ДОДАТОК А

Перелік джерел посилання науковими напрямами керівника та науковців кафедри Програмної інженерії

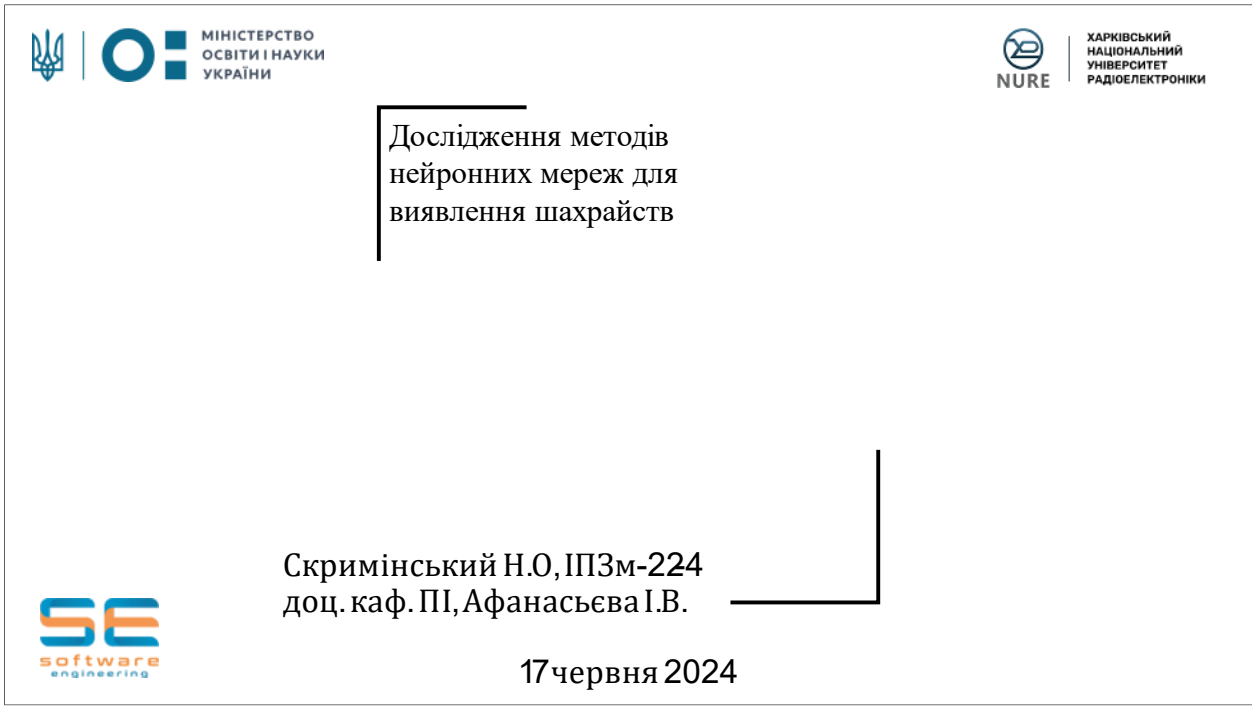
12. Afanasieva, I., Golian, N., Golian, V., Khovrat, A., & Onyshchenko, K. (2023). Application of Neural Networks to Identify of Fake News. CEUR Workshop Proceedings, 3396, 346-358.

