POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Projekt z przedmiotu: ”Bezpieczeństwo sieci komputerowych I”

Temat:
„Możliwości wykorzystania generatywnych modeli AI do detekcji anomalii ataków”

Wykonał:

Imię i nazwisko: Michał Maciej

Nr albumu: 167814

Rok studiów: 4

Grupa projektowa: 2

Spis treści

1. Wstęp

Współczesne środowiska informatyczne stają w obliczu coraz większych wyzwań związanych z bezpieczeństwem sieciowym. Ataki hakerskie, w tym phishing, ransomware, czy ataki DDoS, stanowią nieustanny problem dla organizacji i instytucji na całym świecie. Pomimo zastosowania wszelakich metod zabezpieczeń, coraz bardziej zaawansowane techniki ataków sprawiają , że tradycyjne metody detekcji zaczynają być niewystarczające.

Wraz z rozwojem sztucznej inteligencji, pojawiają się nowe możliwości w zakresie detekcji anomalii i przeciwdziałania atakom cybernetycznym. Generatywne modele AI, takie jak generatywne sieci przeciwdziałania (GANs) czy autoenkodery, wykazują potencjał w identyfikowaniu nietypowych zachowań w sieciach komputerowych, co może skutkować przyczynieniem się do skuteczniejszej ochrony przed atakami.

1.1. Podłoże problemu

Wobec rosnącej liczby incydentów związanych z bezpieczeństwem sieciowym oraz ciągłą potrzebą ulepszania metod detekcji, istnieje niezwłoczna potrzeba zbadania możliwości wykorzystania generatywnych modeli sztucznej inteligencji do identyfikacji anomalii w ruchu sieciowym. Projekt ma na celu zobrazowanie tych możliwości oraz opracowanie metody detekcji opartej na zaawansowanych technikach sztucznej inteligencji.

1.2. Cel projektu

Głównym celem projektu jest zbadanie potencjału oraz skuteczności wykorzystania generatywnych modeli sztucznej inteligencji do detekcji anomalii w ruchu sieciowym. Konkretnie, projekt zakłada:

- **Przegląd istniejących metod detekcji anomalii**: Przeprowadzenie analizy i oceny tradycyjnych oraz nowoczesnych metod detekcji anomalii w sieciach komputerowych, w celu zidentyfikowania ich zalet i ograniczeń.
- **Eksploracja generatywnych modeli AI:** Badanie różnych typów generatywnych modeli sztucznej inteligencji, takich jak generatywne adwersarialne sieci neuronowe (GANs), autoenkodery, modele wariacyjne autoenkoderów (VAEs), generatywne modele przeciwdziałania (GSN), oraz ChatGPT, pod kątem ich potencjalnego zastosowania w detekcji anomalii.
- **Projektowanie nowatorskiej metody detekcji:** Opracowanie zaawansowanej metody detekcji anomalii opartej na wybranym generatywnym modelu sztucznej inteligencji, uwzględniającej specyfikę analizowanego ruchu sieciowego oraz wyzwania związane z identyfikacją nietypowych zachowań.
- **Implementacja i testowanie rozwiązania:** Przeprowadzenie implementacji zaproponowanej metody detekcji w środowisku testowym, a następnie dokładne przetestowanie jej skuteczności na różnorodnych zbiorach danych symulujących rzeczywiste warunki działania sieci komputerowych.
- **Ewaluacja i analiza wyników:** Dokładna analiza wyników eksperymentów oraz ocena skuteczności proponowanej metody detekcji anomalii w porównaniu z istniejącymi rozwiązaniami, w celu określenia jej wartości i przydatności praktycznej.

2. Podstawy teoretyczne

2.1. Anomalie w sieciach komputerowych

Anomalie w sieciach komputerowych to nietypowe lub niezwykłe zachowania, które odbiegają od standardowych norm. Mogą one wskazywać na potencjalne zagrożenia bezpieczeństwa, takie jak ataki hakerskie, malware, błędy w systemie lub problemy z infrastrukturą sieciową. Istnieją różne rodzaje anomalii w sieciach komputerowych, w tym:

- **Anomalie w ruchu sieciowym:** Obejmują nieprzewidywalne wzorce transmisji danych, niezwykle duże lub nieoczekiwane przepływy danych, nietypowe wzorce komunikacji między hostami lub komputerami.
- **Anomalie w zachowaniu użytkowników:** Objawiają się jako niezwykłe lub podejrzane aktywności użytkowników, takie jak próby logowania się do systemu z nieautoryzowanych lokalizacji, nieoczekiwane zapytania o dostęp do poufnych danych, czy podejrzane zmiany w zachowaniu.
- **Anomalie w wykorzystaniu zasobów:** Dotyczą nieprawidłowego lub niezwykle intensywnego korzystania z zasobów sieciowych, takich jak przepustowość sieci, moc obliczeniowa czy pamięć.
- **Anomalie w bezpieczeństwie:** Objawiają się jako niespodziewane próby naruszenia zabezpieczeń, nietypowe logowania się do systemów, próby ataków typu brute-force czy próby wykorzystania luk w zabezpieczeniach.

