Analyzing a Python Programming Example of Building an Artificial Intelligence Machine Learning Model for Detecting Cyber-Attacks – Benefits and Challenges

MLADEN U. RADAKOVIĆ, Information Technology School - ITS, Belgrade
MARINA D. MARJANOVIĆ, University Singidunum, Belgrade

Professional paper
UDC: 007:004.056.5
004.852
DOI: 10.5937/tehnika2302187R

Artificial intelligence systems offer a wide range of real-world applications that are already being used in many different fields. They can be found in computer systems, web browsers, smart home appliances, navigation and vehicle management, shape, and sound identification, in the medical, economic, agricultural fields… The effectiveness of their application, the outcomes, and the time required to create such models vary, and trends point their designers toward techniques that allow the construction of quicker and more accurate solutions. Through the examination of one example of the development of an artificial intelligence model using the Python programming language, this study attempts to demonstrate the advantages and difficulties of its implementation. The possibilities and method of creating a functional model of artificial intelligence are demonstrated through the presentation of its design and code examples for a realistic set of data. Used dataset contains information needed for analysis and detection of computer network cyber-attacks.

Key Words*: machine learning, artificial intelligence, python, cyber security, computer network, intrusion detection*

1. INTRODUCTION

The development of artificial intelligence (AI) technology and machine learning (ML) [1] began shortly after the Second World War, and today its application is present in many scientific, business, and other fields. By using models and systems based on this technology, it is possible to teach machines to work, to behave and think like people, and even to imitate some of their actions [2].

The development of such AI systems is done using various specialized programs, programming languages and machines. One of the most used tools for this purpose is the Python programming language. With many developed specialized modules, based on this widely implemented programming language, programmers can use ready-made classes and functions [3]. This significantly facilitates and speeds up the creation of the required, appropriate solution.

This paper is organized into several parts that show the creation process and analyze an example of creating an ML model that detects cyber-attacks, using the Python programming language. After the introductory part, in the second part of the work, the method of creating an ML model will be presented in detail. With reference to the entire procedure, data preparation, distribution into sets for model training and testing, structure creation, compiling, training and evaluation of model performance will be analyzed. The third section of the paper will present the results obtained by experimenting on various settings of the model.

These outcomes show the differences obtained by changing the design parameters and the structure of the created system. Special attention in the paper will be devoted to the practical creation of the ML model itself in the Python programming language.

The success of its functioning is most influenced by the selection of its internal structure, parameters and functions that essentially describe and make up the model. In the fourth part of the paper, the influence of the parameters on the results and their interpretation will be discussed.

The model's success in achieving its main task - detecting unauthorized intrusions into computer networks (detection of cyber-attacks) - will be presented and analyzed. Finally, the possibilities and limitations of using such systems will be discussed, taking into

Author's address: Mladen Radaković, Information Technology School - ITS, Belgrade, Savski nasip 7
e-mail: mladen.radakovic@gmail.com
Paper received: 31.01.2023.
Paper accepted: 13.04.2023.

account future practical solutions and academic research.

2. A METHOD FOR CREATING MACHINE LEARNING MODEL USING PYTHON

Artificial Intelligence and Machine Learning models can predict results or make decisions based on mathematical models created earlier. These models are being created based on previously processed and evaluated data. Their main goal is to help and replace humans in making decisions. The theory of AI is based on the modeling of machine intelligence using hardware and software. They try to mimic functioning of human brain and with the use of mathematical functions that act as artificial neurons and their networks [4].

Algorithms for ML can be divided into three major groups: supervised, unsupervised, and reinforced learning models [5]. Supervised depend on the data that has been evaluated and graded previously by a person. Unsupervised is based on large datasets with ML models providing analytical observations without human input or action. Reinforced is using manual input as a corrective function that improves parameters of mathematical functions that create ML model.

Example presented in this paper is demonstrating creation of supervised ML model based on previously classified data in Python using its open-source modules. Many specialized Python modules [6] support and enable the implementation of systems and models for machine learning. Some of the most important steps in ML model creation and the libraries used in this paper are presented on Figure 1 (with their respective module names).