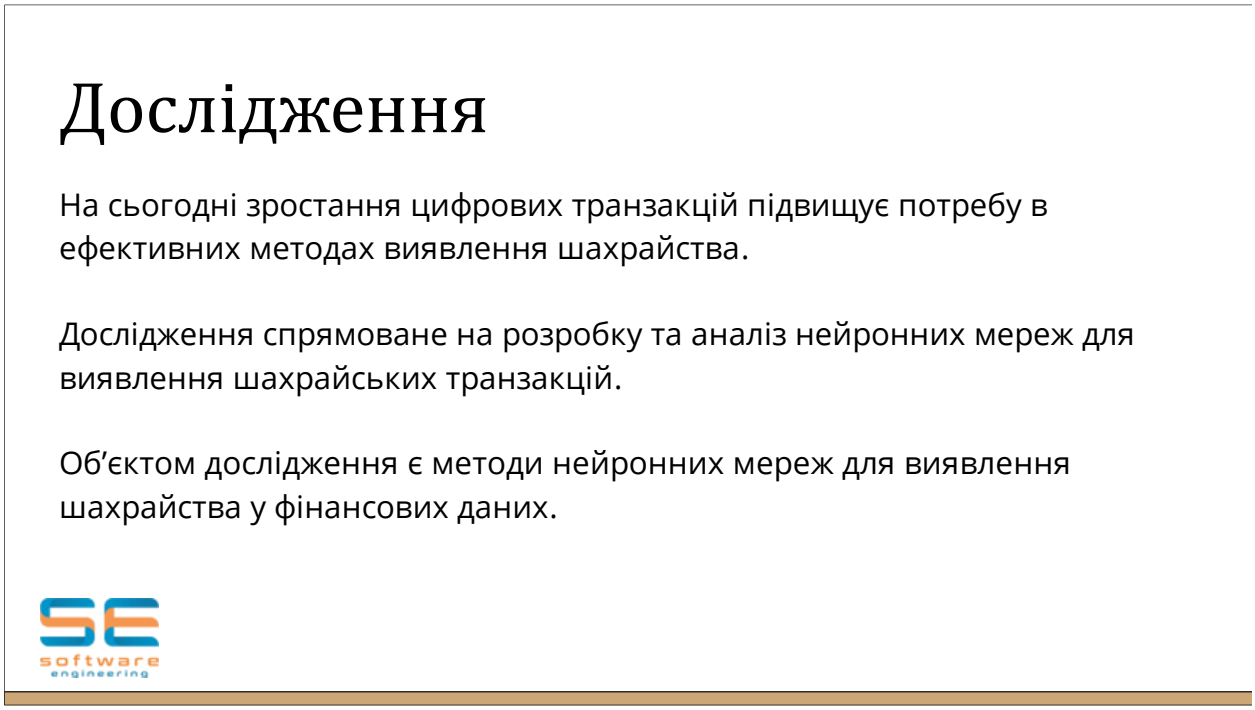
13. Afanasieva I.V., et al. Neural network approach for emotional recognition in text / I.V. Afanasieva, D.S. Nazarenko, N.V. Golian // Біоніка інтелекту, Харків: ХНУРЕ, 2019. – 1(92). – С. 9-14.

19. Nazarenko, D., Afanasieva, I., Golian, N., & Golian, V. (2021). Investigation of the deep learning approaches to classify emotions in texts. CEUR Workshop Proceedings, 2870, 206-224.

ДОДАТОК Б

Слайди презентації





Огляд літератури (аналогів)

Для проведення загального аналізу предметної області було використано наступні джерела:

– Seventh report on card fraud;

– Credit card fraud.

Для проведення дослідження методів використано наступні джерела:

– Harris Hawks optimization;

– Sagemaker-graph-fraud-detection.

Постановка задачі

- розглянути і проаналізувати існуючі методи виявлення шахрайста за допомогою нейроних мереж;
- визначити переваги і обмеження нейромереж;
- обрати тип та конфігурацію нейронної мережі, яка найбільш підходить для вирішення поставленої задачі;
- відібрати та підготувати набор даних для навчання та тестування;
- навчання нейронної мережі на підготовленому наборі даних для розпізнавання шаблонів, пов'язаних з шахрайством;
- аналіз результатів виявлення шахрайства, включаючи точність, чутливість, специфічність та інші метрики, що відображають ефективність моделі;
- порівняти з іншими рішеннями.

Аналіз методів

•ANN з Adam
•ANN з HOA
•GNN з DGL

•NumPy: для роботи з багатовимірними масивами та виконання математичних операцій.
•Pandas: для обробки та аналізу даних.
•Matplotlib: для візуалізації даних та результатів аналізу.
•Scikit-learn: для попередньої обробки даних, реалізації методів машинного навчання та оцінки моделей.
•TensorFlow та Keras: для створення, навчання та оцінки нейронних мереж.
•PyTorch: для розробки та тестування нейронних мереж, зокрема для обчислень на графічних процесорах.

