Enhancing Hardware Security: A Comparative Study of Machine Learning and Deep Learning Approaches for Detecting Hardware Trojans

Sharmila Sivalingam
EECE 7390 - Computer Hardware Security
Northeastern University
Boston, USA
sivalingam.s@northeastern.edu

***Abstract*—In the current landscape, the security of IoT devices garners significant attention from both researchers and malicious actors due to their vulnerabilities. Hence, prioritizing hardware security is imperative, particularly in countering hardware trojans, which pose substantial threats to device integrity. This project underscores the critical importance of secure integrated circuits (ICs) in ensuring the safety of IoT devices globally. The rapid pace of technological advancement compels modern businesses to outsource circuit design, inadvertently creating opportunities for hardware trojans to infiltrate. These insidious trojans clandestinely compromise ICs, leading to data breaches and performance degradation, with potentially devastating consequences. Mitigating these risks necessitates a multifaceted approach, incorporating traditional and machine learning-based methodologies. Through meticulous experimentation, various machine learning and deep learning algorithms were assessed for their efficacy in detecting trojan-infected ICs. Notably, the Gradient Boosting algorithm emerged as the most accurate for hardware trojan detection and accuracy is 99.56% . Additionally, the model's performance was benchmarked against existing projects, confirming its superior accuracy. This study underscores the critical need for vigilance in addressing evolving cybersecurity threats and offers actionable insights for bolstering digital infrastructure security.**

Keywords—Hardware trojan, Machine learning, Integrated circuits.

I. INTRODUCTION:

In today's era of rapid technological advancement, the proliferation of sophisticated software and hardware solutions has become synonymous with progress. However, this complexity has also ushered in a new era of security challenges, particularly in the realm of integrated circuits (ICs). The integration of small sensors into an ever-expanding array of devices has compounded these challenges, necessitating innovative solutions to safeguard against potential threats.

One such threat that has emerged amidst this technological landscape is the proliferation of hardware trojans (HTs). These insidious forms of malware can be surreptitiously introduced into ICs at various stages of development, remaining dormant until triggered by a predetermined activation mechanism. Once activated, HTs can wreak havoc on IC functionality, leading to unexpected failures, circuit damage, and even the compromise of sensitive data, regardless of encryption measures in place.

Addressing the threat posed by HTs requires a multifaceted approach that spans both the pre-silicon and post-silicon phases of IC development. Pre-silicon verification processes aim to detect any unauthorized alterations to IC designs, while post-silicon testing involves comparing circuit behavior with a validated "golden" model or reverse-engineering the manufacturing process to verify IC integrity.

Despite efforts to mitigate these risks, the inherently deceptive nature of HTs and their wide-ranging potential for exploitation pose significant challenges. As such, ongoing academic research and development efforts are focused on enhancing detection and prevention mechanisms to safeguard against the proliferation of hardware trojans and uphold the integrity of our digital infrastructure.

Detecting hardware trojans early in the lifecycle of IoT devices is crucial for maintaining the security and integrity of these interconnected systems. Hardware trojans pose significant threats to the foundational components of IoT hardware, such as integrated circuits (ICs), which are vulnerable to infiltration by malicious actors. Early detection enables organizations to proactively address potential security breaches and implement countermeasures to safeguard sensitive data and prevent disruption to IoT operations. By identifying hardware trojans at their inception, organizations can develop tailored security protocols to mitigate risks and minimize the likelihood of costly security incidents. Prioritizing early detection measures is essential for preserving the reliability and functionality of IoT devices and networks, thereby ensuring the continued effectiveness and efficiency of connected objects in various industries, including healthcare, automotive, military, and smart metering.

In our study will delve into related works in Section II, outline our methodology in Section III, describe the experimental setup in Section IV, and provide clear insights into results and discussions in Section V.

