Machine Learning Approaches to Choose Heroes in Dota 2

Iuliia Porokhnenko, Petr Polezhaev, Alexander Shukhman
Orenburg State University
Orenburg, Russian Federation
yulkins2@gmail.com, polezhaev.ds@gmail.com, shukhman@gmail.com

***Abstract*—The winning in the multiplayer online game Dota 2 for teams is a sum of many factors. One of the most significant of them is the right choice of heroes for the team. It is possible to predict a match result based on the chosen heroes for both teams. This paper considers different approaches to predicting results of a match using machine learning methods to solve the classification problem. The experimental comparison of predictive classification models was done, including the optimization of their hyperparameters. It showed that the best classification models are linear regression, linear support vector machine, as well as neural network with Softplus and Sigmoid activation functions. The fastest of them is the linear regression model, so it is best suited for practical implementation.**

I. Introduction

Computer games have become an important social, cultural and economic factor. Multiplayer online games currently attract a large number of players and have a wide audience of observers. In addition, there is a growing interest in games related to eSports.

ESports includes team or individual competitions based on computer games. All eSports disciplines are divided into several main classes, which differ in the properties of spaces, models, game problem and developed skills of cyber sportsmen. Multiplayer Online Battle Arena (MOBA) games are among the most popular in eSports.

Prediction about the results of matches in sports games has always been a popular topic in the machine learning area. Sports analytics is often used to make decisions in professional kinds of sport. Therefore, it can be assumed that such systems will also be useful for users of multiplayer online games.

MOBA is a genre of computer games that combines the real-time strategy and computer role-playing. One of the most popular games in the MOBA genre is Dota 2 developed by Valve Corporation. Nearly 500 thousand players play it every day. So, it was selected as a subject of the current research.

In Dota2, two teams of players fight on a certain type of maps in real time. The map is a combination of three lines (top, middle and bottom) and the area between them (jungles). Each player controls one hero, which he chooses from a list of heroes. Heroes differ from each other in various features and characteristics. During the match, heroes can become stronger, develop new skills, enhance their characteristics and acquire objects. The goal of the game is the destruction of the main building belonged to the enemy's team by player controlled heroes and creeps controlled by artificial intelligence.

Section 2 describes the existing approaches to solving the problem of increasing the efficiency of a team in the multiplayer online game DOTA 2. Formulation of machine learning problem for DOTA 2 and preparation of the training dataset are considered in Section 3. Section 4 presents the classifying models chosen from well-known frameworks, the classifiers are used to solve the formulated problem. Comparative study results of the classifying models trained on the prepared dataset, the optimization of their hyperparameters, as well as the results of neural network study (on CPU, GPU and TPU) and its hyperparameter optimization, are described in Section 5.

II. Related work

There are many different approaches to solving the problem of increasing the efficiency of a team in the multiplayer online game DOTA 2.

The success of the team is influenced by many different factors. They can be divided conditionally into two groups: the personal contribution of each player and the principle of team building.

The first group includes such characteristics as the player's experience, the sequence of his actions in the game, made decisions, etc. Paper [1] considers player models and methods, which can get results closest to the behavior of a real player. Player models take into account the choice of hero skills depending on the game situation and the sequence of such choices to increase their skills. The choice of items for purchasing by hero was also considered as part of studying the influence of the hero's personal contribution to the success of the whole team [2].

More attention is paid to the principle of team building. In [3] Nataliia Pobiedina and others prove that a correct choice of heroes by player is the factor, which has the most significant effect on the success of a team in the game.

There are several approaches to choosing the heroes for the match. Existing services, such as DotaBuff [4] and DotaPicker [5], use analysis of statistic information obtained from played matches. These statistics are updated sometime after each

release of the new update in the game to actualize available information.

Machine learning is another approach, which is used for the selection of heroes in the game.

In [6] Filip Beskyd considers a decision tree and a neural network to predict the outcome of the game based on chosen heroes.

Zhengxing Chen and others [7] compared Monte-Carlo method, logistic regression and gradient boosting to predict the outcome of a match.