Опис програмного забезпечення, що було використано у дослідженні

Першим кроком стало збирання та підготовка даних. Потім побудова нейромережі та налаштування моделі.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(543, 256)
        self.fc2 = nn.Linear(256,128)
        self.fc3 = nn.Linear(128,64)
        self.fc4 = nn.Linear(64,1)

        self.dropout = nn.Dropout(p= 0.6, inplace = True)

    def forward(self, x):
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.dropout(F.relu(self.fc3(x)))
        x = F.sigmoid(self.fc4(x))
        return x
```

```
criterion = BCELoss()
optimizer = Adam(net.parameters())
```

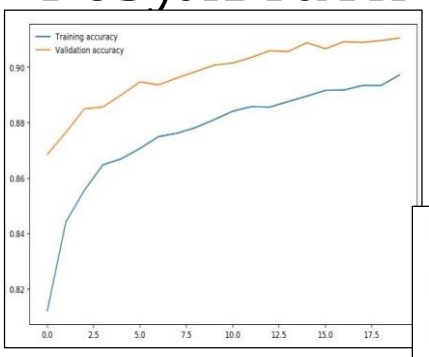
Реалізація стратегії хижих птахів

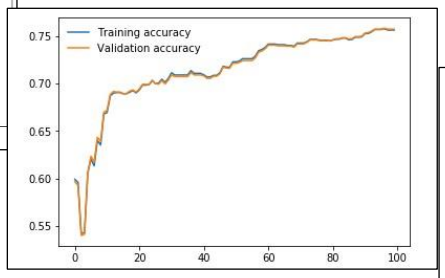
```python
def update_solution(self, solution, best_solution):
    r1, r2, r3 = np.random.rand(3)
    q = 2 * r3 - 1
    q_tensor = torch.tensor(q, dtype=torch.float).to(device)
    d = torch.abs(best_solution - solution)
    new_solution = best_solution - r1 * d * torch.sign(q_tensor)

    # Adding more variability by introducing noise
    noise = torch.randn_like(new_solution) * 0.01
    new_solution += noise