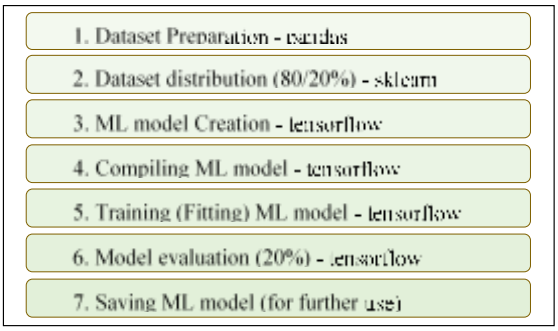

Figure 1 - Supervised Machine Learning Model Implementation Algorithm using Python

The development of a ML model for detecting network cyber-attacks usually involves several key steps, including data collection, feature extraction, model selection, and model evaluation. Dataset used typically includes network traffic records, system logs and other relevant data. After extraction of relevant features from data, selection of an appropriate machine learning model is the most important step. For detecting network cyber-attacks, commonly used models are including decision trees, random forests, deep learning models, support vector machines, hybrid models, etc. Model evaluation is a final step that evaluates performance of created machine learning model [7].

2.1. DATASET PREPARATION FOR SUPERVISED ML MODELS

Data should be collected, cleaned, integrated, and processed before being used to create ML models. It should be sufficiently large to receive meaningful and accurate results from the model. Collecting data might involve various sources (web, databases, files, …). Cleaning the data assumes correcting errors in the set, adding missing values, removing excessive data, duplicates, or irrelevant data. If data is acquired from multiple sources, it should be merged, consolidated, and sorted in useable format.

ML models are typically created as tools for working with real-world, empirical data, but they can also be trained and learned using ready-made, prepared data that is widely available online. The following are the largest and most broadly used online databases: Google Dataset Search, Kaggle, World Bank Open Data [8], AI & ML at Cisco DevNet Learning Labs, World Health Organization, Cern Open Data, Earth Data, UCI Machine Learning Repository, Data Hub, OpenML, etc.

This example's data set is in a table containing information about computer network intrusion detection. Set represents a collection of 41 features extracted from TCP/IP traffic analyzed from a source to a destination IP address. Each record (table row) has been labeled as normal or anomalous connection. Using data cleaning and integration standards and procedures, raw data has been cleaned, processed, and converted into a format that can be worked with. Data from the internet website Kaggle.com was used to test the model. Table 1 displays a sample of data (header and footer) that will be used to build the ML model.

Table 1. The structure of the dataset table used to build the machine learning model

duration	src_bytes	dst_bytes	land	urgent	:	root_shell	count	srv_count	serror_rate	srv_serror_rate	rerror_rate	srv_rerror_rate	same_srv_rate	diff_srv_rate	dst_host_count	class (alert=1)
0	491	0	0	0	...	0	2	2	0	0	0	0	1	0	150	0
0	146	0	0	0	...	0	13	1	0	0	0	0	0.08	0.15	255	0
0	0	0	0	0	...	0	123	6	1	1	0	0	0.05	0.07	255	1
0	232	8153	0	0	...	0	5	5	0.2	0.2	0	0	1	0	30	0
0	199	420	0	0	...	0	30	32	0	0	0	0	1	0	255	0
0	0	0	0	0	...	0	121	19	0	0	1	1	0.16	0.06	255	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0	300	13788	0	0	...	0	8	9	0	0.1	0	0	1	0	91	0
0	0	0	0	0	0	0	38	9	1	1	0	0	0.24	0.11	255	1

The complete example database is available on the Kaggle website [9]. For the algorithms to be able to work with the original table, provided data had to be validated and checked for missing values, and other errors. Data curing, filtering, and excluding extreme values from the dataset help with getting representative dataset. Some columns with text values had to be removed or converted to numerical values. The result of the manual classification is displayed in the rightmost column („class“), indicating whether the network behavior was considered malicious (1) or not (0).

The presented data is divided into two subsets, which are used to train the future ML model:

1. Without the rightmost column: „data_no_results“

2. Data containing only the result that corresponds to the first dataset: "data_only_results"

These two datasets, that were being divided by the method displayed in Figure 2, are then used as input data for the ML model's training function.