It worth nothing that all authors, who use the machine learning approach in their research, gain accuracy ranged from 50% to 70%. Moreover, the use of the same machine learning models by different researchers gives different results. This happens because these models are applied to different datasets or they are configured by different hyperparameters or structures (for neural networks).

At present, there are no studies, which can help to choose the most efficient set of hyperparameters that affect the accuracy of predicting game outcome.

III. Formulation of Machine Learning Problem and Dataset Preparation

The problem of our research is to build an effective algorithm for predicting match results based on information about the DOTA 2 heroes chosen by players. This problem is a binary classification problem with two output classes meaning victory for the “radiant” or “dire” team.

The OpenDota API allows developers to get data about Dota 2 matches. The information about 56,690 matches played in 2018 was obtained through the OpenDota API. This information corresponds to a number of requirements.

First, game modes must be such that each hero has a non-zero probability of appearing in the game. Such modes are “all pick” (each player chooses one from all the heroes), “single draft” (each player chooses one from three random heroes), “all random” (each player gets a random hero), “random draft” (players take turns choosing heroes from a pool of 50 random heroes), “captain’s draft” (each team is assigned a captain who chooses heroes from the list of random heroes), “captain’s mode” (each team is assigned a captain who chooses heroes for his team and prohibits ones for opponents).

Secondly, the skill level of the players in the match must be high. Such a requirement is necessary, so only those matches are considered to learn our model, in which heroes are selected based on any strategic considerations.

Thirdly, only those matches are taken into account in which players do not leave the match until the end. This is necessary so that the math result depends only on skills and choice of the team's heroes, and doesn’t depend on the balance and the number of players.

Based on downloaded data, a training set of labeled data was created consisting of pairs of vectors (x, y). Here x is a vector of size 216, which contains information about picks (pick is the choice of a hero), the first half of which is for the “radiant” team, the second half for the “dire” team:

$$x_i = \begin{cases} 1, if\ player\ of\ the\ "radiant"team \\ had\ played\ with\ the\ hero\ id = i, \\ 0,\ otherwise, \end{cases}$$

$$x_{108+i} = \begin{cases} 1, \quad if\ player\ of\ the\ "dire"team \\ had\ played\ with\ the\ hero\ id = i, \\ 0,\ otherwise. \end{cases}$$

There are 108 different heroes, and each of them can be selected once by both teams. Vector y contains information about the match result:

$$y = \begin{cases} 1, if\ "radiant"team\ had\ won, \\ 0,\ overwise. \end{cases}$$

For further research, it was necessary to compare various classification methods and select the optimal combination of their hyperparameters.

IV. Classification Models for Solving the Problem

Gradient Boosting Classification (GBC) is a machine learning method for solving regression and classification problems that creates a prediction model as a linear combination of basic classifiers, which minimize the differentiated loss function.

Random Forest Classification (RFC) is a machine learning algorithm that uses a set of decision trees, which reduces retraining problems and improves accuracy in comparison with a single tree. The result is obtained by aggregating the responses of multiple decision trees.

XGBoost Classification (XGBC) uses pre-sorted algorithm and Histogram-based algorithm for computing the best split [8].

Logistic regression (LR) is a method for constructing a linear classifier to estimate a posteriori probabilities of belonging objects to classes. It is a statistical model used to predict the probability of an event occurring by fitting data to a logistic curve. LR is a very powerful algorithm, especially for high-dimensional problems. It is actively used in Kaggle competitions along with tree boosting approaches.

Linear support vector classification (LSVC) is an algorithm for solving classification problems using only the linear core. Compared to the SVM algorithm, Linear SVC learns faster and scales better.

CatBoost Classification (CBC) is a machine learning algorithm using gradient boosting on decision trees, it available in the CatBoost library from Yandex. It is a follower of the MatrixNet algorithm, which is used for ranking and forecasting. Also, it is the base for recommender technologies.

The implementations of classification methods for training and testing were used from the Scikit-learn, XGBoost and CatBoost libraries. Keras with Tensorflow were used to implement the neural network.