    return new_solution
```

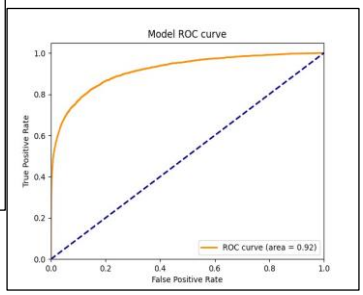
Зміст проведеного експерименту

- Вибір та підготовка набору даних (Pandas, NumPy, Matplotlib та Scikit-learn).
- Створення нейронної мережі (PyTorch).
- Тренування моделі (Adam, HOA).
- Оцінка та валідація моделі.

Результати експерименту







Аналіз отриманих результатів

Враховуючи, що дані дуже незбалансовані, нам потрібно знайти компроміс між Recall і Precision. Враховуючи, що помилкова класифікація нешахрайських транзакцій як шахрайських серйозно вплине на досвід користувачів, пріоритетом є точність. Тому точність для кожного методу є:

- Прототип з Adam: забезпечує високу точність як на навчальних, так і на валідаційних даних (приблизно 0.90);
- GNN з DGL: відмінно працює для задач виявлення шахрайства з високою точністю (0.92);
- Прототип з HOA: показує стабільне зростання точності, досягаючи близько 0.75 як на навчальних, так і на валідаційних даних.

Висновки

- Проведено огляд та аналіз існуючих методів виявлення шахрайства за допомогою нейронних мереж.
- Обрано тип та конфігурацію нейронної мережі, яка найбільше підходить для задачі виявлення шахрайства.
- Відбір та підготовка набору даних для навчання та тестування моделі.
- Нейронну мережу навчено розпізнавати шахрайські патерни на підготовленому наборі даних.
- Проведено аналіз результатів виявлення шахрайства, включаючи метрики точності.
- Проведено порівняння з іншими методами виявлення шахрайства.

Публікація результатів

Роботу було представлено на IX Міжнародна науково-технічна конференція «Поліграфічні, мультимедійні та web-технології». Збірник знаходиться у друці

33. ДОСЛІДЖЕННЯ МЕТОДІВ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИЯВЛЕННЯ ШАХРАЙСТВА.
Афанасьєва І.В., Скримінський Н.О.

Підсумки

На основі цих результатів можна зробити висновок, що GNN з DGL є найбільш перспективною моделлю для цього конкретного завдання, оскільки вона досягає найвищої точності.

Таким чином, якщо важлива висока точність і ресурсні обмеження не є критичними, GNN з DGL буде найкращим вибором. Adam також показує високу точність і швидке сходження, підходить для багатьох загальних задач. HOA може бути корисним у випадках, коли потрібна стабільність і уникнення перенавчання, хоча його точність трохи нижча

Дякую за увагу!

ДОДАТОК В

Тези доповіді для конференції

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки (Україна)
ДНУ «Книжкова палата України ім. Івана Федорова» (Україна)
Громадська спілка «Українська асоціація видавців і книгорозповсюджувачів» (Україна)
Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського» (Україна)
Українська академія друкарства (Україна)
Варшавська політехніка (Польща)
Університет штату Гуанахуато (Мексика)
Ташкентський інститут текстильної та легкої промисловості (Узбекистан)

PRINT MULTIMEDIA & WEB

IX Міжнародна науково-технічна конференція
«Поліграфічні, мультимедійні та web-технології»
14-18 травня 2024 року

ПРОГРАМА

Конференція буде проходити за адресою: м. Харків, пр. Науки, 14.
(в дистанційному режимі)
Пленарне онлайн-засідання – 14 травня 2024 року в 9-30.
Онлайн-засідання секцій – 15, 16 травня 2023 року в 11-00.

Телефони для довідок:
Заступник голови Оргкомітету: проф. Дейнеко Жанна Валентинівна,
тел. +38(099) 492-18-49
Секретарі Оргкомітету:
ст. викл. Чеботарьова Ірина Борисівна, доц. Вовк О.В.
тел.+38(068) 884-85-62
Адреса електронної пошти: pmw@nure.ua
Сайт конференції: pmw.nure.ua

Регламент доповідей:
на пленарному засіданні – *до 20 хвилин*;
на секційних засіданнях – *10-15 хвилин*;
молодіжна школа-семінар – *6-8 хвили.*

32. АЛГОРИТМИ ЛОКАЛІЗАЦІЇ ТРАЄКТОРІЇ ТРАНСПОРТНИХ ЗАСОБІВ У ВІДЕОПОТОКІ. **Супрун О.О., Котельніков І.В.**
33. ДОСЛІДЖЕННЯ МЕТОДІВ НЕЙРОННИХ МЕРЕЖ ДЛЯ ВИЯВЛЕННЯ ШАХРАЙСТВА. **Афанасьєва І.В., Скримінський Н.О.**
34. ПОБУДОВА ER-МОДЕЛІ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНОЇ ТЕХНОЛОГІЇ УПРАВЛІННЯ ЕЛЕКТРОННИМИ НАУКОВИМИ ВИДАННЯМИ. **Нерода Т.В., Мороз Р.Б.**