Rozpoznanie i analiza anomalii w sieciach komputerowych jest kluczowym elementem zapewnienia bezpieczeństwa informacji oraz ochrony infrastruktury przed atakami cybernetycznymi. Metody detekcji anomalii są stosowane do monitorowania ruchu sieciowego, identyfikowania nieprawidłowych zachowań i reagowania na potencjalne zagrożenia w czasie rzeczywistym.

2.2. Metody detekcji anomalii

Metody detekcji anomalii są stosowane w celu identyfikacji nieprawidłowych zachowań w sieciach komputerowych. Istnieje wiele różnych podejść do detekcji anomalii, z których niektóre opierają się na regułach, a inne wykorzystują zaawansowane techniki sztucznej inteligencji. Oto kilka głównych metod detekcji anomalii:

- **Metody oparte na regułach:** Polegają na definiowaniu zestawu reguł lub heurystyk, które określają, co jest uznawane za normalne zachowanie w sieci, a co za anomalie. Przykłady to detekcja nietypowych wzorców ruchu, nadmiernego wykorzystania zasobów lub podejrzanych aktywności użytkowników.
- **Metody statystyczne:** Te metody analizują statystyki dotyczące ruchu sieciowego, takie jak średnie, odchylenia standardowe czy histogramy, aby identyfikować anomalie. Przykłady obejmują analizę anomalii na podstawie rozkładów prawdopodobieństwa czy wykrywanie odstępstw od typowych wzorców czasowych.
- **Metody uczenia maszynowego:** Wykorzystują zaawansowane techniki uczenia maszynowego, aby nauczyć się rozpoznawać anomalie na podstawie danych treningowych. Przykłady to algorytmy klasyfikacji, grupowania czy generatywne modele sztucznej inteligencji, takie jak autoenkodery czy generatywne adversarialne sieci neuronowe (GANs).
- **Metody bazujące na heurystykach i eksperckiej wiedzy:** Te podejścia wykorzystują wiedzę ekspercką w dziedzinie bezpieczeństwa informatycznego do

definiowania reguł detekcji anomalii. Mogą być one skuteczne w identyfikowaniu subtelnych ataków, które mogą być trudne do wykrycia za pomocą innych metod.

W praktyce często stosuje się kombinacje różnych metod detekcji anomalii, aby zwiększyć skuteczność systemu i zmniejszyć liczbę fałszywych alarmów. Każda z tych metod ma swoje zalety i ograniczenia, dlatego istotne jest dopasowanie ich do konkretnego środowiska i potrzeb organizacji.

2.3. Generatywne modele sztucznej inteligencji

Generatywne modele sztucznej inteligencji (AI) są rodzajem modeli, które mają zdolność generowania nowych danych na podstawie danych treningowych, które zostały użyte do ich nauki.

2.3.1 Podstawy działania

Generatywne modele sztucznej inteligencji (AI) są rodzajem modeli, które mają zdolność generowania nowych danych na podstawie danych treningowych, które zostały użyte do ich nauki. Podstawowa idea polega na tym, że model jest trenowany na zbiorze danych treningowych w taki sposób, aby nauczył się reprezentować rozkład prawdopodobieństwa danych wejściowych. Następnie, po zakończeniu procesu uczenia, model jest w stanie generować nowe przykłady danych, które są podobne do tych, które widział w procesie uczenia.

Generatywne modele sztucznej inteligencji działają na zasadzie próby odtworzenia danych treningowych oraz nauki ich rozkładów prawdopodobieństwa. Są one zwykle oparte na głębokich sieciach neuronowych, takich jak generatywne adversarialne sieci neuronowe (GANs), autoenkodery czy variational autoencoders (VAEs). Te modele wykorzystują różne techniki, takie jak propagacja wsteczna, funkcje straty, optymalizacja gradientowa itp., aby nauczyć się reprezentacji danych i generować nowe przykłady.

