NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

UDC 004.8

Research Project Report

on the topic "Detecting and analysis of fraudulent transactions in the financial sector using machine learning methods"

(interim, the first stage)

Fulfilled by the Student:
group #БПАД___211___ Evelina Urusova Victorovna
Surname, First name, Patronymic, if any

Checked by the Project Supervisor:
Valiullin Adel Marsovich
Surname, First name, Patronymic (if any), Academic title (if any)
Head of the Center for Artificial Intelligence Technology
Job
Gazprombank JSC
Place of Work (Company or HSE Department)

Moscow 2024

Table of contents

Аннотация

В данной работе исследуется проблема определения финансовых мошенничеств с применением методов машинного обучения. Акцент сделан на разработке моделей для мгновенного выявления подозрительных транзакций. Особое внимание уделяется использованию классических моделей, таких как Xgboost, K-neighbours, Случайный лес и прочих, а также автокодировщиков для анализа многомерных данных. Работа также включает создание API, позволяющего интегрировать разработанные модели в банковские системы. Результаты показывают высокую эффективность предложенных решений в идентификации мошеннических операций.

Annotation

This project investigates the problem of detecting financial fraud using machine learning methods. The focus is on developing models for the immediate identification of suspicious transactions. Special attention is given to the use of classical models such as Xgboost, K-neighbors, Random Forest, and others, as well as autoencoders for analyzing multidimensional data. The work also includes the creation of an API that allows the integration of developed models into banking systems. The results demonstrate the high effectiveness of the proposed solutions in identifying fraudulent operations.

Key words:

Financial fraud, fraud detection, machine learning, Xgboost, autoencoder, neural networks, API creation.

1. Introduction

1.1 Domain Description

Fraud, or deception, is the illegal use of deception, any attempts to cheat the system or derive undue benefit. Financial fraud is one of the most common forms of fraud. Examples include credit fraud, insurance fraud, financial machinations, and tax evasion, and finally payment fraud, such as using someone else's credit or debit card to make purchases or withdraw money. My work on anti-fraud is focused on measures and technologies designed to prevent, detect, and respond to fraudulent transactions.

The relevance of anti-fraud measures in the payment sphere has particularly increased in recent years. For example, the COVID-19 pandemic accelerated the shift to digital payments, as evidenced by statistical data. In early 2020, particularly in January, about 22.9 million online payments were registered. However, by November, this figure had increased by 40%, reaching 32 million transactions. This trend continued to strengthen in 2021, with the volume of online payments increasing by another 20%. According to the latest data provided by the top ten banks, seven out of ten Russians now pay for goods and services using smartphones. It should also be noted that according to the Russian Prosecutor General's Office, at the beginning of 2020 there was a significant increase in crimes involving the use of electronic payment instruments — their number increased by 120%.

Such a trend in the growth of digital payments and the consequent fraud necessitates the strengthening of anti-fraud measures.

1.2 Problem Statement

In this work, the task of developing and implementing various machine learning models for the real-time detection of fraudulent transactions was addressed. The main goal was to ensure high accuracy in identifying anomalous transactions with minimal delay, as it is critically important to promptly respond to potential threats to prevent financial losses, enhance user experience, and ultimately comply with regulatory requirements imposed in many jurisdictions.

In addition to classical machine learning models, special attention was given to implementing an autoencoder. Its primary task was to compress data into a latent space with subsequent reconstruction of the input information, which is particularly useful in high-dimensional datasets. Another advantage of implementing such an artificial neural network is that it involves unsupervised learning, which is especially important when labels are absent, and in the context of fraudulent transactions, transaction data often contain sensitive information and access to labels may be limited.

Additionally, fraudulent transactions are a relatively rare occurrence compared to the total number of transactions, creating a class imbalance problem, making the development of an autoencoder a relevant task.

Additionally, the project also includes the development of a basic API as a product, allowing the integration of models with the best performance into existing bank information systems.

2. Literature Review

2.1 Cybertonica

In the exploration of contemporary approaches to financial fraud detection, particular attention was given to the domestic company Cybertonica, whose innovative solutions in cybersecurity and anti-fraud merit consideration. To study the company and its technologies, I utilized their official website, blog, presentations, and videos from professional conferences.

Cybertonica began as a startup but, thanks to its innovative approaches, successfully grew into an international organization. Their primary tool involves analyzing user behavioral models to detect deviations from the norm and assign scores to anomalies as potential threats. This process is analogous to using a labeled grid that covers and analyzes various behavioral patterns, including predefined indicators of deception.

One of the challenges faced by the company was reducing transactional fraud among merchants in acquiring. The company employed classification methods with key criteria: MCC codes and geographical analysis. For instance, transactions from specific countries, such as Turkey from the list of top 100 riskiest merchants, underwent additional analysis. The company successfully managed to reduce chargeback levels by 40%; however, there exists a challenge related to working with merchants indirectly through a payment gateway as an intermediary. It is crucial not only to analyze data from payment gateways but also to work directly with merchants to implement necessary protective mechanisms, such as scripts on e-commerce platforms that can gather additional data. Additionally, there may be insufficient data. For example, with 4000 records distributed across 20 categories, each category receives an average of only 200 records. This quantity may be insufficient for detecting complex patterns and dependencies, especially if the data is heterogeneous and contains a lot of noise or exceptions.

Another task assigned to the company was to protect loyalty programs and card payments in taxis during the 2018 FIFA World Cup due to high activity. Potential frauds could include clients using stolen cards or deliberate underreporting of trip durations by drivers for fare manipulation. To address

this issue, the company collected various user data, including their ratings and transaction histories, to form profiles and focused on "good" clients since they outnumber fraudulent ones, thereby reducing Type I and Type II errors and increasing conversion rates.

Cybertonica utilizes advanced technologies in user behavior analytics, machine learning, and threat intelligence for fraud detection and prevention. However, their current model lacks the application of artificial intelligence methods such as autoencoders, which are used in my project. This not only enhances fraud detection accuracy but also reduces false positives. Thus, my system can more effectively distinguish genuine fraud cases from false alarms by analyzing the standard behavior of "good users" and detecting deviations from these norms. Additionally, the accuracy of my classical machine learning model reaches 94%, with a recall rate of 77%, whereas models developed by Cybertonica exhibit an average accuracy of 70% and a recall rate of 68%. These results confirm the effectiveness of my work, although it's worth considering potential differences in the datasets on which our models were trained and tested, as well as the diversity of testing conditions—from banking to gaming sectors—as this can significantly impact model performance.

2.2 Publication "Safeguarding against Cyber Threats: Machine Learning-Based Approaches for Real-Time Fraud Detection and Prevention”

In the course of reviewing relevant publications, an article authored by researchers from the Department of Data Sciences and Computer Applications at Manipal Institute of Technology was also analyzed. In their research, they propose machine learning methods used for detecting financial fraud. The primary motivation for studying this article was that fraud detection, according to the article, occurs in real-time based on decision trees, logistic regression, and artificial neural networks (ANN).

The research methodology includes using a dataset from Kaggle, which contains transaction types, account balances, and fraud indicators. Data analysis involves preprocessing methods such as handling missing values and encoding categorical variables. Model evaluation is conducted using metrics such as accuracy, precision, recall, and F1-score.

Thus, decision trees are used for classification and regression, where each decision in a node is based on maximizing information gain, determined through Gini impurity. Logistic regression predicts the probability of an example belonging to a class using a sigmoid function, while ANN models complex nonlinear dependencies in data, enabling deep learning and pattern recognition for effective fraud detection.