We chose these libraries due to their popularity, high level of optimization, availability of good documentation, examples, as well as a large community of users.

Python was used to develop a Jupyter notebook for training and evaluation of abovementioned algorithms.

V. Experimental study of models

Training and testing of models were carried out with Google Colaboratory. It is a cloud service to access a remote machine with machine learning software. The virtual environment provided by Google has the following characteristics: Intel (R) Xeon (R) CPU with a clock frequency of 2.20GHz and 1 core, 13GB of RAM, 33GB of free memory. All necessary machine-learning libraries (except CatBoost) and their dependencies are included to it. CatBoost was installed with pip command line tool for Python.

All data is divided into training and test sets. The test set is 10% of the total data, i.e. 5670 matches.

Evaluation of the efficiency of classification models was carried out using the k-fold cross-validation method (5 blocks). For each classification algorithm, the parameters were optimized using the GridSearchCV algorithm, which performs a complete enumeration by a manually defined subset of the space of hyperparameters of the training algorithm. Table I for each classification model presents the hyperparameters and their enumerated values. Table II shows the best combinations of hyperparameters for each model.

TABLE I. Enumerated values of hyperparameters of classification models

Model	Hyperparameter	Values
LR	penalty	l1, l2
	C	0.001, 0.01, 0.1, 1.0, 10.0
	solver	lfbgs
LSVC	C	1, 10, 100
	Gamma	0.001, 0.01, 0.1, 1.0, 10.0
GBC	loss	deviance
	max_features	sqrt, log2
	criterion	mae, friedman_mse
	n_estimators	10
RFC	n_estimators	200, 500
	max_features	log2, auto, sqrt
	max_depth	10, 25, 50
	criterion	gini, entropy
XGBC	colsample_bytree	0.6, 0.8, 1.0
	learning_rate	0.05, 0.1
	silent	1
	nthread	2, 4, 8
	min_child_weight	1, 5, 10
	n_estimators	5
	subsample	0.6, 0.8, 1.0
	max_depth	3, 4, 5
	objective	binary:logistic
CBC	iterations	50, 100
	depth	4, 6, 8
	loss_function	Logloss, CrossEntropy
	verbose	True
	learning_rate	0.01, 0.03, 0.1
	l2_leaf_reg	-4, 0, 4

TABLE II. Optimal values of hyperparameters of classification models

Model	Hyperparameter	Hyperparameter meaning	Optimal value
LR	penalty	Used to specify the norm used in the penalization	l1
	C	Inverse of regularization strength	1.0
	solver	Algorithm to use in the optimization problem	lfbgs
LSVC	C	Penalty parameter C of the error term	10
	Gamma	Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’	1
GBC	loss	Loss function to be optimized	deviance
	max_features	The number of features to consider when looking for the best split	sqrt
	criterion	The function to measure the quality of a split	mae
	n_estimators	The number of estimators as selected by early stopping	10
RFC	n_estimators	The number of trees in the forest	500
	max_features	The number of features to consider when looking for the best split	log2
	max_depth	The maximum depth of the tree	50
	criterion	The function to measure the quality of a split	entropy
XGBC	colsample_bytree	The subsample ratio of columns when constructing each tree	1.0
	learning_rate	Step size shrinkage used in update to prevents overfitting	0.05
	silent	Verbosity of printing messages	1
	nthread	Number of parallel threads used to run XGBoost	4
	min_child_weight	Minimum sum of instance weight (hessian) needed in a child	10
	n_estimators	The number of trees in the forest	5
	subsample	Subsample ratio of the training instances	1.0
	max_depth	Maximum depth of a tree	5
	objective	Learning task parameter	binary:logistic
CBC	iterations	The metric to use in training	100
	depth	The maximum number of trees that can be built when solving machine learning problems	6
	loss_function	The metric to use in training	Logloss
	verbose	Defines the logging level	True
	learning_rate	The learning rate	0.1
	l2_leaf_reg	L2 regularization coefficient	4

Table III presents the values of the efficiency (for optimal hyperparameters) and the learning time for each classification model.