II. RELATED WORKS:

In the realm of computing, a fundamental disparity exists between machines and humans: machines lack the innate ability to learn from experience, relying instead on explicit instructions for task execution. Machine Learning (ML) serves as a pivotal bridge across this chasm, empowering

computers to glean insights from past experiences and data through computational algorithms and mathematical models. knowledge from data, obviating the need for explicit programming. In the contemporary era, industries are generating vast volumes of data at an unprecedented rate, commonly referred to as Big Data. This data encompasses a diverse array of numerical values and categorical information, essential for training ML models. Data collection entails the meticulous gathering of information from various sources, including websites, institutional databases, and sensor networks, to form comprehensive datasets for analysis. Preparing data for ML analysis is a crucial step in ensuring the accuracy and reliability of subsequent model outcomes. This process involves cleansing the data to remove duplicates, errors, and biases, thereby enhancing its quality and integrity. Additionally, data visualization techniques are employed to uncover patterns, identify outliers, and gain insights into the underlying structure of the dataset. Selecting the appropriate ML model is paramount to the success of any data-driven analysis. Factors such as model accuracy, scalability, and interpretability are carefully considered during this stage. Supervised learning algorithms utilize labeled data to make predictions, while unsupervised learning methods uncover hidden patterns and structures within unlabeled data, facilitating tasks such as clustering and dimensionality reduction. Model training involves iteratively optimizing the model's parameters to improve its performance. Supervised learning algorithms learn from labeled data to make accurate predictions, while unsupervised learning algorithms derive insights from unlabeled data to identify underlying patterns and structures. After training, the model's performance is evaluated using unseen data to assess its accuracy and generalization capabilities. Increasing the size of training and testing datasets can lead to more robust and reliable model evaluation. Fine-tuning model parameters is essential for optimizing performance and achieving the desired outcomes. Adjusting parameters such as learning rates and regularization terms can significantly impact the model's predictive accuracy and stability. Once trained and evaluated, the ML model is ready to make predictions based on learned patterns and insights from the dataset. These predictions enable informed decision-making in various domains, ranging from healthcare and finance to manufacturing and marketing.

Machine Learning (ML) employs various learning models to tackle classification and regression challenges. Among the most common are Logistic Regression (LR), K-Nearest Neighbors (KNN), Support Vector Machines (SVMs), Random Forests (RF), and Gradient Boosting (GB). LR is a supervised learning algorithm utilized for classification tasks, employing a logistic sigmoid function to transform its output into discrete classes. KNN, on the other hand, relies on the principle of similarity, assigning labels to data points based on their proximity to neighboring points in the feature space.

As a subfield of Artificial Intelligence (AI), ML endeavors to equip systems with the capability to autonomously acquire
SVMs, rooted in statistical learning theories, are versatile models used for classification and regression tasks. They utilize a kernel trick to transform input features into a higher-dimensional space, enabling the identification of a hyperplane that maximizes the margin between different classes.

RF employs decision trees, graphical representations of data, to describe potential outcomes of decisions. By iteratively splitting data based on feature attributes, decision trees construct a predictive model that can handle both classification and regression tasks.
Gradient Boosting, a powerful ensemble method, combines multiple weak prediction models, typically decision trees, to create a robust predictive model. By iteratively improving upon the predictions of previous models, Gradient Boosting produces a final ensemble model that offers superior performance in classification and regression tasks.
Each of these learning models offers unique advantages and is suited to different types of data and problem domains. Understanding their principles and characteristics is essential for effectively applying ML techniques in practical scenarios.

III. METHODOLOGY

A. Proposed Model:

To address the challenge of identifying Hardware Trojan Attacks, we propose a comprehensive Hardware Trojan detection and classification technique. The primary focus lies on constructing a Model Builder framework for efficient data collection, categorization, preprocessing, model training, and feature selection. In our comparative analysis, we will evaluate the performance of five machine learning models, namely Support Vector Machine (SVM), Gradient Boosting (GB), K-nearest neighbors (KNN), logistic regression (LR), and random forest (RF). Additionally, we will explore the capabilities of deep learning methods, specifically Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN), for IC classification.