The results of the higher education academy's research showed that the models demonstrate high accuracy in detecting fraudulent transactions, as evidenced by confusion matrices and learning

curves. However, artificial neural networks emerged as the leaders, showing particularly good results in adapting to dynamically changing threats, making them the most valuable tool in combating cyber fraud.

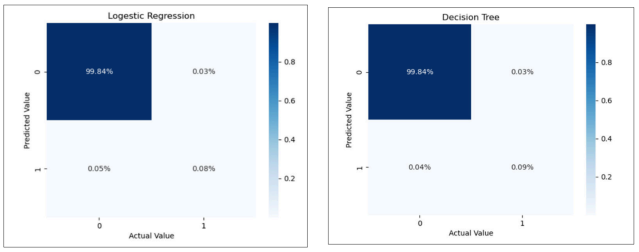


Figure 1. Confusion matrix for logistic regression and decision tree.

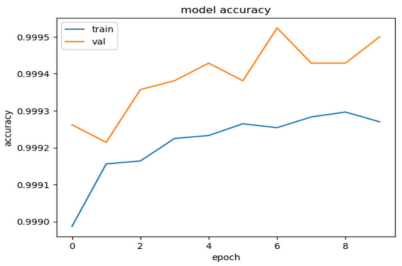


Figure 2. Training and validation accuracy for ANN.

3. Experimental Research

3.1 Data Analysis and Preparation

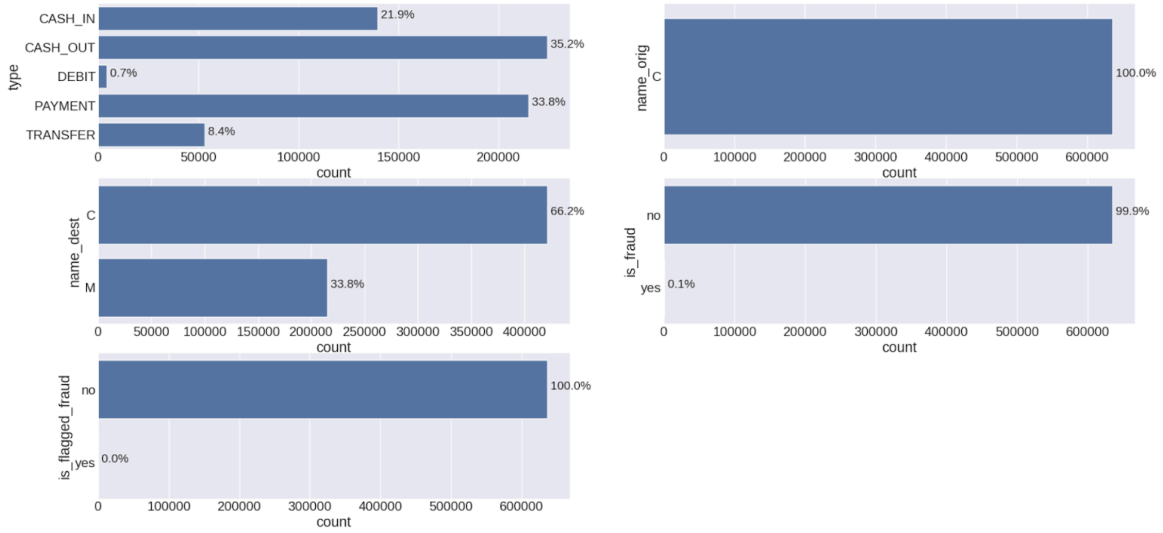
For the conduct of this research, I utilized a dataset on financial transactions provided by Kaggle. In the preliminary analysis phase, the dataset underwent downloading and initial data processing. It comprises information on 636,262 transactions collected over 30 days (744 hourly intervals, designated as 'step'). Each record includes data on the transaction type ('type'), amount ('amount'), initiator ('nameOrig'), initial and final balances of the sender ('oldbalanceOrg', 'newbalanceOrig'), recipient ('nameDest'), as well as the initial and final balances of the recipient ('oldbalanceDest', 'newbalanceDest'). Additionally, the dataset identifies transactions classified as fraudulent ('isFraud') and flags those presumed to involve large-scale transfers exceeding 200,000 ('isFlaggedFraud').

In the data preparation process, I standardized the column names, converting them to snake_case to ensure uniformity and facilitate subsequent encoding. I also conducted a check for missing values (NA), which confirmed the integrity of the data – missing values are absent across all dataset columns. This indicates the high quality of the collected data and allows for its detailed analysis without the need for preliminary cleaning.

Subsequently, I segmented the columns into numerical and categorical data. For the categorical variables, I performed a descriptive statistical analysis, which revealed key characteristics of the distributions:

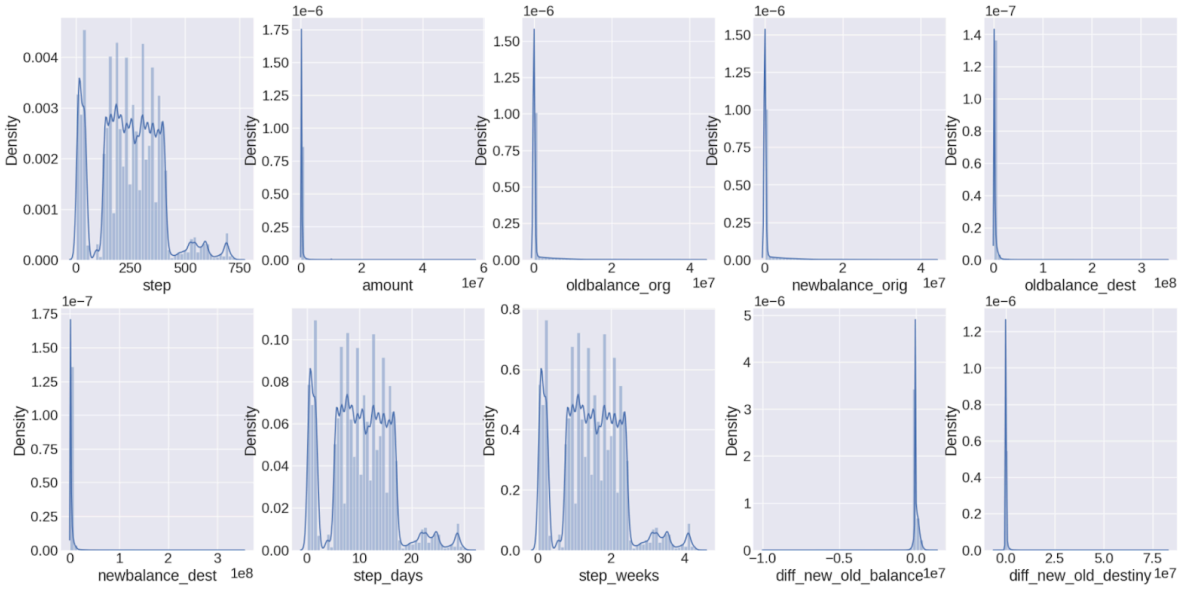
1. The coefficient of variation for all data exceeds 25%, indicating a lack of homogeneity.
2. The 'step' variable ranges from 1 to 742 hours, equivalent to 30 days.