The ROC AUC (Receiver Operating Characteristic Area Under Curve), which represents the area bounded by the ROC curve and the axis of the fraction of false positive classifications, was used as a metric for assessing the efficiency of classifiers.

This metric is used because the data is not balanced, that is, the number of instances in each of the classes is not the same. For such data, it is better to use the ROC AUC metric, rather than accuracy, since it is based on True Positive Rate and False Positive Rate.

TABLE III. EVALUATION OF THE EFFICIENCY OF CLASSIFICATION MODELS

Model	ROC AUC	Time, seconds
LR	**0.7739**	**0.20**
LSVC	**0.7739**	**1.53**
GBC	0.7409	16.01
RFC	0.7002	1.36
XGBC	0.7402	18.61
CBC	0.7673	12.85

The analysis of Table III shows that LR and LSVC algorithms provide the greatest efficiency of classification, CBC – slightly less value of AUC. The fastest method is the LR algorithm. The remaining algorithms have AUC value close to 0.7.

Fig.1 and Fig. 2 show the normalized confusion matrices for LR and LSVC algorithm. Both models are difficult to classify the "radiant" team due to imbalanced dataset.

Fig. 3 shows the ROC curves that allow evaluating the quality of the classification. They display the relationship between True Positive Rate and False Positive Rate. According to the results, the best classifiers are LR, LSVC and CBC.

A neural network also can be used to solve the classification problem. Fig. 4 shows the scheme of the neural network obtained by the TensorBoard visualizer.