35. РОЗРОБКА ПІДСИСТЕМИ КАЛІБРУВАННЯ ТЕНЗОСЕНСОРА ДЛЯ СМАРТ-КОНТЕЙНЕРА НАКОПИЧЕННЯ СУБСТРАТНИХ ОБРІЗКІВ. **Нерода Т.В., Сторожук Д.**
36. ОСОБЛИВОСТІ ОБРОБКИ АСТРОНОМІЧНОГО ВІДЕО. **Хламов С.В., Плігунова Е.В.**
37. ПЕРЕТВОРЕННЯ 3D-МОДЕЛЕЙ ПРОГРАМАМИ-СЛАЙСЕРАМИ ЯК КЛЮЧОВИЙ ФАКТОР ДОСЯГНЕННЯ ЯКОСТІ АДИТИВНОГО ДРУКУ. **Володько М.Ю.**
38. АДАПТАЦІЯ ПОЯСНЕНЬ З УРАХУВАННЯ ДІЙ КОРИСТУВАЧА НА ОСНОВІ АНАЛІЗУ ЖУРНАЛІВ ПОДІЙ МЕТОДАМИ PROCESS MINING. **Чалий С.Ф., Єрохін Д.О.**

ДОДАТОК Г

Фрагменти коду

Файл HOA.py

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

import warnings
warnings.filterwarnings("ignore")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
print('SETUP COMPLETED')

column_types = {'TransactionID': 'int32', 'isFraud': 'int8', 'TransactionDT':
'int32', 'TransactionAmt': 'float32', 'ProductCD': 'object', 'card1': 'int16',
'card2': 'float32', 'card3': 'float32', 'card4': 'object', 'card5': 'float32',
'card6': 'object', 'addr1': 'float32', 'addr2': 'float32', 'dist1': 'float32',
'dist2': 'float32', 'P_emaildomain': 'object', 'R_emaildomain': 'object', 'C1':
'float32', 'C2': 'float32', 'C3': 'float32', 'C4': 'float32', 'C5': 'float32',
'C6': 'float32', 'C7': 'float32', 'C8': 'float32'}

train_df = pd.read_csv('../input/ieee-fraud-detection/train_transaction.csv',
dtype=column_types)

train_y = train_df['isFraud']
train_df.drop('isFraud', axis=1, inplace=True)

# Select categorical columns
categorical_cols = [cname for cname in train_df.columns if train_df[cname].dtype
== "object"]

# Select numerical columns
numerical_cols = [cname for cname in train_df.columns if train_df[cname].dtype in
['int8', 'int16', 'int32', 'float32']]

# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[('imputer',
SimpleImputer(strategy='constant')), ('scale', StandardScaler())])

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[('imputer',
SimpleImputer(strategy='constant')), ('onehot', OneHotEncoder(dtype=np.int8,
handle_unknown='ignore'))])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_cols),
    ('cat', categorical_transformer, categorical_cols)
```

```
])

train_df = preprocessor.fit_transform(train_df)
train_y = train_y.values

# Convert csr_matrix to np.ndarray
train_df = train_df.toarray()
print("Shape of transformed train data:", train_df.shape)

train_tmp, valid_tmp, y_train_tmp, y_valid_tmp = train_test_split(train_df,
train_y, stratify=train_y)

train_tmp = torch.from_numpy(train_tmp).type(torch.float).to(device)
y_train_tmp = torch.from_numpy(y_train_tmp).type(torch.float).to(device).view(-1,
1)
valid_tmp = torch.from_numpy(valid_tmp).type(torch.float).to(device)
y_valid_tmp = torch.from_numpy(y_valid_tmp).type(torch.float).to(device).view(-1,
1)

train_loader = DataLoader(TensorDataset(train_tmp, y_train_tmp), batch_size=64,
shuffle=True)  # Reduced batch size
valid_loader = DataLoader(TensorDataset(valid_tmp, y_valid_tmp), batch_size=64,
shuffle=False)

# Adjust input size of the network based on the transformed data shape
input_size = train_df.shape[1]

class Net(nn.Module):
    def __init__(self, input_size):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 1)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.dropout(F.leaky_relu(self.fc1(x)))  # Using Leaky ReLU
        x = self.dropout(F.leaky_relu(self.fc2(x)))
        x = self.dropout(F.leaky_relu(self.fc3(x)))
        x = torch.sigmoid(self.fc4(x))
        return x

class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a given
patience."""
    def __init__(self, patience=10, verbose=False, delta=0, path='checkpoint.pt',
trace_func=print):
        self.patience = patience
        self.verbose = verbose
        self.counter = 0
        self.best_score = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.delta = delta
        self.path = path
        self.trace_func = trace_func

    def __call__(self, val_loss, model):
        score = val_loss
        if self.best_score is None:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
```