3. Some variables exhibit a high peak and are right-skewed.
4. 50% of the values in 'newbalance_orig' are 0. This might suggest that some transfers do not reach the destination.
5. The skewness is significantly positive, suggesting that values may be concentrated in a smaller range.

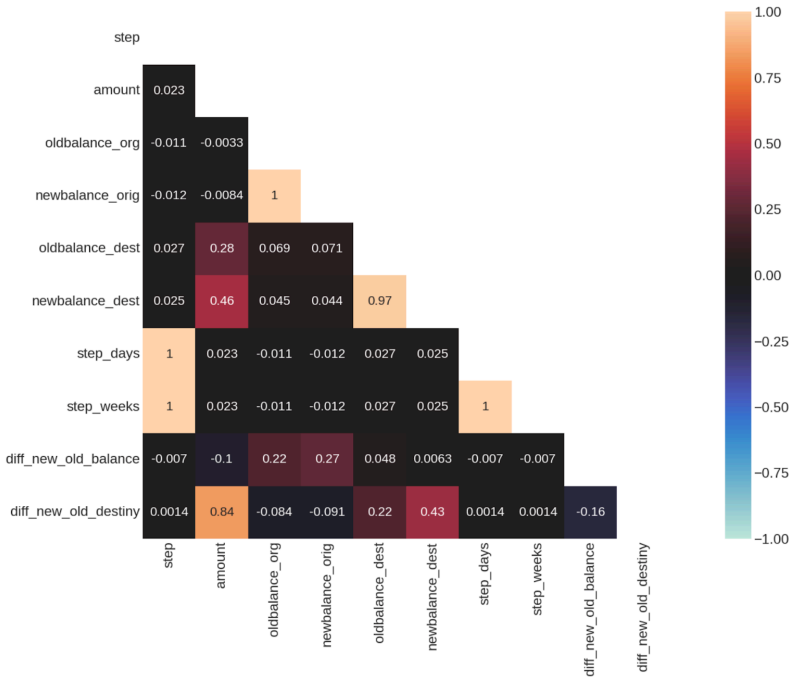


I analyzed the numerical variables, calculating additional metrics such as the range of values, coefficient of variation, skewness, and kurtosis, to gain a deep understanding of the data distribution and identify potential anomalies. The following observations were obtained, which will be used for further hypothesis formulation:

1. Every dataset exhibits a coefficient of variation exceeding 25%, indicating a lack of uniformity.
2. The range of the 'step' variable spans from 1 to 742 hours, covering a period of 30 days.
3. Several variables display pronounced peaks and exhibit a rightward skew.
4. Half of the 'newbalance_orig' values are zero, suggesting the possibility of transfers not reaching their intended destination.
5. The positive skewness is notably high, implying a concentration of values in a narrower range.



I represented the correlation matrix as a heatmap.



3.2 Description of the basic autoencoder model.

The basic model chosen is a neural autoencoder. The model's task is binary classification with an imbalanced number of class labels. The architecture of the model can be described as follows:

Encoder:

1. The first Dense layer (fully connected layer) with encoding_dim neurons. It uses hyperbolic tangent (tanh) as the activation function and applies L1 activity regularization with a coefficient of 10e-5.

2. The second Dense layer with int(encoding_dim / 2) neurons and ReLU activation function.

Decoder:

1. The first Dense layer with int(encoding_dim / 2) neurons and tanh activation function.

2. The second Dense layer with input_dim neurons (dimensionality of input data) and ReLU activation function.

Output Layer:

This layer returns the model's output data. Thus, the model has two hidden layers—one in the encoder and one in the decoder. The input layer takes data, and the output layer returns reconstructed data. The total number of layers in the model is four.

After the training process, only part of the autoencoder's encoder is retained to encode similar data types used in the training process. Various methods for constraining the network include:

1. Keeping hidden layers small: If the size of each hidden layer remains as small as possible, the network will be forced to select only representative features of the data, thereby encoding the data.

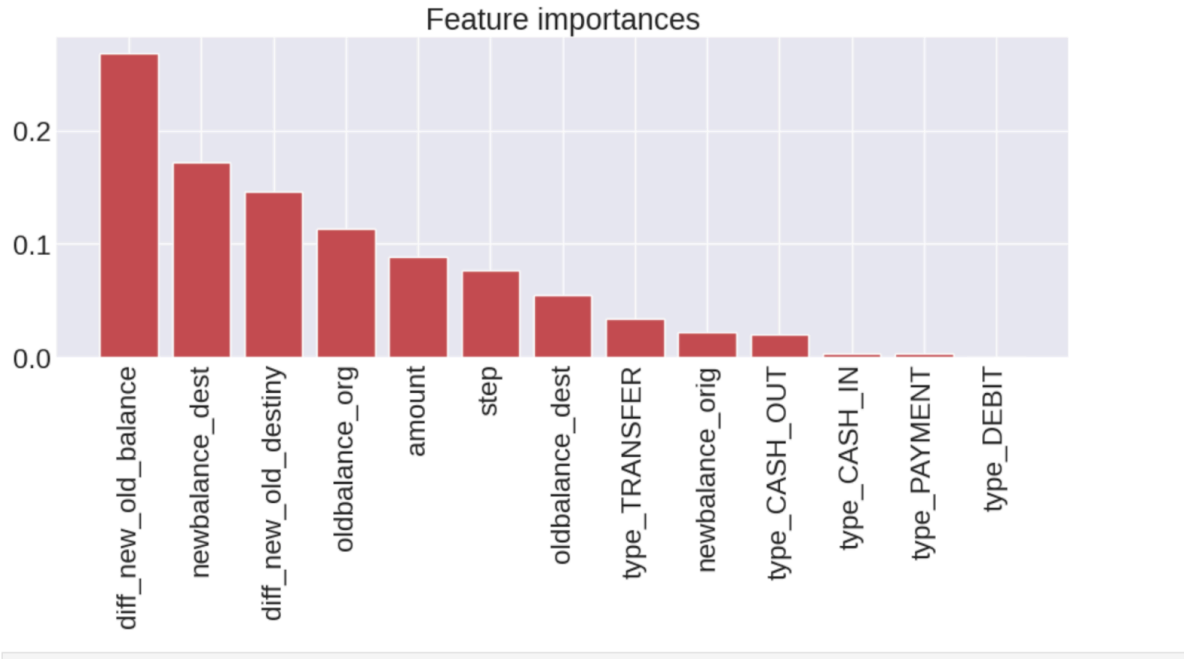
2. Regularization: In this method, a loss is added to the cost function that encourages the network to learn in ways other than copying the input data.

3. Noise removal: Another way to constrain the network is to add noise to the input data and train the network to remove noise from the data.

4. Adjusting activation functions: This method involves changing the activation functions of various nodes so that most nodes are in a resting state, effectively reducing the size of the hidden layers.

3.3 Feature Selection

A classifier based on random forest was created for training the model. Accordingly, a model was trained using the training data and corresponding class labels, then the importance score of each feature in the random forest model was calculated. The visualization of features by their importance level is shown in the figure below:



3.4 Data, Optimization Criterion, and Quality Metrics

The model was trained on a dataset of credit card transactions that occurred over two days, with 492 frauds out of 284,807 transactions.

The mean squared error was chosen as the loss function. Optimization was performed using Adam with a learning rate of 0.001. Model weights were initialized using a normal distribution.

The evaluation metrics chosen for assessing the model's quality were F1-score, recall, and ROC AUC score.