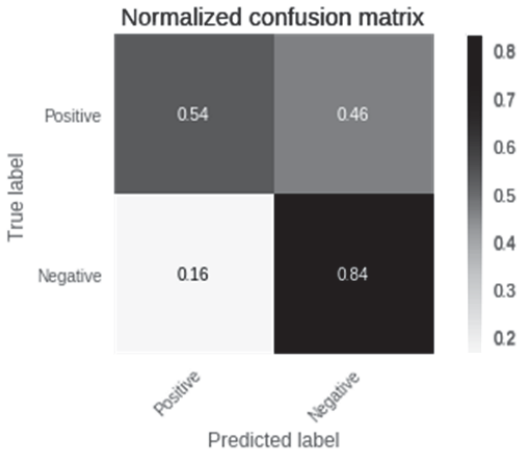

Fig. 1. Normalized confusion matrix for LR algorithm

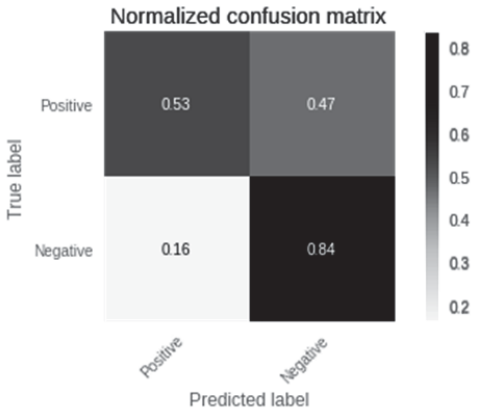

Fig. 2. Normalized confusion matrix for LSVC algorithm

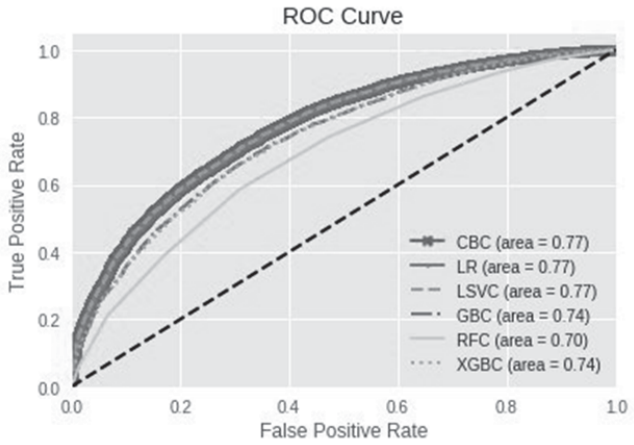

Fig. 3. ROC curves of classification models

Code Fragment 1 shows the code for the Keras framework which describes the creation of this network. Used hyperparameters are presented in Table 4. The network input layer is a binary vector of 216 values, which is then divided into two parts x_radiant and x_dire using Keras Lambda layers, then the Dense layers hero_layer_1, hero_layer_2, hero_layer_1_1, hero_layer_2_1 are applied to them, which are necessary to eliminate the influence of the heroes on the "dire" or "radiant" team on the match result, the resulting output vectors are concatenated. Then there are three more hidden Dense layers together with dropout regularization layers to get a solution. The output layer of the neural network is the Dense layer with 1 neuron and the Sigmoid activation function. This network architecture was considered by Mark Dunne in [9].

Code Fragment 1 Neural network for Keras framework

```
# Input layer, which represents a vector of 216 binary values
input = Input(shape=(216,))
# Lambda layer, which extracts the first 108 values of input as
x_radiant (choice of "radiant" team)
x_radiant = Lambda(lambda x: x[:, :108])(input)
# Lambda layer, which extracts the next 108 values of input
as x_dire (choice of "dire" team)
x_dire = Lambda(lambda x: x[:, 108:])(input)
# Dense layer of NUM_UNITS_IN_FIRST_LAYERS
```

```
neurons
hero_layers_1 = Dense (NUM_UNITS_IN_FIRST_ LAYERS, activation='relu')
# Apply it to x_dire, the output is dire_layer1
dire_layer1 = hero_layers_1(x_dire)
# Apply it to x_radiant, the output is radiant_layer1
radiant_layer1 = hero_layers_1(x_radiant)
# Dense layer of NUM_UNITS_IN_FIRST_LAYERS neurons
hero_layer_2 = Dense (NUM_UNITS_IN_FIRST_ LAYERS, activation='relu')
# Apply it to dire_layer1, the output is dire_layer2
dire_layer2 = hero_layer_2(dire_layer1)
# Apply it to radiant_layer1, the output is radiant_layer2
radiant_layer2 = hero_layer_2(radiant_layer1)
# Concatenate dire_layer2 and radiant_layer2
conc = concatenate([dire_layer2, radiant_layer2])
# Apply dropout regularizations
dropout1 = Dropout(DROPOUT1)(conc)
dropout2 = Dropout(DROPOUT2)(dropout1)
# Dense layer of NUM_UNITS_HIDDEN1 neurons
hidden1 = Dense(NUM_UNITS_HIDDEN1, activation='relu')(dropout2)
# Dropout regularization
drop_hidden1 = Dropout(DROPOUT2)(hidden1)
# Dense layer of NUM_UNITS_HIDDEN2 neurons
hidden2 = Dense (NUM_UNITS_HIDDEN2, activation='relu')(drop_hidden1)
# Dropout regularization
drop_hidden2 = Dropout(DROPOUT2)(hidden2)
# Dense layer of NUM_UNITS_HIDDEN3 neurons
hidden3 = Dense (NUM_UNITS_HIDDEN3, activation='relu')(drop_hidden2)
# Output layer
output = Dense(1, activation='sigmoid')(hidden3)
# Final Keras model
model = Model(inputs=input, outputs=output)
```

The hyperparameters of the neural network were optimized by the GridSearchCV algorithm. Table IV contains the meaning and possible values of each parameter. The optimal combination of hyperparameters is shown in Table V.