After the initial stage of data preparation has been completed successfully, the distribution of the data into sets for model training (fit) and testing (evaluate) comes next. The remainder of the essay discusses this.

```
Import pandas as pd  # module for work w/data like CSV files
dataset_file = \
"Network_Intrusion_Detection_Train_Dataset.csv"
data_with_results = pd.read_csv(dataset_file)
# One dataset data wo/results and other w/only results
data_only_results = data_with_results["class"].copy()
data_no_results = data_with_results.drop(columns=["class"])
```

Figure 2 - Preparation and distribution of data for further work

2.2. Distribution of datasets into subsets for training and testing

The collected and pre-processed data should be divided into two groups - one for training the model and one for testing it. The division of data into sets for training and testing is used in supervised systems, to confirm the effectiveness of the formed model's successful implementation [10]. The principle of dividing data into 80% for training and 20% for testing may vary depending on the number and type of data. Based on the supplied parameter „test_size“, the function presented in Figure 3 automatically separates the data into two logical subsets. The value 0.2 in the illustrative example represents the 20% of the data that the function will retain for later testing the ML model.

In the example presented, dataset for model training contains 25.192 rows of data in 42 columns.

```
# 2. Data distribution (20% for model testing)
From sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = \
  train_test_split(data_no_results, data_only_results, \
  test_size=0.2, random_state=0)
```

Figure 3 - Dataset distribution into four subsets

2.2.1. Data subset for training ml model (fit)

The database's most significant and substantial portion of 80% in the example given is used to fit and train the model. Depending on the amount and type of data, we allocate a different percentage for model training. The model could be better prepared for use and prediction in the future with a larger dataset sample size. The more diverse the data set, the better the neural network architecture „learns“ and adapts its internal connections and weights for future use.

2.2.2. Data subset for testing created ml model (evaluate)

By inserting a set of data for testing (20%) into an already created model and comparing the results of the model with real, categorized results, it can be concluded whether a correction is needed in the design of the model itself. If the evaluation reveals that it does not produce satisfactory results, the model structure should be adjusted and tested again.

2.3. Creating a machine learning model

One of the most important steps when designing a ML model is the correct choice of structure and the appropriate setting of the parameters that will make up the future algorithm. These parameters determine the fundamental parts of the produced model. The configuration chosen directly affects how well the model works and how accurate it is.

If after creating and training the model the results are not satisfactory, it is necessary to correct all or some of the parameters that describe it (model shape and type, number of neural network layers, number of output nodes of each layer, activation function, error function, ...). Figure 4 demonstrates the actual process of building the ML model's sequential structure.

```
# 3. Creating a ML model structure
Import tensorflow as tf
model = tf.keras.models.Sequential()
# A Sequential model is suitable for plain stack of layers
model.add(tf.keras.layers.Dense(16,
                                input_shape=x_train.shape[1:],
                                activation="sigmoid"))
model.add(tf.keras.layers.Dense(16, activation="sigmoid"))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))
```

Figure 4 - Building layers of neural networks for the sequential model

Neural networks, as in Figure 5, are created from artificial neurons that receive and process input data.

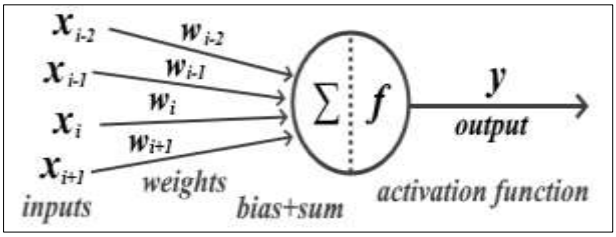

Figure 5 - Concept of artificial neuron [12]

Data is passed through the input layer, passes through the hidden layer (which can be composed of several layers) and exits through the output layer of the neuron [11]. In the example processed in this paper, in the program code, only three layers were created: one input (16), one hidden (16) and one output (1).

Only the input layer needs to define and pass the shape of the data that will be forwarded to the algorithm („input_shape“). The number 16 (for the input layer) represents the number of „output“ neurons that are passed to the next layer - the only hidden layer in the example shown in this paper. The hidden layer forwards to the output also 16 neurons, while the output layer gives only one (output) result. For research purposes, only one result is required and expected, due to the form and type of data being processed.

In the analyzed example, the „sigmoid“ function is used, which is shown in Figure 6, and which best corresponds to the data used. Due to the choice of this activation function and only one neuron, the result of the algorithm will be exactly one (discrete) value between the values 0 and 1. Additionally to the activation function used in this work, sigmoid, tangent-hyperbolic (tanh), and so-called rectification linear functions (ReLU) are commonly utilized. Figure 6 displays their appearance [13].