3.5 Hyperparameters

The following hyperparameters are defined in the autoencoder model:

- `encoding_dim`: The dimensionality of the hidden layer (latent space) that encodes the input data into a more compact representation. It is set to 14.

- `nb_epoch`: The number of training epochs, i.e., the number of times the entire dataset passes through the network during training. It is set to 100.

- `batch_size`: The number of data samples passed into the network for one training iteration. The batch size is set to 32.

- `Optimizer`: The optimization algorithm that determines how the network weights are updated during training. In this case, the 'adam' optimizer is used.

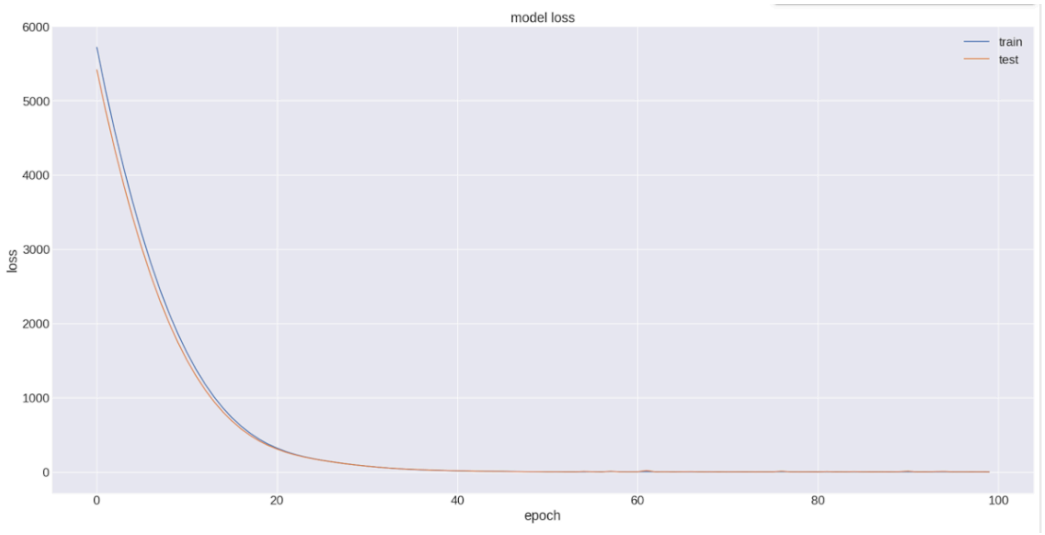
- `Loss`: The loss function that measures the discrepancy between the model's predictions and the true values during training. The 'mean_squared_error' loss function is used, which computes the mean squared error between predicted and true values.

3.6 Model Training

The autoencoder model was trained with pre-selected features ranked by importance. Below is the list of selected features represented as an array list:

final_columns_selected = ['step', 'oldbalance_org', 'newbalance_orig', 'newbalance_dest', 'diff_new_old_balance', 'diff_new_old_destiny', 'type_TRANSFER']

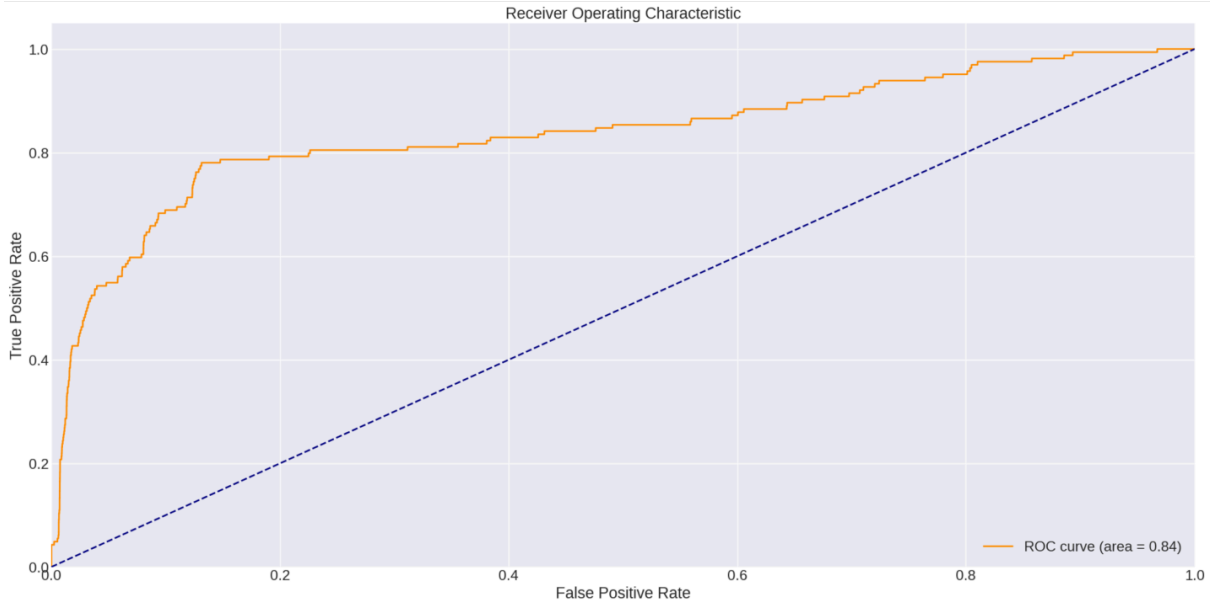
The training results are presented in the form of a graph constructed using TensorBoard.



ROC AUC (Receiver Operating Characteristic Area Under the Curve) is a metric for evaluating the quality of binary classification, measuring the area under the ROC curve. The ROC curve (Receiver Operating Characteristic) illustrates the relationship between the true positive rate and the false positive rate at various classification thresholds.

The ROC AUC metric provides an overall assessment of model performance, regardless of the specific classification threshold chosen. A higher ROC AUC value indicates better model performance: a value of 1 indicates a perfect model, while a value of 0.5 indicates random guessing.

Below is the result of computing the ROC AUC metric. It is evident that the area under the ROC curve is 0.84, indicating a significant level of accuracy.



Next, a classification report is presented, which includes accuracy evaluation metrics for unbalanced datasets by calculating the harmonic mean.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	0.00	0.00	0.00	164
accuracy			1.00	127253
macro avg	0.50	0.50	0.50	127253
weighted avg	1.00	1.00	1.00	127253

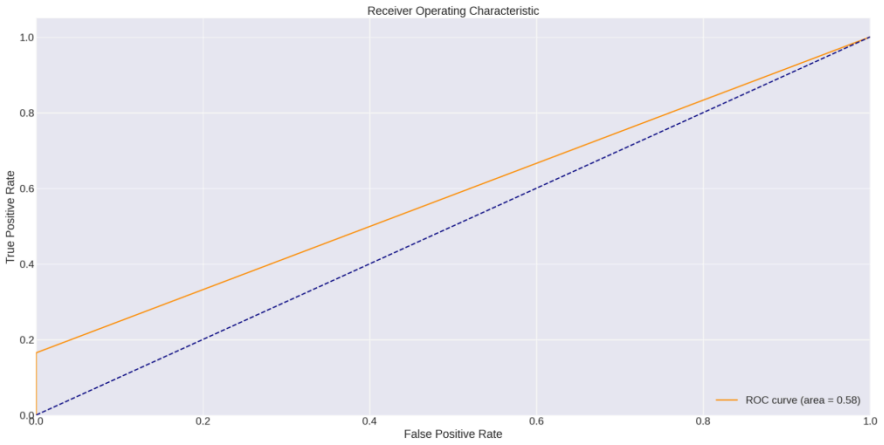
It is noteworthy that while all accuracy metrics for the first class (is_fraud=0) are at 100%, the result for the second class is almost zero.