2.3.2. Przykłady zastosowań

Generatywne modele sztucznej inteligencji znajdują szerokie zastosowanie w wielu dziedzinach, zarówno w badaniach naukowych, jak i w praktycznych aplikacjach. Kilka przykładów zastosowań obejmuje:

- **Generowanie obrazów:** Generatywne modele AI są wykorzystywane do generowania realistycznych obrazów, takich jak portrety, krajobrazy czy tekstury. Przykłady to tworzenie obrazów twarzy ludzkich za pomocą GANs.
- **Generowanie tekstu:** Te modele mogą być również używane do generowania tekstu, takiego jak opowiadania, artykuły czy dialogi. Przykładem może być generowanie tekstu na podstawie zadanego tematu za pomocą modeli języka naturalnego opartych na GANs.
- **Detekcja anomalii:** Generatywne modele sztucznej inteligencji mogą być wykorzystane do detekcji anomalii w danych, w tym w ruchu sieciowym, obrazach medycznych czy analizie finansowej. Przykładowo, autoenkodery mogą być użyte do wykrywania nietypowych wzorców w danych, które mogą wskazywać na ataki cybernetyczne.
- **Generowanie dźwięku i muzyki:** Generatywne modele mogą generować dźwięki i muzykę, co znalazło zastosowanie w produkcji audio i kompozycji muzycznej. Przykłady obejmują generowanie melodii muzycznych za pomocą VAEs.

Te przykłady pokazują, że generatywne modele sztucznej inteligencji mają ogromny potencjał w różnych dziedzinach i mogą być wykorzystane do różnorodnych zastosowań, od tworzenia sztuki po rozwiązania w obszarze bezpieczeństwa i analizy danych.

3. Analiza istniejących rozwiązań

3.1. Tradycyjne metody detekcji anomalii

Tradycyjne metody detekcji anomalii obejmują szeroki zakres technik i algorytmów, które zostały rozwinięte przed pojawieniem się generatywnych modeli sztucznej inteligencji. Poniżej przedstawiono kilka głównych tradycyjnych metod detekcji anomalii:

- **Statystyczne podejścia:** Metody oparte na statystyce wykorzystują analizę statystyczną danych, aby identyfikować odstępstwa od normy. Przykłady to analiza histogramów, wykrywanie odchyleń standardowych czy badanie rozkładów prawdopodobieństwa.
- **Metody oparte na regułach:** Podejścia te polegają na definiowaniu zestawu reguł lub heurystyk, które określają, co jest uznawane za normalne zachowanie w sieci. Przykłady to wykrywanie nietypowych wzorców ruchu sieciowego, analiza logów systemowych czy monitorowanie nieautoryzowanego dostępu.
- **Analiza zachowań użytkowników:** Metody te koncentrują się na analizie zachowań użytkowników w sieci, aby wykryć nieprawidłowości, takie jak niezwykłe aktywności, podejrzane logowania czy nieoczekiwane zmiany w zachowaniu.
- **Metody bazujące na maszynowym uczeniu:** Choć są one bardziej nowoczesne, to nadal zaliczają się do tradycyjnych metod. Obejmują one algorytmy uczenia maszynowego, takie jak klasyfikacja, regresja czy klasteryzacja, które są wykorzystywane do identyfikowania anomalii w danych.

Analiza tradycyjnych metod detekcji anomalii pozwala na zrozumienie ich zalet i ograniczeń w kontekście obecnych wyzwań w zakresie bezpieczeństwa informatycznego. Choć te metody były powszechnie stosowane przez wiele lat, to coraz częściej wykorzystuje się generatywne modele sztucznej inteligencji, aby poprawić skuteczność detekcji i redukcję liczby fałszywych alarmów.

3.2. Wykorzystanie modeli AI w detekcji ataków

Wykorzystanie modeli sztucznej inteligencji (AI) w detekcji ataków stanowi coraz bardziej obiecującą strategię w walce z rosnącym zagrożeniem cybernetycznym. Przy użyciu zaawansowanych technik uczenia maszynowego i głębokich sieci neuronowych, modele AI są w stanie analizować duże ilości danych w czasie rzeczywistym, identyfikując nietypowe wzorce i podejrzane zachowania, które mogą wskazywać na potencjalne ataki. Poniżej przedstawiono kilka głównych sposobów, w jakie modele AI są wykorzystywane w detekcji ataków:

- **Analiza zachowań:** Modele AI mogą analizować zachowania użytkowników oraz ruch sieciowy, aby wykryć niezwykłe lub podejrzane aktywności. Na podstawie analizy dużej ilości danych, modele są w stanie identyfikować nietypowe wzorce, które mogą wskazywać na ataki typu phishing, ransomware czy ataki hakerskie.
- **Detekcja nieznanych zagrożeń:** Modele AI mogą być trenowane na danych historycznych zawierających informacje o znanym zachowaniu atakujących. Następnie, po zakończeniu procesu uczenia, modele są w stanie rozpoznawać nowe,

nieznane zagrożenia, które nie pasują do żadnego znanego wzorca, co pozwala na szybką reakcję na pojawiające się ataki.