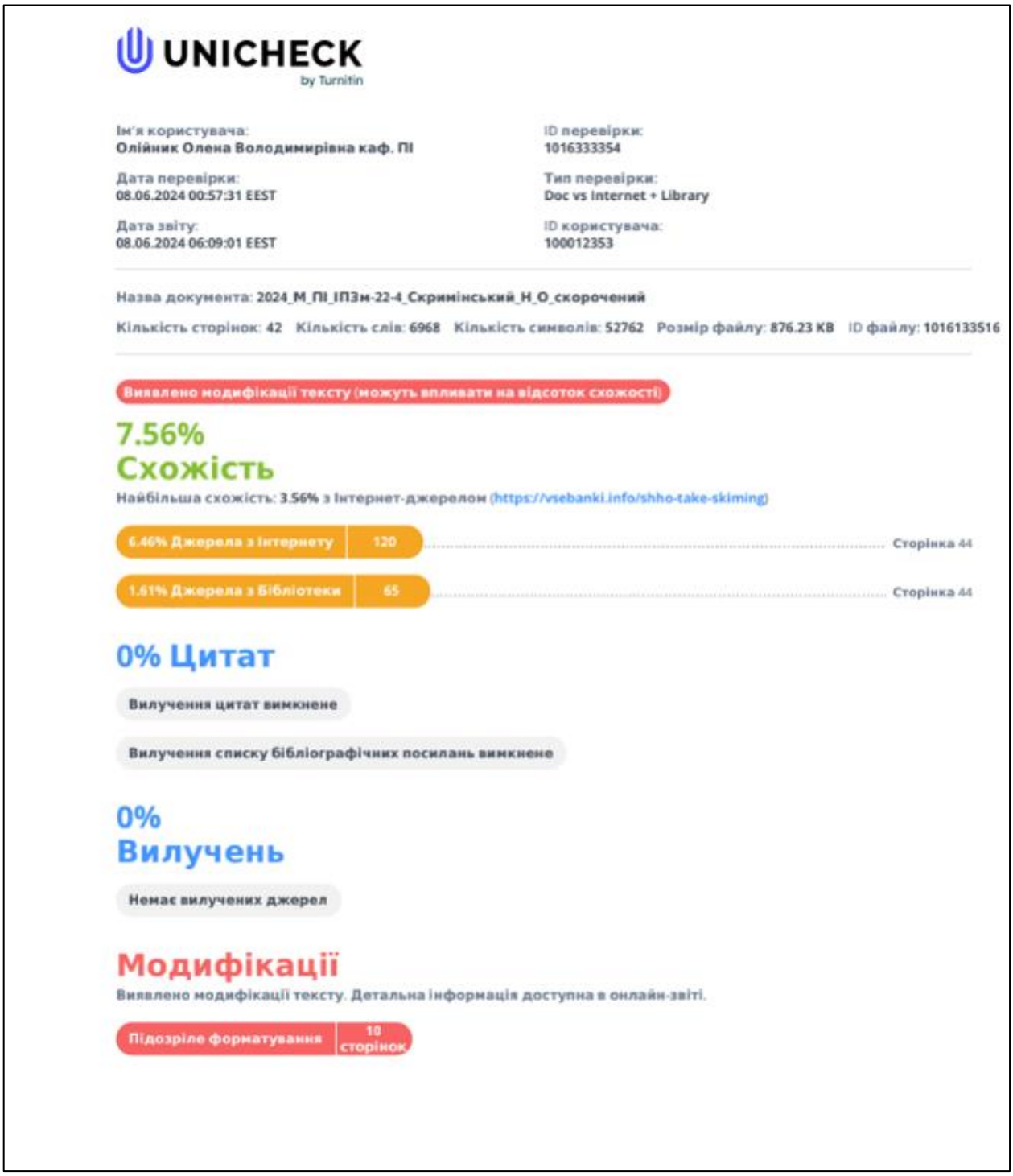
```
        elif score < self.best_score + self.delta:
            self.counter += 1
            self.trace_func(f'EarlyStopping counter: {self.counter} out of
{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        if self.verbose:
            self.trace_func(f'Validation loss decreased ({self.val_loss_min:.6f} -
-> {val_loss:.6f}).  Saving model ...')
        torch.save(model.state_dict(), self.path)
        self.val_loss_min = val_loss

class HawkOptimizer:
    def __init__(self, net, pop_size=30, max_iter=100):  # Increased pop_size and
max_iter
        self.net = net
        self.pop_size = pop_size
        self.max_iter = max_iter
        self.dim = sum(p.numel() for p in net.parameters())
        self.pop = [self.initialize_solution() for _ in range(pop_size)]
        self.best_solution = None
        self.best_fitness = float('inf')
        self.train_auroc_list = []
        self.valid_auroc_list = []

    def initialize_solution(self):
        solution = []
        for p in self.net.parameters():
            solution.append(p.data.clone().view(-1) +
torch.randn_like(p.data.clone().view(-1)) * 0.1)  # Adding randomness
        return torch.cat(solution).to(device)

    def evaluate_fitness(self, solution, loader, criterion):
        idx = 0
        for p in self.net.parameters():
            p.data = solution[idx:idx + p.numel()].view(p.size()).clone()
            idx += p.numel()
        self.net.eval()
        loss = 0
        true_labels = []
        pred_labels = []
        with torch.no_grad():
            for data, target in loader:
                output = self.net(data)
                loss += criterion(output, target).item()
                true_labels.extend(target.cpu().numpy())
                pred_labels.extend(output.cpu().numpy())
        roc_auc = roc_auc_score(true_labels, pred_labels)
        return loss / len(loader), roc_auc

    def update_solution(self, solution, best_solution):
        r1, r2, r3 = np.random.rand(3)
        q = 2 * r3 - 1
        q_tensor = torch.tensor(q, dtype=torch.float).to(device)
        d = torch.abs(best_solution - solution)
        new_solution = best_solution - r1 * d * torch.sign(q_tensor)
```

```
        # Adding more variability by introducing noise
        noise = torch.randn_like(new_solution) * 0.01
        new_solution += noise

        return new_solution

    def optimize(self, train_loader, valid_loader, criterion, patience=10):
        early_stopping = EarlyStopping(patience=patience, verbose=True)
        for iter in range(self.max_iter):
            fitnesses = [self.evaluate_fitness(sol, valid_loader, criterion) for
sol in self.pop]
            losses, roc_aucs = zip(*fitnesses)
            best_idx = np.argmin(losses)
            if losses[best_idx] < self.best_fitness:
                self.best_fitness = losses[best_idx]
                self.best_solution = self.pop[best_idx].clone()

            new_pop = []
            for sol in self.pop:
                new_sol = self.update_solution(sol, self.best_solution)
                new_pop.append(new_sol)