3.7 Training of Other ML Models

3.7.1 Logistic Regression

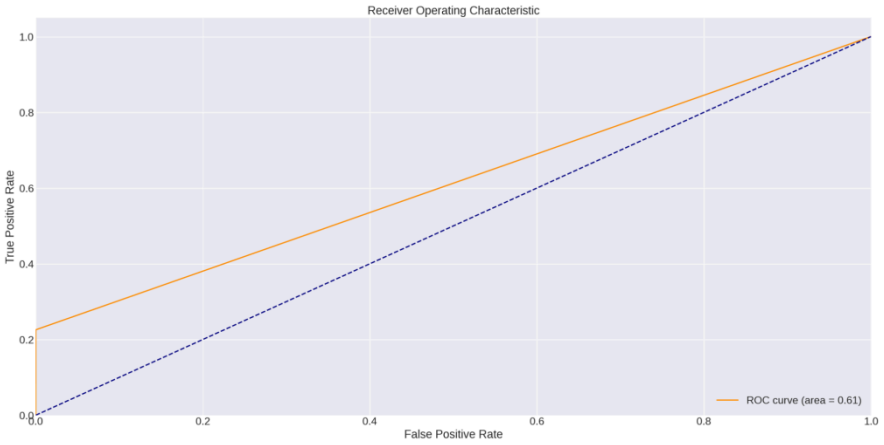
Below are the results based on Logistic Regression in the form of a classification report and ROC AUC curve.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	1.00	0.16	0.28	164
accuracy			1.00	127253
macro avg	1.00	0.58	0.64	127253
weighted avg	1.00	1.00	1.00	127253



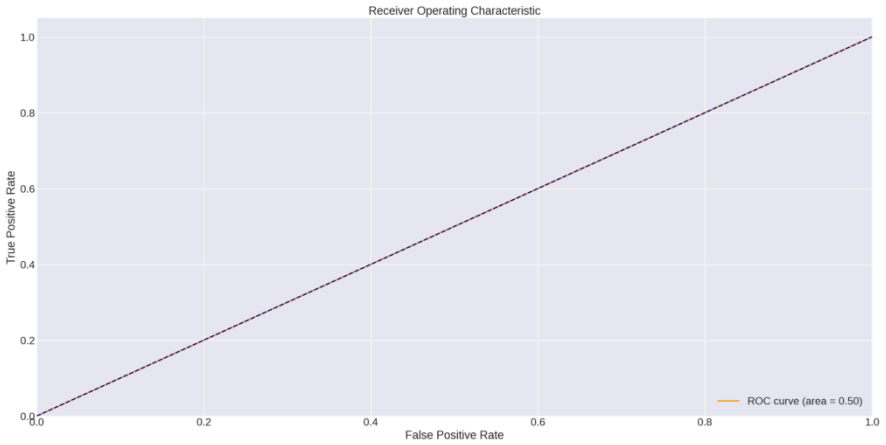
3.7.2 K-Nearest Neighbors

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	0.95	0.23	0.36	164
accuracy			1.00	127253
macro avg	0.97	0.61	0.68	127253
weighted avg	1.00	1.00	1.00	127253



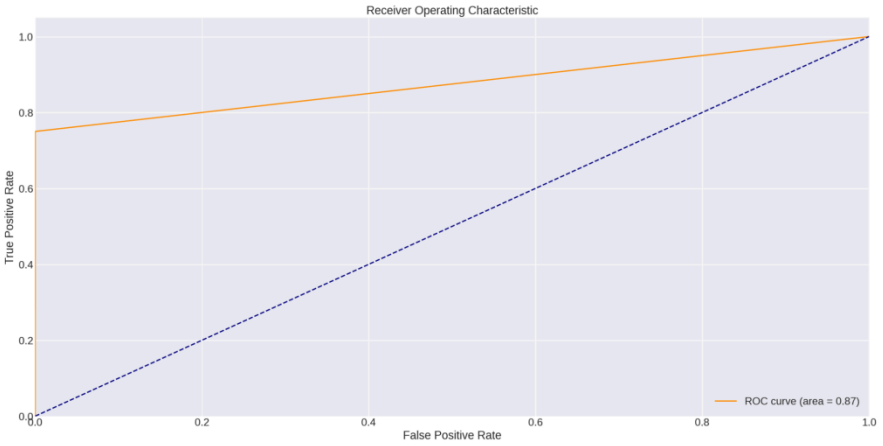
3.7.3 Support Vector Machine

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	0.00	0.00	0.00	164
accuracy			1.00	127253
macro avg	0.50	0.50	0.50	127253
weighted avg	1.00	1.00	1.00	127253



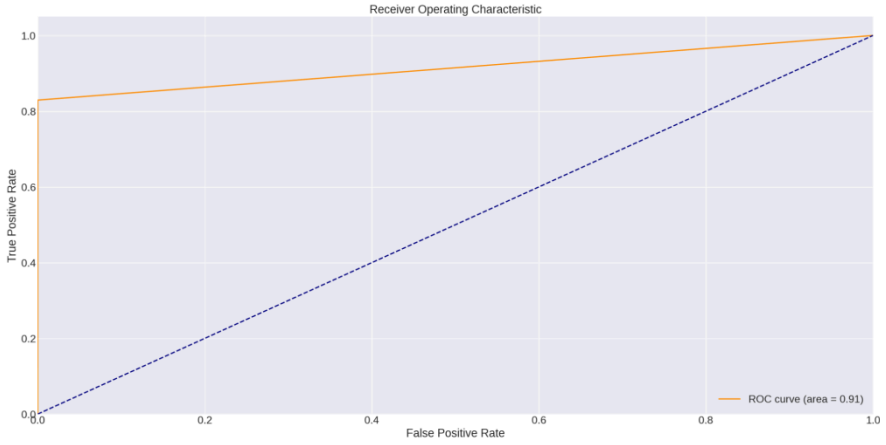
3.7.4 Random Forest

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	0.98	0.75	0.85	164
accuracy			1.00	127253
macro avg	0.99	0.87	0.93	127253
weighted avg	1.00	1.00	1.00	127253



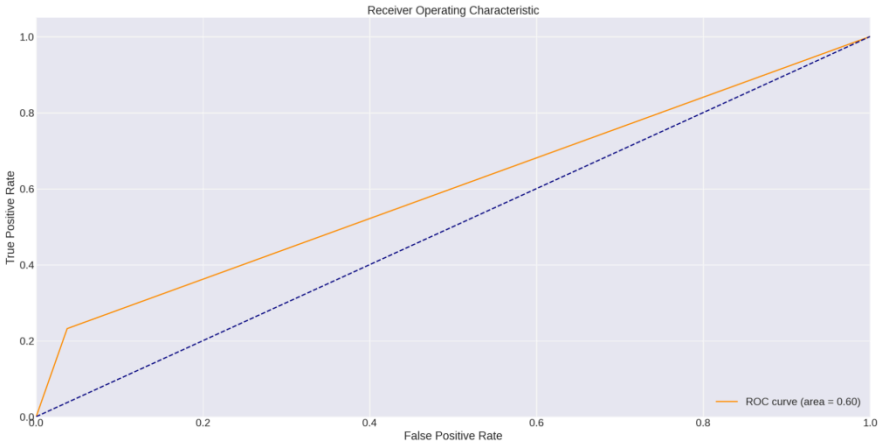
3.7.5 XGBoost

	precision	recall	f1-score	support
0	1.00	1.00	1.00	127089
1	0.96	0.83	0.89	164
accuracy			1.00	127253
macro avg	0.98	0.91	0.95	127253
weighted avg	1.00	1.00	1.00	127253