TABLE IV. ENUMERATED VALUES OF HYPERPARAMETERS OF NEURAL NETWORK

Hyperparameter	Hyperparameter meaning	Value
DROPOUT1	Percentage of dropped values of neurons outputs	0.3, 0.5
DROPOUT2		0.3, 0.5
BATCH_SIZE	Number of training samples in the batch	512, 1024
NUM_EPOCHS	Number of epochs	50, 100, 150
NUM_UNITS_IN_FIRST _LAYER	Number of neurons in the first and second hidden layers	50, 80, 100
NUM_UNITS_HIDDEN1	Number of neurons in the hidden layer that is the first after concatenation layer	150, 120, 100
NUM_UNITS_HIDDEN2	Number of neurons in the hidden layer that is the second after concatenation layer	100, 75, 50
NUM_UNITS_HIDDEN3	Number of neurons in the hidden layer that is the third after concatenation layer	50, 25, 10

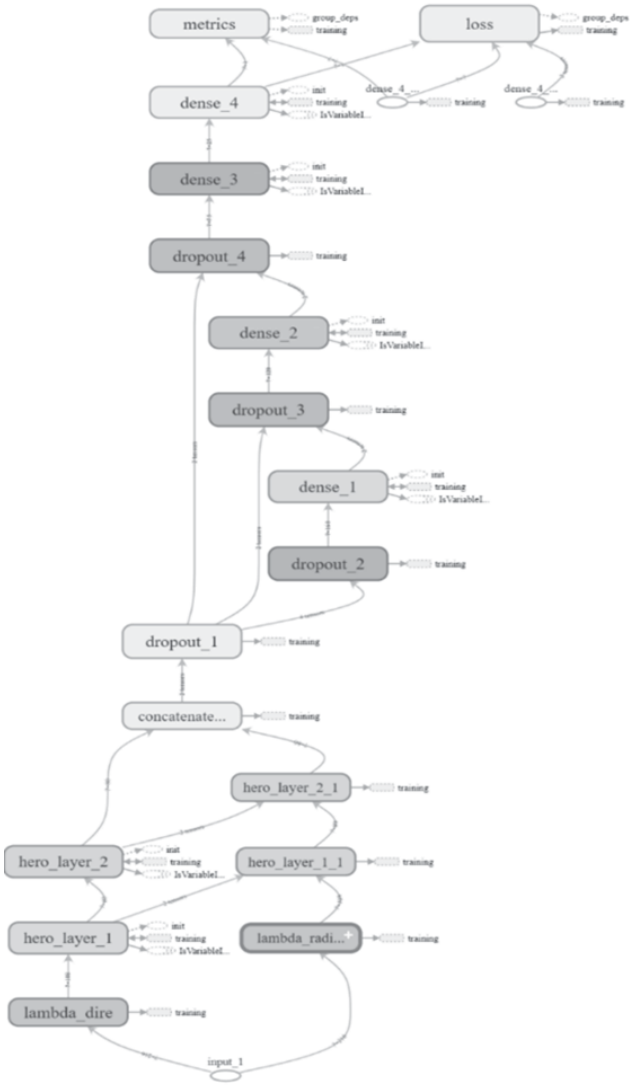

Fig. 4. Neural network scheme

TABLE V. THE OPTIMAL COMBINATION OF THE NEURAL NETWORK HYPERPARAMETERS

11	Value
DROPOUT1	0.5
DROPOUT2	0.5
BATCH_SIZE	512
NUM_EPOCHS	100
NUM_UNITS_IN_FIRST_LAYER	80
NUM_UNITS_HIDDEN1	100
NUM_UNITS_HIDDEN2	50
NUM_UNITS_HIDDEN3	25

The neural network training was conducted using the Google Collaboratory. This service provides the opportunity to train models on GPU Tesla K80 with 13 Gb of video memory and on TPU – Tensor Processing Unit from Google, which is intended for a larger volume of computations with reduced precision. TPU uses XLA (Accelerated Linear Algebra) – compiler that optimizes TensorFlow computations.

Table VI shows the results of neural network training on CPU, GPU and TPU for optimal combination of hyperparameters.

TABLE VI. NEURAL NETWORK TRAINING ON CPU, GPU AND TPU

Hardware accelerator	Time, seconds	ROC AUC
CPU	828.52	0.7711
GPU	547.32	0.7705
TPU	**304.92**	**0.7716**

We can conclude from obtained results that the fastest way of training the neural network is using TPU, which is nearly 1.8 times faster than GPU Tesla K80.