            self.pop = new_pop

            train_loss, train_roc_auc = self.evaluate_fitness(self.best_solution,
train_loader, criterion)
            valid_loss, valid_roc_auc = self.evaluate_fitness(self.best_solution,
valid_loader, criterion)

            self.train_auroc_list.append(train_roc_auc)
            self.valid_auroc_list.append(valid_roc_auc)

            print(f"Iteration {iter + 1}, Train ROC AUC: {train_roc_auc}, Valid
ROC AUC: {valid_roc_auc}, Best Fitness: {self.best_fitness}")

            early_stopping(valid_roc_auc, self.net)
            if early_stopping.early_stop:
                print("Early stopping")
                break

        idx = 0
        for p in self.net.parameters():
            p.data = self.best_solution[idx:idx +
p.numel()].view(p.size()).clone()
            idx += p.numel()

input_size = train_df.shape[1]
net = Net(input_size).to(device)

# Use weighted BCE loss to address class imbalance
pos_weight = torch.tensor([len(y_train_tmp) / sum(y_train_tmp)]).to(device)
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

# Hawk Optimizer
optimizer = HawkOptimizer(net, pop_size=30, max_iter=100)

# Train the network using HOA
optimizer.optimize(train_loader, valid_loader, criterion)

# Evaluate and plot results
import matplotlib.pyplot as plt

plt.plot(optimizer.train_auroc_list, label='Training accuracy')
plt.plot(optimizer.valid_auroc_list, label='Validation accuracy')
```

```
plt.legend(frameon=False)
plt.show()

# Make predictions on test set
submit_df = pd.read_csv('../input/ieee-fraud-detection/test_transaction.csv',
dtype=column_types)
sub = pd.read_csv('../input/ieee-fraud-detection/sample_submission.csv')

submit_df = preprocessor.transform(submit_df)
submit_df = submit_df.toarray()
submit_tmp = torch.from_numpy(submit_df).type(torch.float).to(device)
fake_labels = torch.zeros(submit_tmp.size(0)).type(torch.float).to(device)
submit_loader = DataLoader(TensorDataset(submit_tmp, fake_labels), batch_size=64,
shuffle=False)

submission = []
with torch.no_grad():
    net.eval()
    for samples, _ in submit_loader:
        log_ps = net(samples)
        submission.extend(log_ps.cpu().numpy())

sub['isFraud'] = submission
sub.to_csv('submission.csv', index=False)
```

ДОДАТОК Д

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016333354

Дата перевірки:
08.06.2024 00:57:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 06:09:01 EEST

ID користувача:
100012353

Назва документа: 2024_М_ПІ_ІПЗм-22-4_Скримінський_Н_О_скорочений

Кількість сторінок: 42 Кількість слів: 6968 Кількість символів: 52762 Розмір файлу: 876.23 KB ID файлу: 1016133516

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.56%
Схожість

Найбільша схожість: 3.56% з Інтернет-джерелом (https://vsebanki.info/shho-take-skiming)

6.46% Джерела з Інтернету 120 ..... Сторінка 44

1.61% Джерела з Бібліотеки 65 ..... Сторінка 44

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 10 сторінок

ДОДАТОК Е

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-4 (група)

Скримінський Нікіта Олександрович

(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

11.06.2024