3.7.6 LightGBM

	precision	recall	f1-score	support
0	1.00	0.96	0.98	127089
1	0.01	0.23	0.02	164
accuracy			0.96	127253
macro avg	0.50	0.60	0.50	127253
weighted avg	1.00	0.96	0.98	127253



As a result of the comparative analysis, the XGBoost Classifier model turned out to be the best. Below are the results of finding the optimal hyperparameters of the model using GridSearchCV. Five cross-validation iterations were selected.

```
{
        'booster': 'gbtree',
        'eta': 0.3,
        'scale_pos_weight':
}
```

4. API

This API is designed to predict fraud in transaction data. Users can send requests with transaction details, and the API will provide predictions based on a pre-trained machine learning model.

4.1 Technology Stack Used:

This API was developed using the following technologies and tools:

1. Programming Language: Python
2. Web Framework: Flask
3. Data Processing: Pandas
4. Model Serialization: Joblib
5. API Testing and Interaction: Postman, cURL
6. Code Development and Testing: Jupyter Notebook, Visual Studio Code

4.2 Preparation for Development

First, we needed to save our best model to the local machine for future use when running the application. Next, all necessary information for sending a request was extracted, and the model's performance was tested on a single dataset entry.

9.0 API

Saving our best model to use it in the app:

```
import joblib
import os

path = os.path.expanduser('~/xgb_gs.joblib')

joblib.dump(xgb_gs, path)
```

```
['/Users/shklyarmike/xgb_gs.joblib']
```

Printing columns and test data that will be used in application:

```
print(X_train_cs.columns.tolist())
print(X_train_cs.head())
```

```
['step', 'oldbalance_org', 'newbalance_orig', 'newbalance_dest', 'diff_new_old_balance', 'diff_new_old_destiny', 'type_TRANSFER']
        step  oldbalance_org  newbalance_orig  newbalance_dest  diff_new_old_balance  diff_new_old_destiny  type_TRANSFER
418833   304        0.000013         0.000707         0.001815              0.881133              0.069490              0
535343   448        0.000000         0.000000         0.004738              0.878805              0.073839              0
419647   129        0.000000         0.000000         0.009856              0.878805              0.082316              1
358991    14        0.000000         0.000000         0.004747              0.878805              0.079537              0
186087   331        0.000000         0.000000         0.034297              0.878805              0.071922              1
```

Testing the model on a specific example:

```
test_data = pd.DataFrame([{
    "step": 304,
    "oldbalance_org": 0.000013,
    "newbalance_orig": 0.000070,
    "newbalance_dest": 0.001815,
    "diff_new_old_balance": 0.881133,
    "diff_new_old_destiny": 0.069490,
    "type_TRANSFER": 0
}], index=[0])

# Prediction
prediction = xgb_gs.predict(test_data)
print(prediction)
```

```
[0]
```

4.3 Writing the base version of the Application

The application was developed and run locally using the Flask web framework and VS Code:

```
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)

# Загрузка модели из файла
model = joblib.load('/Users/shklyarmike/xgb_gs.joblib')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)  # Получаем данные из POST запроса
    prediction = model.predict(data)  # Делаем предсказание
    return jsonify(prediction.tolist())  # Возвращаем предсказания в JSON формате

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')  # Запуск сервера на всех доступных IP адресах
```

The base version just uploaded the data from the request to the model and showed the result without any special UI.

4.4 Key Points of Application Operation

Base URL:

http://127.0.0.1:5000/

Endpoints:

POST /predict

Accepts transaction data and returns a fraud prediction.

Request Parameters:

None.

Request Body: The request body must contain a JSON object with the following fields:

1. step (int): The time step associated with the transaction.
2. oldbalance_org (float): The initial balance of the sender's account before the transaction.
3. newbalance_orig (float): The new balance of the sender's account after the transaction.
4. newbalance_dest (float): The new balance of the recipient's account after the transaction.
5. diff_new_old_balance (float): The difference between the new and old balance of the sender's account.
6. diff_new_old_destiny (float): The difference between the new and old balance of the recipient's account.
7. type_TRANSFER (int): Indicator of the transaction type (0 or 1, where 1 indicates a transfer).

Example Request Body:

```
{
 "step": 305,
 "oldbalance_org": 0.000015,
 "newbalance_orig": 0.000075,
 "newbalance_dest": 0.001825,
 "diff_new_old_balance": 0.881150,
 "diff_new_old_destiny": 0.069500,
 "type_TRANSFER": 1
}
```

Response Format: The API returns a JSON object with the prediction:

prediction (int): The model's prediction, where 1 indicates fraud and 0 indicates no fraud.

Example Response:

```
{
 "prediction": 1
}
```

Response Status Codes:

200 OK: The request was successfully processed.

400 Bad Request: There was an error in the request data. For example, a required field is missing or a field does not match the expected format.

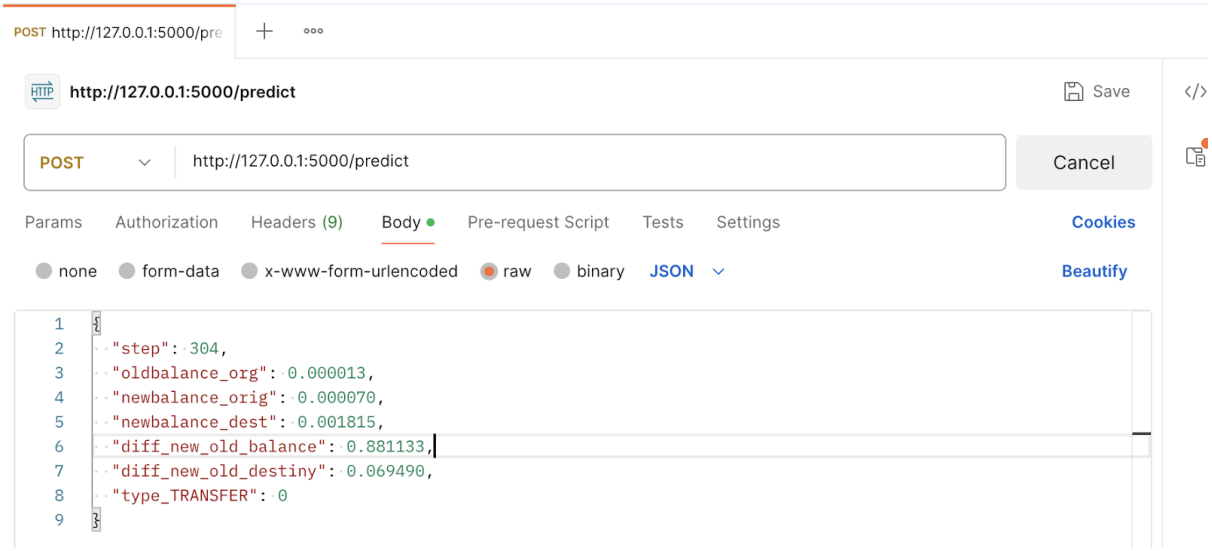
500 Internal Server Error: An internal server error occurred.