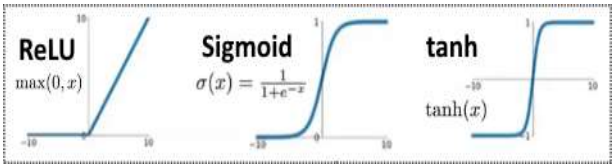

Figure 6 - Commonly utilized Activation functions

2.4. Compiling of ml model

Compiling is the process of translating high-level human comprehensible computer code into low-level executable instructions that run directly on a machine (computer). When compiling this model, an optimizer „Adam“ (Figure 7) was selected [14]. This practically determines the way and methodology in which the weighting coefficients (parameters) that affect the training of the model will be valorized. Compiling the ML model is most influenced by the “optimizer”, „loss“ and „metrics“, which will be explained in more detail in the rest of this part of the paper.

```
# 4. Compiling the model
model.compile(optimizer="Adam",
              loss="mse",metrics=["accuracy"])
```

Figure 7 - Compiling the model

2.4.1. Model optimization function - „optimizer“

The choice of the „optimizer“ algorithm that will be used to optimize the ML model directly affects the results that the model will generate. The functions that can be selected to optimize the operation of the model are the following: Adam, Adadelta, Adagrad, RMS-prop, Adamax, Ftrl, Nadam, Optimizer class, SGD. The 'Adam' optimizer was used in this paper's example.

2.4.2. Error calculation function - „loss“

The „loss“ error calculation function is one of the bases for analyzing and optimizing model performance. It is a crucial component of every ML model as it is measuring the difference between predicted output and the actual model output. It can have a significant impact on the model performance. It is chosen based on the nature of the data and the problem specifics being addressed. There is no best ‘loss’ function that fits all scenarios because it is dependent on a variety of factors. The type of data, nature of the problem, data distribution, and metric performance all have a direct impact on function selection [15]. Commonly used „loss“ functions in machine learning are based on calculation of mean squared error, mean absolute error, binary cross-entropy loss, categorical cross-entropy, hinge loss, Huber loss, etc.

In this example, for this purpose of this case, taking in consideration all the example parameters, the algorithm that calculates the standard deviation („mse“ - mean squared error) was selected. This function is also known as the „cost“ function, which considers the probability of the uncertainty of the obtained forecast (result). The „loss“ function estimates how much the real results deviate from the results obtained by the ML model. An ideal model result, with an ideal algorithm would work without error and it would generate zero in loss.

2.4.3. FUnction for evaluating the accuracy of model operation - „metrics"

The „metrics“ function is a critical tool in model selection, optimization, and evaluation, and it is a fundamental tool in evaluating the accuracy and performance of machine learning models. These functions compare the predicted values generated by a machine learning model to the true values in the data set to measure performance metrics like accuracy, precision, recall, and F1 score.

The parameter with which we define the „metrics“ function is used to evaluate the performance of the created model. They are like „loss“ functions, since they are responsible for showing the accuracy of the model's work, but unlike them, they do not affect the „training“ of the model. True positive, true negative, false positive, and false negative counts, for example, can be calculated for each time step or for a series of time steps in a sequential model used to classify time-series data. In the shown example, a function was selected that will show the accuracy of the result assessment in percentages (accuracy) during model training.

2.5. TRaining ml model with prepared dataset (fit)

To the previously created mathematical model, the „fit“ function is used to pass the prepared data set and train it (Figure 8). With this procedure, the weighting coefficients (Figure 5) are adjusted to the type and value of the data.

Changing epochs (number of iterations) affects accuracy but also the time for completing the training. ML model creator should set appropriate number that will provide satisfactory results within acceptable timeframe. The influence of epochs on the accuracy is shown in Figures 11 and 12.

```
# 5. Variable 'epochs' present Number of iterations
Number_of_iterations = 20
model.fit(x_train, y_train, epochs=number_of_iterations)
```

Figure 8 - Training ML model (fit)

2.6. Evaluating ml model accuracy and level of error

By using the created ML model and inserting existing data into it (previously separated 20% for testing), we get results that we can compare with real ones. The deviation of the results obtained by the model and the actual results prepared for testing shows the real precision of the model (in this example shown by the standard deviation). The Python code for evaluating the accuracy of the model is shown in Figure 9.

