Применение алгоритмов машинного обучения для анализа статистических моделей имеет достаточно длинную историю. Развитие компьютерной техники дало этим алгоритмам новое дыхание. Особенно громкую известность получило такое направление машинного обучения, как глубинное обучение. Однако авторы полагают, что многие исследователи пытаются использовать методы глубинного обучения за пределами их применимости. Этому способствуют как широкая распространённость программных комплексов, реализующих алгоритмы глубинного обучения, так и кажущаяся простота исследования. Всё это стало побудительным мотивом для проведения сравнения алгоритмов глубинного обучения и классических алгоритмов машинного обучения.

В качестве задачи был выбран эксперимент на Большом адронном коллайдере, поскольку авторы знакомы с данной научной областью, а также потому, что данные эксперимента доступны публично. В статье проводится сравнение различных алгоритмов машинного обучения применительно к задаче распознания реакции распада $\tau^- \to \mu^- + \mu^- + \mu^+$ на Большом адронном коллайдере. Используются готовые свободные реализации алгоритмов машинного обучения. Алгоритмы сравниваются друг с другом на основе вычисляемых метрик. В результате исследования можно сделать вывод, что все рассмотренные методы машинного обучения вполне сопоставимы друг с другом (с учётом выбранных метрик), при этом разные методы имеют разные области применимости.

Ключевые слова: мюонный распад, машинное обучение, нейронные сети