4.5 Testing the base version of the application

The initial version of the application could be used from the terminal with the CURL command:

curl -X POST http://127.0.0.1:5000/predict -H "Content-Type: application/json" -d '{"step": 305, "oldbalance_org": 0.000015, "newbalance_orig": 0.000075, "newbalance_dest": 0.001825, "diff_new_old_balance": 0.881150, "diff_new_old_destiny": 0.069500, "type_TRANSFER": 1}'

Or with third-party applications, such as Postman:



The request of our initial version was:

[0] - in case of clear transaction

[1] - in case of fraud transaction

4.6 Writing the final version of the application

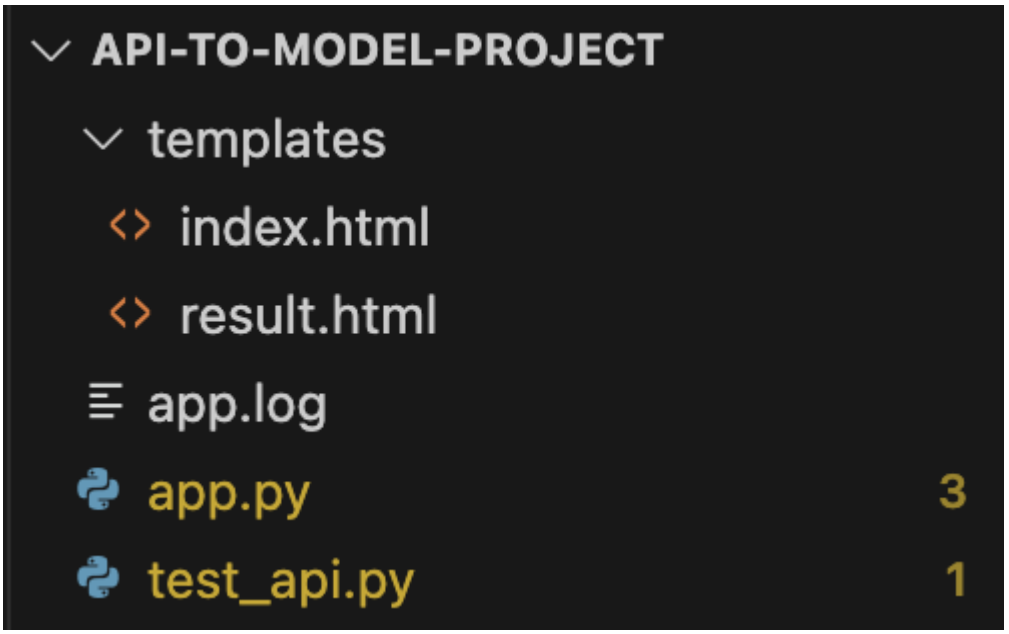
After testing the initial version of the application it was necessary to enhance it with .html files to make the UI look better. What is more, the exceptions were handled with special codes.

Here is the updated blocks of code of the application:

```python
@app.route('/', methods=['GET'])
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get data from form
        data = request.form.to_dict()
        data = {k: [float(v)] for k, v in data.items()}  # Transforming values to float
        data_frame = pd.DataFrame(data)
        # Prediction of the model
        prediction = model.predict(data_frame)
        result = 'Fraud' if prediction[0] == 1 else 'Not Fraud'
        # Turn the result back on the new page
        return render_template('result.html', prediction=result)
    except Exception as e:
        return str(e), 500
```

The GET request will work after the person opens the page in the browser and the POST request after the person pushes the “Predic” button.

Here is the structure of the final project:



4.7 Testing the final version of the application

Firstly, we prepare data in the notebook to fill the parameters required to the model and to find the examples of fraud/not fraud transactions:

Looking for data to test our application

```
[119]
fraud_indices = y_train[y_train == 1].index
NOT_fraud_indices = y_train[y_train == 0].index
```

```
[120]
fraud_transactions = X_train_cs.loc[fraud_indices]
NOT_fraud_transactions = X_train_cs.loc[NOT_fraud_indices]
```

```
[121]
print(fraud_transactions.head(3))
print(NOT_fraud_transactions.head(3))
```

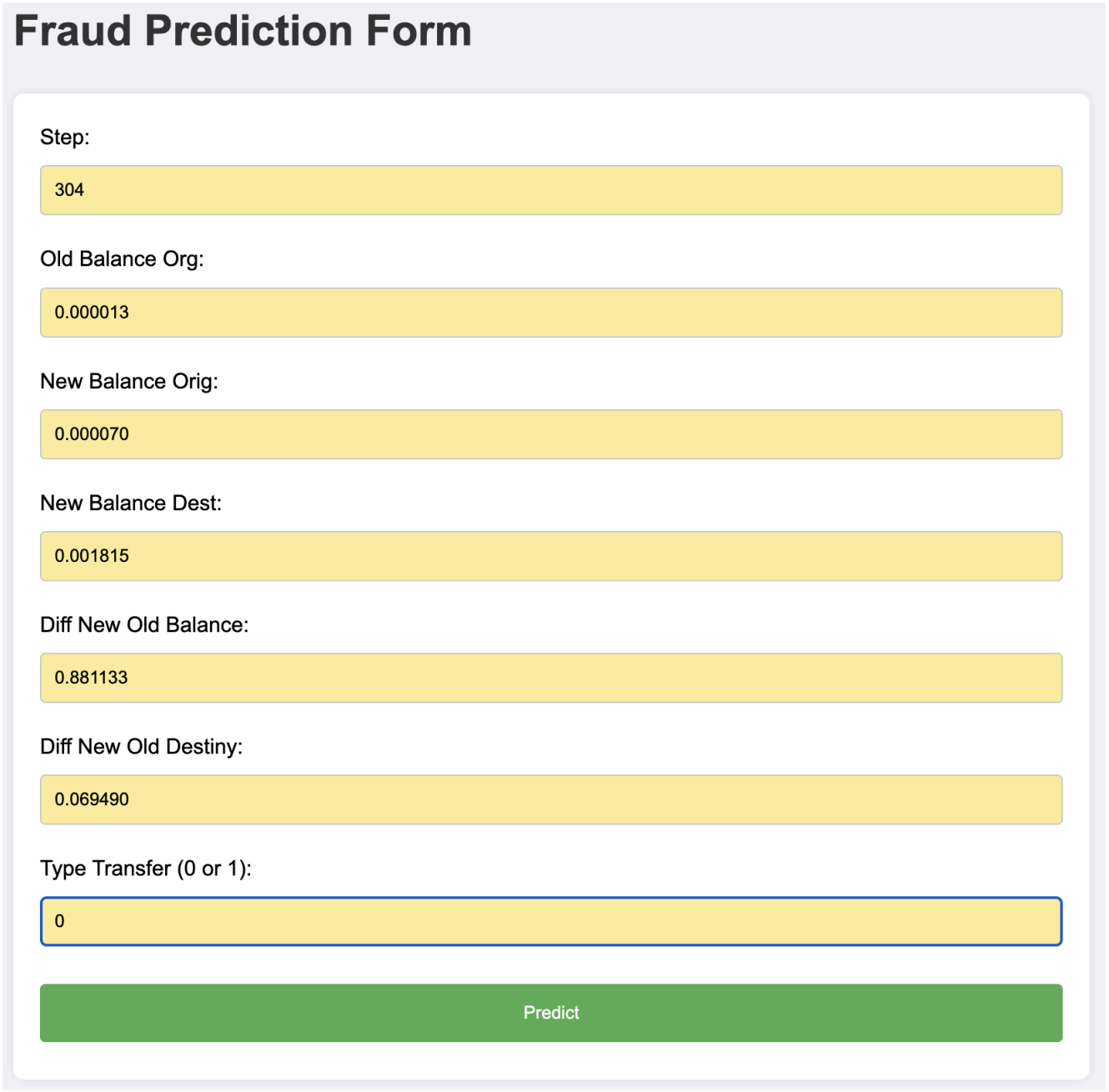
```
...          step  oldbalance_org  newbalance_orig  newbalance_dest  diff_new_old_balance  diff_new_old_destiny  type_TRANSFER
479963   157        0.024757              0.0         0.002644              0.796239              0.080338              0
504468   694        0.136386              0.0         0.000000              0.423946              0.069787              1
290034   688        0.025590              0.0         0.000000              0.793461              0.069787              1
         step  oldbalance_org  newbalance_orig  newbalance_dest  diff_new_old_balance  diff_new_old_destiny  type_TRANSFER
418833   304        0.000013         0.000707         0.001815              0.881133              0.069490              0
535343   448        0.000000         0.000000         0.004738              0.878805              0.073839              0
419647   129        0.000000         0.000000         0.009856              0.878805              0.082316              1
```