In the "Model Training" phase, we will utilize the "Sequential" model class of Keras to construct our initial model architecture. This model comprises an input layer followed by three dense layers, where the output of each layer serves as the input for the subsequent layer. To mitigate the risk of overfitting, a dropout layer is introduced, followed by a fully connected layer and a dense layer with a sigmoid function. Configuring the learning process is crucial, achieved through the 'compile' function, before training the model using the provided training data to optimize weights and improve model performance. Through this proposed approach, we aim to develop a robust and accurate detection and classification system for Hardware Trojan Attacks.

B. Dataset procession and feature extraction:

The experiments were conducted using a dataset sourced from research benchmarks available in Trust-Hub, a reputable public repository containing both Trojan-free (TF) and Trojan-infected (TI) circuits. Our dataset was compiled using all available circuits from Trust-Hub, totaling approximately 1,000 circuits. Among these, 21 circuits were identified as TF, while the remaining circuits were classified as TI. Through an extraction process, the final dataset was constructed, comprising 50 distinct characteristics for each circuit.

Throughout the dataset refinement process, we addressed minor discrepancies such as missing features to enhance data integrity. After an initial set of 50 characteristics was evaluated, we strategically removed highly correlated features, retaining a subset of 24 relevant characteristics. Despite the limited presence of only 21 instances labeled as Trojan-free (TF) out of 1,000 circuits, we employed a resampling technique to balance the TF-to-TI ratio by replicating each TF instance to match the total number of TI instances within the same circuit category. This technique, while not mandatory, is recommended for managing imbalanced datasets. Furthermore, we conducted feature selection by eliminating redundant features to streamline our algorithm, prioritizing the retention of essential features crucial for accurate predictions. This approach optimized both cost and time efficiency.

IV. EXPERIMENT AND RESULTS

A. Implementation:

In the implementation phase, various machine learning and deep learning models were instantiated and configured for experimentation.

For the deep learning model, a Sequential model was constructed using the Keras library. This model consisted of three densely connected layers, with ReLU activation functions for the hidden layers and a softmax activation function for the output layer. The model was compiled using the Adam optimizer and categorical cross-entropy loss function, with metrics including mean squared error (MSE) and accuracy.

For the Support Vector Machine (SVM) model, the SVC function from the sklearn library was employed, utilizing a radial basis function (RBF) kernel with specified parameters for C and gamma values.The Logistic Regression (LR) model was implemented using the LogisticRegression function from the sklearn library, configuring parameters such as the solver and maximum number of iterations.The Gradient Boosting (GB) model was instantiated using the GradientBoostingClassifier function from the sklearn library, with specified parameters for learning rate and number of estimators. For the K-Nearest Neighbors (KNN) model, the KNeighborsClassifier function from the sklearn library was utilized, setting the number of neighbors to 3. Lastly, the Random Forest (RF) model was created using the RandomForestClassifier function from the sklearn library, with parameters including the number of estimators, maximum depth, and random state.

These implementations form the basis for conducting experiments and evaluating the performance of each model in the context of hardware Trojan detection.

B. Results:

The Python script (Appendix A) implements a comparison of various machine learning and deep learning models for a hardware Trojan detection task. The script imports necessary modules including time, sys, itertools, matplotlib.pyplot, and functions from different Python files corresponding to the machine learning and deep learning models, as well as data preparation.

The main function loads and prepares the dataset using the prepare_data function from the load_data module. Then, it calls each machine learning and deep learning model function (svm, random_forest, dl_model, gradient_boosting, k_neighbors, logistic_regression) passing the training and testing data as arguments to train and evaluate the models. Each model function returns the training time and accuracy of the respective model (TABLE I).

TABLE I. ALGORITHM WITH ACCURACY AND EXECUTION TIME

Algorithm	**Execution Time (seconds)**	**Accuracy**
MLP	45.34	90.64
SVM	0.49	77.31
RF	0.02	99.01
GB	0.23	99.56
KNN	0.02	99.45
LR	0.02	71.81

After obtaining the training times and accuracies of each model, the script plots two bar charts using matplotlib.pyplot. The first bar chart compares the accuracy of each model (Fig. 1.), while the second bar chart compares the execution time of each model (Fig. 2.). Different colors are used to distinguish between the models in the plots.