Table VII presents the AUC values for the neural network depending on the number of hidden layers after the concatenation layer in its structure. Training of the neural network was performed taking into account the optimal combination of hyperparameters.

TABLE VII. TRAINING OF THE NEURAL NETWORK FOR A DIFFERENT NUMBER OF HIDDEN LAYERS

Number of hidden layers	Time, s	ROC AUC
1	56134.62	0.7273
2	56285.84	0.7493
3	**547.32**	**0.7705**
4	698.84	0.7630

Analysis of the results shows that the most optimal in terms of training time and efficiency are three hidden layers in the structure of the neural network.

In addition, different activation functions were considered (see Table VIII). Activation functions Softplus, Softsign, ReLU, Tanh and Sigmoid show good values of ROC AUC metric. Tanh and Sigmoid demonstrate the best values. The least efficient is the Exponential function.

Thus, the best implementation of neural network should use the optimal hyperparameters from Table V, Softplus or Sigmoid activation function and contain three hidden layers after the concatenation layer in its structure. For the best performance, it can be trained on TPU.

VI. CONCLUSION

Analysis of the classification models shows that the maximum value of ROC AUC metric that can be archived with hyperparameter optimization is 0.77. In this case, the best models are Linear Regression, Linear SVC and neural network with the activation functions Softplus and Sigmoid. The fastest of them is the linear regression model, so it is best suited for practical implementation.

TABLE VII. COMPARISON OF ACTIVATION FUNCTIONS

ACTIVATION FUNCTION	ROC AUC
SOFTMAX	0.7686
ELU	0.7696
SELU	0.7697
SOFTPLUS	**0.7740**
SOFTSIGN	0.7704
RELU	0.7715
TANH	0.7702
SIGMOID	**0.7740**
EXPONENTIAL	0.6197

In the future, we plan to study other factors potentially affecting the outcome of a match, such as player experience, characteristics of heroes and their synergy within a team. They can improve the efficiency of classification models and provide more accurate information about the match outcome.

In this case, it will be necessary to use different feature engineering techniques, which are not applicable to the classification problem in its current formulation.

ACKNOWLEDGMENT

The research was supported by the Russian Foundation for Basic Research in the framework of the scientific project No. 18-07-01446 and by the Foundation for Assistance to Small Innovative Enterprises No. 12794GU/2018 from 04/26/2018.

REFERENCES

[1] J. Pfau, J. D. Smeddinck, & R. Malaka, "Towards Deep Player Behavior Models in MMORPGs", *in Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play,* Oct. 2018, pp. 381-392.

[2] W. Looi, M. Dhaliwal, R. Alhajj, & J. Rokne, "Recommender System for Items in Dota 2", *IEEE Transactions on Games*, 2018.

[3] N. Pobiedina, J. Neidhardt, M. D. C. C. Moreno, L. Grad-Gyenge, & H. Werthner, "On successful team formation: Statistical analysis of a multiplayer online game", *in Business Informatics (CBI) 2013 IEEE 15th Conference*, July 2013, pp. 55-62.

[4] DotaBuff official website, DotaBuff, Web: https://ru.dotabuff.com/

[5] DotaPicker official website, DotaPicker, Web: http://dotapicker.com/

[6] F. Beskyd, "Predicting the Dota 2 Game Results", *in Czech Technical University in Prague, Calculation and Information Center,* 2018.

[7] Z. Chen, T. H. D. Nguyen, Y. Xu, C. Amato, S. Cooper, Y. Sun, & M. S. El-Nasr, "The art of drafting: a team-oriented hero recommendation system for multiplayer online battle arena games", *in Proceedings of the 12th ACM Conference on Recommender Systems,* Sept. 2018, pp. 200-208.

[8] Towards Data Science, CatBoost vs. Light GBM vs. XGBoost, Web: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

[9] Github, Tensorflow Dota Predictor, Web: http://markdunne.github.io/2016/02/07/TensorFlow-Dota-Model