To test the application it is necessary to start it on the server (local computer):

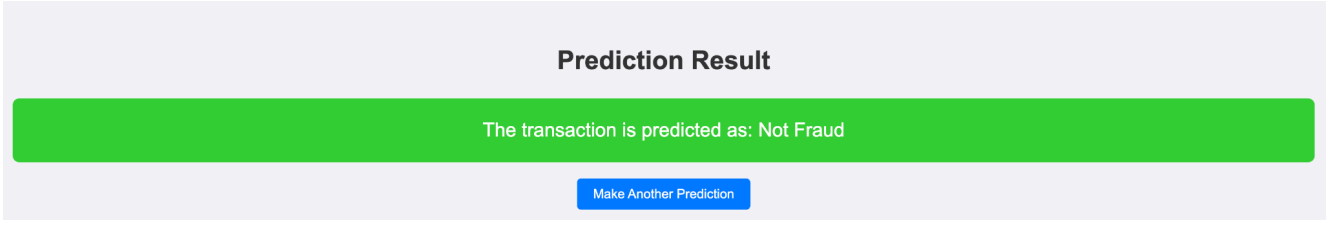
```
(virt_python) shklyarmike@shklyarmike-osx api-to-model-project % python app.py

 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.6:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 190-610-609
127.0.0.1 - - [02/Jun/2024 22:03:15] "POST /predict HTTP/1.1" 500 -
127.0.0.1 - - [02/Jun/2024 22:03:49] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:03:49] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [02/Jun/2024 22:03:58] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:04:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:06:52] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:07:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:08:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:09:16] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:11:19] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:11:19] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:11:32] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:13:31] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:15:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:15:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:15:31] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:15:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:18:28] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:18:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:24:58] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:25:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:28:17] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [02/Jun/2024 22:29:55] "GET / HTTP/1.1" 200 -
 * Detected change in '/Users/shklyarmike/api-to-model-project/test_api.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 190-610-609
```

After interacting with the web-site we see logs in our terminal that show us the time and type of requests. It helps us to control what is happening.

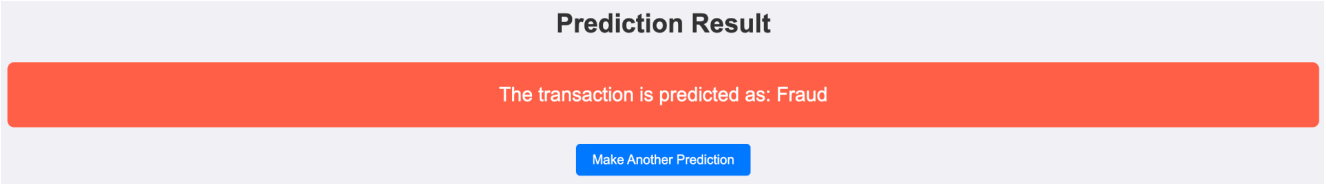
Here is the UI of the application:



We take data from the initial dataset and fill the fields. After that we push the button “predict” and receive the result with prediction:



Here is the output for the fraud transaction:



5. Conclusions

To sum up, the XGBoost model has demonstrated superior performance in detecting financial fraud, as evidenced by its metrics: precision for fraud cases was 0.96, recall 0.83, and the F1-score 0.89. These results underline the model's capability to identify fraudulent operations with high confidence, a critical feature for financial security.

Additionally, cross-validation results for XGBoost show the model's stability with a precision of 0.952, recall of 0.773, and an average F1-score of 0.853, indicating its robust generalization ability.

XGBoost has significantly outperformed other models, including autoencoders. The underperformance of autoencoders could potentially be attributed to:

1. Class Imbalance: Autoencoders are not ideally suited for heavily imbalanced data as they can overfit to the more frequent class, in this case, the non-fraudulent transactions.

2. Anomaly Detection Complexity: Autoencoders are designed for input data reconstruction, so their ability to detect anomalies depends on how well they can distinguish between 'normal' and abnormal data. In financial transactions, anomalies might not be very distinct or too varied, complicating their detection.

It's also crucial to note that the developed API facilitates the integration of the XGBoost model into banking systems for real-time transaction monitoring. This is especially valuable for providing prompt responses to potential threats and preventing fraud. The API delivers precise predictions based on transaction data, making it a useful tool for enhancing banking security.

The use of technologies such as the Flask web framework and Pandas for data processing, along with tools like Postman for testing and interacting with the API, ensures flexibility and efficiency in application development and testing. These tools help accelerate development and reduce the risk of errors when implementing the system in real operations.

List of Sources

1. Kathrin Melcher, Rosaria Silipo: Fraud detection using Random Forest algorithms, Neural Autoencoder, and Isolation Forest. (Date of request: 17.10.2023).
2. Vikas R. Shetty,Pooja R. andRashmi Laxmikant Malghan: Safeguarding against Cyber Threats: Machine Learning-Based Approaches for Real-Time Fraud Detection and Prevention. (Date of request: 29.10.2023).
3. LLC "Cybertonica": Anti-fraud as a business growth driver. (Date of request: 11.11.2023).
4. Evgeny Shadrin: ML and AI for banks and businesses. The experience of Cybertonica. (Date of request: 30.11.2023).
5. Mordor Intelligence: Market Size and Share Analysis of Fraud Detection and Prevention – Growth Trends and Forecasts (2024–2029). (Date of request: 13.12.2023).
6. IABAC GLOBAL: Fraud Detection through Data Analytics: Identifying Anomalies and Patterns. (Date of request: 17.01.2024).
7. Salim Hasham, Rob Hayden, and Rob Wavra: Combating payments fraud and enhancing customer experience. (Date of request: 23.02.2024).
8. Cyber Media: Comparison of Russian Anti-Fraud Systems: Verification, Analytics, and Blocking Mechanisms. (Date of request: 25.02.2024).
9. Brett Slatkin: Effective Python: 59 Ways to Write Better Python. (Date of request: 28.03.2024).
10. Armin Ronacher: Flask documentation. (Date of request: 19.04.2024).