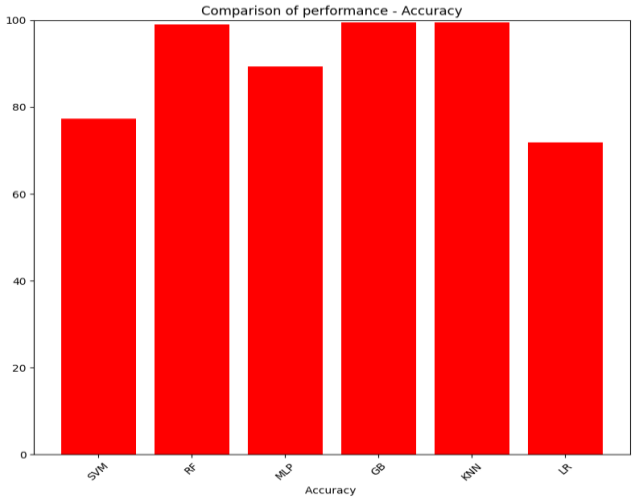

Fig. 1. Compare the accuracy of each model.

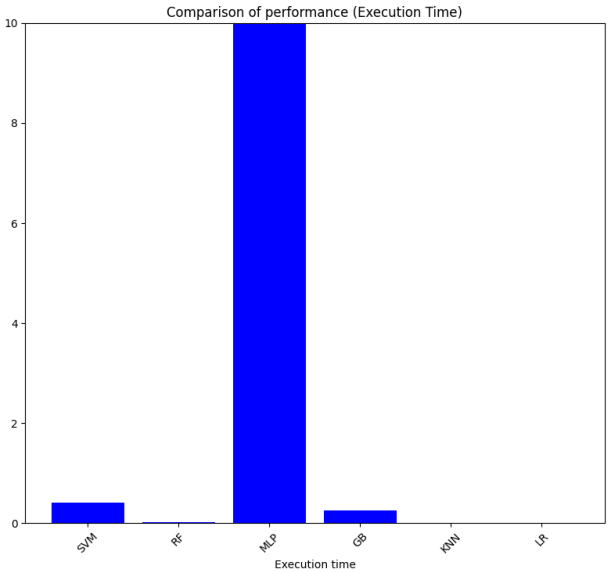

Fig. 2. Compare the execution time of each model.

The confusion matrix is a performance evaluation tool used to visualize the performance of a classification model. In this context, the rows represent the true labels of the data, while the columns represent the predicted labels. Specifically, in this example, there are two classes: "trojan free" and "trojan infected."

The confusion matrix displays the count of instances where the true label matches the predicted label for each class combination. For instance, [trojan free, trojan free] indicates the number of instances correctly classified as trojan free, which is 648. Similarly, [trojan infected, trojan infected] represents the count of instances correctly classified as trojan infected, which is 256.

The matrix allows for a comprehensive understanding of the model's performance, highlighting areas of correct and incorrect predictions for each class.

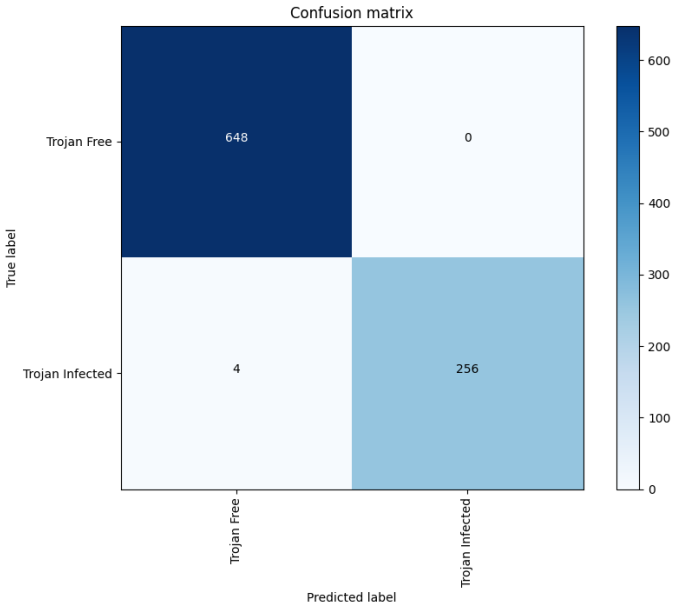

Fig. 3. Confusion Matrix.