The performance evaluation function is a good starting point for revising the structure and parameters of the generated ML model. In case of need for correction, model parameters are changed.

```
# 6. Accuracy evaluation method - metrics=["accuracy"]
evaluation_result = model.evaluate(x_test, y_test)
standard_deviation = evaluation_result[0]
prediction_accuracy = evaluation_result[1]
```

Figure 9 - ML Model accuracy evaluation

2.7. Saving and using created ml model

If the initial results obtained using the data set for testing are not satisfactory, we return to refining the model by changing the internal structure, parameters and other characteristics that describe it. After testing the functionality of the created model and testing the reliability and accuracy, in case the accuracy meets the set needs, the model is ready for use with a real data set. To the created algorithm, we pass new, real data, without results, which have never been passed to the model before, and the model estimates and predicts the outcome with a certain precision.

Saving the prepared, trained (trained) model is done as in Figure 10. After creating and saving the model, which consists of an architecture, a set of weighting coefficients, optimizers and parameters that define the „oss“ and „metrics“ functions, its subsequent application is possible in different ways and on different platforms.

```
# 7. ML model saving to a file
model.save("myMLmodel.h5")

# 8. ML model loading procedure
From tensorflow import keras
model = keras.models.load_model('path/myMLmodel.h5')
```

Figure 10 - Saving and Loading ML model

At the beginning of the application of the model in practice, it is recommended to control the obtained results with a set of data for verification. These data, as well as the data used for training and testing the model, also contain the final results. In this way, the quality of the model and the reliability of future results can be confirmed with great certainty.

3. RESULTS OBTAINED BY TESTING OF DEVELOPED ML MODEL

Based on the type, amount, and characteristics of the data, it is necessary to set a suitable task and create a plan for the implementation of the algorithm creation project. The basis for a successful model creation process is well-prepared, numerous and diverse data that cover the largest possible number of situations and scenarios. The results are a direct consequence of good data preparation and model design.

All obtained results directly depend on the parameters used for creation (structure, shape, functions, and parameters) and model training („optimizer“ function, „loss“ function, „metrics“ function). The main indicators of the quality of the created model are accuracy (precision) and error level (degree of deviation of the forecasted result from the real one).

For example, modification of some included parameters that define the ML model results in its worse or better performance. The direct correlation of the parameters of the used activation function (sigmoid, hard_sigmoid, tanH, ReLU, swish, linear) with the results is shown in Table 2. The row with the highest precision (sigmoid) is marked.

Table 2. The effect of changing activation function on the accuracy

Activation function	Layers (I, H, O)	Accuracy 20 epochs	Std. Deviation (20 epochs)
hard_sigmoid	32, 40, 1	96.8%	0.026
tanH	32, 40, 1	97.0%	0.024
sigmoid	32, 40, 1	97.7%	0.019
ReLU	32, 40, 1	53.8%	0.462
swish	32, 40, 1	54.3%	0.457
linear	32, 40, 1	48.4%	272

Since the activation function that best matches the type of data in this example (sigmoid) has been established, by changing the number of neurons in the existing three layers (input, hidden and output), a model can be found that meets the needs of the system.

The calculation results are shown in table 3. The row with the highest demonstrated accuracy (32,40,1) is marked.

Table 3. The effect of changing neural layer parameters on precision and error

Layers (In, Hidden, Out)	Accuracy (20 epochs)	Std. Deviation (20 epochs)
40,40,1	97.64%	0.022
32,40,1	97.66%	0.019
32,32,1	96.57%	0.028
24,24,1	96.96%	0.022
16,16,1	97.28%	0.021
8,8,1	97.22%	0.025
4,4,1	96.96%	0.027
2,2,1	94.36%	0.043
1,1,1	94.18%	0.052

A test was run on the same set of data in order to compare the work of the optimization (optimizer) functions [22], the gained precision, and the errors that can be realized by them in the work of the ML model. Table 4 presents the outcomes.

Table 4. The effect of changing the "optimizer" function on the results of the ML model (32, 40, 1; sigmoid, 20 epochs)

Optimizer	Accuracy	Std. Deviation
Adam	96.67%	0.0243
Adadelta	61.62%	0.2429
Adagrad	90.06%	0.1255
RMSprop	97.74%	0.0212
Adamax	97.84%	0.0198
Ftrl	53.84%	0.2478
Nadam	96.96%	0.0247
SGD	94.80%	0.0495

Before accepting and adopting the final solution and parameters for the ML model, it is necessary to analyze the results in detail and take them into account. The following part of the paper is a discussion that explains the results and their dependence on the parameters used.

4. DISCUSSION OF RESULTS

By changing the constructive parameters of the ML model (in the program code in Figures 5, 9, 10), the precision of the forecasted results changes as in Figures 13 and 14 - from the minimum 0.4616 to the maximum 0.9891. The minimum error in the form of standard deviation (the best result) is 0.0099, while the maximum (the worst result) is 0.3899.

It turns out that „ReLU“, „swish“ and „linear“ are not suitable activation functions for data of this type (where a binary result of one or zero is expected), which the results shown prove. Their achieved precision is below 55%, and the error is over 45%.

A three-layer neural network structure was also shown to best fit our data. Testing was performed with the number of iterations up to 500, and by increasing them over 20, no positive change was observed, so 20 was adopted as the number of iterations (epochs) for this data set.

The accuracy values shown (Table 3) represent the probability that the model will predict the outcome for a new data set, never processed before. If we use one input layer with 32 neurons, one hidden layer with 40 neurons and one output layer with 1 output neuron, with activation function „sigmoid“, 20 iterations and other parameters as in the example, we can assume that our model with a probability of 97.66 % accuracy to predict the outcome for a completely new set of data passed to the model.

Increasing the number of iterations (epochs) in the ML model training system has a positive effect up to a certain level (approximately 7 epochs), after which only the processing time increases, but there is no practical benefit. This dependence is shown in Figure 11.

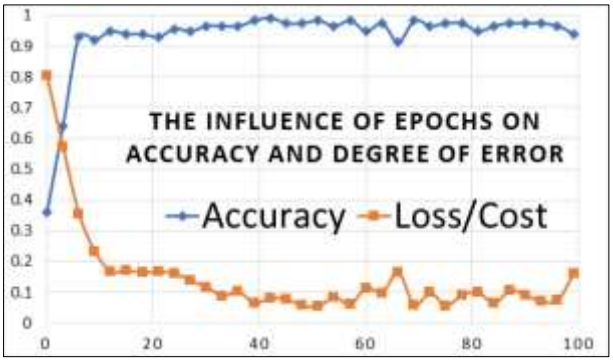

Figure 11 - The influence of number of iterations (epochs) on accuracy and degree of error (loss)

By changing the structural parameters of the function, such as the number of output nodes in the input, hidden and output neural layers, both the precision and the error generated by the function are changed. Increasing the number of output features does not automatically result in a more accurate function. It is necessary to estimate or obtain by iteration the best possible model.

In the presented case, the layers with the output number of nodes from 8 to 40 were examined and it was shown that the combination in which there are 32 nodes at the output of the input layer and 40 nodes at the output of the hidden layer gives the most accurate results, with the smallest degree of error (in this example - standard deviations). The difference can be seen in Figures 11 and 12.

By changing the number of iterations (epochs), in this example from 1 to 20, the conclusion is that over 5 there is no benefit in the form of an increase in precision, but the training time of the model is significantly extended.

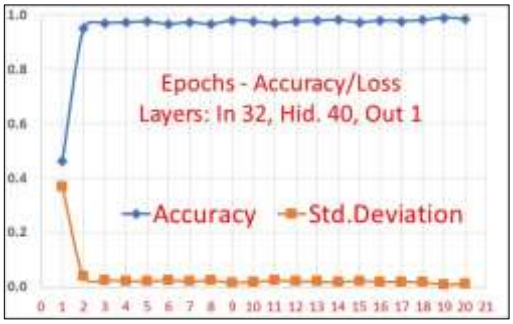

Figure 12 - The best combination of the number of output nodes on the layers of the created model

5. CONCLUSION

Depending on the type and type of data we work with, as well as the results we expect, the acceptable limits of accuracy that the ML model will reach are relative and subjective. Highest accuracy with lowest error rate is a goal.

The result largely depends on the quality, variety and quantity of data, and the database used in this example proved to be suitable for creating a quality ML model. The result of almost 98% accuracy confirms the extremely high level of accuracy.

To successfully create a ML algorithm, it is necessary to:

- Recognize and understand the goal of the project
- Analyze, process, categorize, and prepare data
- Evaluate the duration of data training
- Apply appropriate algorithms for the type and amount of data that the model will work with
- Set the task and goal of the ML model
- Through iterative testing, to come up with a model that best fits the data

After successfully creating a model and saving it, depending on the need, there are wide possibilities of implementation. Some of the most common are:

- publishing it on the internet and/or creating a web service for access from any location
- use as a local application on the computer or similar device on which it is installed
- use on mobile devices in the form of appropriate applications

The field of machine learning and artificial intelligence has a wide application with a significant number of examples in use. Python, with its libraries, has proven as a package that can be used to answer numerous challenges in the field of AI and ML, and which, with digitalization and the development of the fourth industrial revolution, is increasingly necessary and applicable in the business and scientific world.