V. CONCLUSION

Annually, the rapid advancement and complexity of Hardware Trojans pose a significant challenge. The evolution of intricate strategies in their design infects a broad spectrum of integrated circuits, ranging from commonplace circuits in daily utilities to cutting-edge ones utilized in military, industrial, and financial sectors, including sensitive state-funded research areas. Manifesting stealthily, Hardware Trojans exhibit diverse formations and types, causing substantial financial losses to global economic and technological enterprises annually. Left unchecked, these intrusion attempts could impede future technological advancements, severely impacting various aspects of daily life. Given the escalating uncertainties surrounding the design and manufacturing of secure integrated circuits, there arises a pressing need for efficient, adaptable, and scalable countermeasures.

To address Hardware Trojan detection, this study proposes a methodology focusing on the GLN phase of the ASIC circuit via area and power characteristics. Utilizing Design Compiler NXT software for circuit design sourced from the Trust-HUB library, we employed seven ML models for comparison: MLP, RF, SVM, GB, KNN, LR, and XGB. The GB model exhibited the highest precision, achieving an accuracy rate of 99.54%.

REFERENCES

[1] S. Bhunia et al., "Protection against hardware trojan attacks: Towards a comprehensive solution," IEEE Des. Test, 2013, doi: 10.1109/MDT.2012.2196252.

[2] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection", IEEE Design Test Comput., vol. 27, no. 1, pp. 10-25, Jan.—Feb. 2010.

[3] S. Garcia, A. Zunino, and M. Campo, "Botnet behavior detection using network synchronism," in Privacy, Intrusion Detection and Response: Technologies for Protecting Networks. Hershey, PA, USA: IGI Global, 2012, pp. 122—144

[4] K. Hasegawa, M. Oya, M. Yanagisawa and N. Togawa, "Hardware Trojans classification for gate-level netlists based on machine learning", Proc. IEEE 22nd Int. Symp. On-Line Testing Robust Syst. Des., pp. 203-206, 2016.

[5] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, "Trojan detection using IC fingerprinting", Proc. IEEE Symp. Secur. Privacy, pp. 296-310, 2007.

[6] H. Salmani and M. Tehranipoor, "Trust-Hub", Oct. 2020, [online] Available: https://trust-hub.org/home.

[7] S. Yu, C. Gu, W. Liu and M. O'Neill, "A novel feature extraction strategy for Hardware trojan detection", Proc. IEEE Int. Symp. Circuits Syst., pp. 1-5, 2020.

[8] S. Yu, C. Gu, W. Liu and M. O'Neill, "Deep Learning-Based Hardware Trojan Detection With Block-Based Netlist Information Extraction," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 4, pp. 1837-1853, 1 Oct.-Dec. 2022, doi: 10.1109/TETC.2021.3116484.