REFERENCES

[1] Radaković, M. Audio Signal Preparation Process for Deep Learning Application Using Python – *Sinteza 2021-International Scientific Conference on IT and Data Related Research*. Singidunum University, pp. 146-152, 2021.

[2] Radaković M, Nestorov S, Radaković K. Veštačka inteligencija i računari kao pomoć u školskom i vanškolskom obrazovanju dece sa smetnjama u razvoju, *Multidisciplinarni pristupi u edukaciji i rehabilitaciji, Sarajevo, Bosna i Hercegovina*, pp. 351-360, 2021.

[3] Gholizadeh S. Top Popular Python Libraries in Research - *Authorea Preprints*, pp.142-145, 2022.

[4] Nagy Z. Osnove veštačke inteligencije i mašinskog učenja, *Kompjuter Biblioteka*, pp. 46, 2019

[5] Banoula M. Machine Learning Steps: A Complete Guide [Internet]. [cited 15.11.2022] Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps

[6] Erickson, Bradley J. et.al. Toolkits and Libraries for Deep Learning - *Journal of Digital Imaging*, 2017.

[7] Ahmad, Zeeshan et. al. Network intrusion detection system: A systematic study of machine learning and deep learning approaches - *Transactions on Emerging Telecommunications Technologies*, 2021.

[8] World Bank Open Data [Internet]. *World Bank Group*, [cited 06.01.2023]. Available: https://data.worldbank.org/

[9] Abaoijang. Network Intrusion Detection Dataset Download [Internet]. [cited 31.12.2022] Available: https://www.kaggle.com/code/abaojiang/network-intrusion-detection-a-simple-ml-workflow/data

[10] Splitting Your Dataset with Scitkit-Learn train_test_split [Internet]. *Datagy website*. [cited 5. Jan. 2023]. Available: https://datagy.io/sklearn-train-test-split/

[11] Nduati J. Introduction to Neural Networks [Internet]. [cited 25.12.2022]. Available: https://www.section.io/engineering-education/introduction-to-neural-networks/

[12] Single Layer Perceptron in TensorFlow [Internet]. *Website javatpoint.com*. [cited 01.01.2023]. Available: https://www.javatpoint.com/single-layer-perceptron-in-tensorflow

[13] Jadon S. Introduction to Different Activation Functions for Deep Learning [Internet]. [cited 16. 01. 2023]. Available: https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

[14] Sarkar A. et al. An Empirical Comparison of Optimizers for ML Models - *IEEE Access*, 9, 52102-52115. DOI: 10.1109, 2021.

[15] A. Sun, A. Varshney, Y. Shy. A comprehensive review of loss functions in neural networks – *Neurocomputing*, vol. 440, pp. 242-261, 2021.

REZIME

ANALIZA PRIMERA KREIRANJA MODELA VEŠTAČKE INTELIGENCIJE KORIŠĆENJEM PROGRAMSKOG JEZIKA PYTHON – BENEFITI I IZAZOVI

Praktične primene sistema veštačke inteligencije i mašinskog učenja su brojne i raznovrsne i već danas su implementirane u različitim oblastima. Mogu se pronaći u računarskim sistemima, internet pretraživačima, pametnim kućnim uređajima, navigaciji i upravljanju vozilima, prepoznavanju oblika i zvukova, u medicini, ekonomiji, poljoprivredi, itd. Uspešnost njihove realizacije, rezultati i vreme izrade ovakvih modela se razlikuju, a trendovi usmeravaju njihove kreatore na metode koje omogućavaju bržu izradu i kvalitetnija i preciznija rešenja. Ovaj rad ima za cilj da kroz analizu jednog primera kreiranja modela veštačke inteligencije korišćenjem programskog jezika Python, prikaže benefite i izazove njegove implementacije. Kroz prikaz njegovog projektovanja i primere kôda, za realan set podataka, prikazane su mogućnosti i način izrade funkcionalnog modela mašinskog učenja. Korišćeni set podataka sadrži informacije potrebne za analizu i detekciju 'cyber' napada u vidu neovlašćenog upada u računarsku mrežu.

Ključne reči: *veštačka inteligencija, mašinsko učenje, cyber bezbednost, računarske mreže, python*