Appendix A: Python Script

A. Main:

```
import time
import sys
import itertools
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from load_data import prepare_data
from dl_model1 import dl_model
from svm import support_vector_machine
from random_forest import random_forest
from gradient_boosting import gradient_boosting
from knn import k_neighbors
from logistic_regression import logistic_regression

# main
if __name__ == '__main__':
    x_train, x_test, y_train, y_test = prepare_data()
    dl_model_time, dl_model_accuracy = dl_model(x_train,
x_test, y_train, y_test)
    svm_time, svm_accuracy =
support_vector_machine(x_train, x_test, y_train, y_test)
    rf_time, rf_accuracy = random_forest(x_train, x_test,
y_train, y_test)
    grad_time, grad_accuracy = gradient_boosting(x_train,
x_test, y_train, y_test)
    k_time, k_accuracy = k_neighbors(x_train, x_test,
y_train, y_test)
    lr_time, lr_accuracy = logistic_regression(x_train, x_test,
y_train, y_test)
    accuracy = [svm_accuracy, rf_accuracy,
dl_model_accuracy, grad_accuracy,
                k_accuracy, lr_accuracy]
    time_ = [svm_time, rf_time, dl_model_time, grad_time,
k_time, lr_time]
    plt.ylim(0, 100)
    plt.xlabel("Accuracy ")
    plt.title("Comparison of performance - Accuracy")
    l1, l2, l3, l4, l5, l6 = plt.bar(["SVM", "RF", "MLP",
                                      "GB", "KNN", "LR"],
                                      accuracy)
    plt.xticks(rotation=45)
    l1.set_facecolor('r')
    l2.set_facecolor('r')
    l3.set_facecolor('r')
    l4.set_facecolor('r')
    l5.set_facecolor('r')
    l6.set_facecolor('r')
    plt.show()
    plt.close('all')

                color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                horizontalalignment="center"
    plt.ylim(0, 10)
    plt.xlabel("Execution time")
    plt.title("Comparison of performance (Execution Time)")
    c1, c2, c3, c4, c5, c6 = plt.bar(["SVM", "RF", "MLP",
                                      "GB", "KNN", "LR"],
   time_)
    c1.set_facecolor('b')
    c2.set_facecolor('b')
    c3.set_facecolor('b')
    c4.set_facecolor('b')
    c5.set_facecolor('b')
    c6.set_facecolor('b')
    plt.xticks(rotation=45)
    plt.show()
```

B. Confusion Matrix:

```
import time
import sys
import itertools
import matplotlib.pyplot as plt
from preprocess_data import prepare_data

import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

def plot_confusion_matrix(cm, target_names, title,
cmap=None, normalize=False):
    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=90)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:,
np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.1f}".format(cm[i, j]),
                horizontalalignment="center",


                color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label')
    plt.show()
```

C. *Load data:*

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import seaborn as sns

def create_numerics(data):
    # Get nominal columns
    nominal_cols =
data.select_dtypes(include='object').columns.tolist()
    # Turn nominal to numeric
    for nom in nominal_cols:
        enc = LabelEncoder()
        enc.fit(data[nom])
        data[nom] = enc.transform(data[nom])
    return data
def prepare_data():
    data = pd.read_excel("code/data.xlsx")
    data = data.dropna()

    to_drop = [column for column in upper.columns if
any(upper[column] > 0.95)]
    # Drop features
    data = data.drop(data[to_drop], axis=1)
    y = pd.DataFrame(data["Label"]).values
    x = data.drop(["Label"], axis=1)
    trojan_free = data.loc[data['Label']=="'Trojan
Free'"].reset_index()
# balance the ratio between trojan free and infected of the
same circuit category
    for i in range(len(trojan_free)):
        category_substring =
trojan_free['Circuit'][i].replace("'","")
        circuit_group =
data[data['Circuit'].str.contains(category_substring)]
        df1 = circuit_group.iloc[0:1]
        if len(circuit_group) > 1:
            # data = data.append([df1]*(len(circuit_group)-1),
ignore_index=True)
            data = pd.concat([data,
pd.concat([df1]*(len(circuit_group)-1),
ignore_index=True)])
 data.drop(columns=['Circuit'], inplace=True)
    data = create_numerics(data)
    data = shuffle(data, random_state=42)
    # Create correlation matrix
    corr_matrix = data.corr().abs()
    # Select upper triangle of correlation matrix
    upper =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
                              k=1).astype(bool))
    # Find index of feature columns with correlation
greater than 0.95
    scaler = MinMaxScaler(feature_range=(0, 1))
    x = scaler.fit_transform(x)
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=1)
    return(x_train, x_test, y_train, y_test)
```