- **Automatyczne wykrywanie luk w zabezpieczeniach:** Modele AI mogą być wykorzystywane do skanowania systemów i aplikacji w poszukiwaniu potencjalnych luk w zabezpieczeniach, które mogą być wykorzystane przez cyberprzestępców do przeprowadzenia ataków. Wykorzystanie technik uczenia maszynowego pozwala na szybkie i skuteczne wykrycie podatności oraz zastosowanie odpowiednich środków zaradczych.
- **Adaptacyjne uczenie maszynowe:** Modele AI mogą być ciągle aktualizowane i dostosowywane do zmieniającego się środowiska sieciowego oraz nowych rodzajów ataków. Dzięki zdolności do adaptacji, modele są w stanie szybko reagować na nowe zagrożenia i dostosowywać się do zmieniających się warunków.

Wykorzystanie modeli AI w detekcji ataków pozwala na szybką i skuteczną identyfikację oraz eliminację potencjalnych zagrożeń cybernetycznych. W połączeniu z innymi metodami obrony, takimi jak firewalle, systemy detekcji intruzów czy szyfrowanie danych, modele AI stanowią kluczowy element w zapewnieniu bezpieczeństwa infrastruktury informatycznej przed atakami.

3.3. Ograniczenia istniejących rozwiązań

Mimo postępu w dziedzinie detekcji anomalii i wykorzystaniu modeli sztucznej inteligencji, istnieją pewne ograniczenia, które należy uwzględnić przy analizie istniejących rozwiązań:

- **Fałszywe alarmy:** Istnieje ryzyko generowania fałszywych alarmów, zwłaszcza przy stosowaniu zaawansowanych technik sztucznej inteligencji. Nieprawidłowa klasyfikacja normalnego zachowania jako anomalii może prowadzić do nadmiernego wyzwalania alarmów, co może zwiększać obciążenie dla personelu ds. bezpieczeństwa oraz prowadzić do zaniedbywania rzeczywistych zagrożeń.
- **Brak interpretowalności:** Niektóre modele sztucznej inteligencji, szczególnie te oparte na głębokim uczeniu, mogą być trudne do zrozumienia i interpretacji przez ludzi. Brak przejrzystości w działaniu modeli może utrudniać śledzenie procesu decyzyjnego oraz diagnozowanie przyczyn wykrytych anomalii.
- **Znaczące zużycie zasobów:** Niektóre zaawansowane modele sztucznej inteligencji, zwłaszcza te oparte na głębokim uczeniu, mogą wymagać znacznych zasobów obliczeniowych i pamięciowych, co może być wyzwaniem dla organizacji o ograniczonych zasobach technicznych.
- **Złożoność implementacji i utrzymania:** Wdrożenie i utrzymanie zaawansowanych systemów detekcji anomalii opartych na modelach sztucznej inteligencji może być skomplikowane i wymagać zaangażowania wysoko wykwalifikowanych specjalistów. Dodatkowo, aktualizacje i dostosowania do zmieniającego się środowiska mogą być czasochłonne i kosztowne.
- **Wyzwania związane z danymi treningowymi:** Skuteczność modeli sztucznej inteligencji w detekcji anomalii zależy w dużej mierze od jakości i reprezentatywności danych treningowych. Brak wystarczającej ilości danych treningowych lub obecność skrzywionych danych może prowadzić do niewłaściwego uczenia modelu i zmniejszać jego skuteczność w identyfikacji rzeczywistych zagrożeń.

Uwzględnienie tych ograniczeń jest kluczowe przy projektowaniu i wdrażaniu systemów detekcji anomalii, aby zapewnić skuteczną ochronę infrastruktury informatycznej przed atakami cybernetycznymi. Warto również stale monitorować

postęp w dziedzinie sztucznej inteligencji i poszukiwać nowych metod i technik, które mogą pomóc w przezwyciężeniu tych ograniczeń.

4. Projektowana metoda detekcji anomalii

4.1. Wybór generatywnego modelu AI

W kontekście tego projektu jako generatywnego modelu AI został wybrany ChatGPT. ChatGPT, oparty na architekturze GPT (Generative Pre-trained Transformer), jest zaawansowanym modelem językowym stworzonym przez OpenAI. ChatGPT został wytrenowany na dużym zbiorze danych tekstowych i posiada zdolność do generowania płynnych i sensownych odpowiedzi na podstawie zadanego kontekstu.

Wybór ChatGPT jako generatywnego modelu AI w projekcie ma na celu wykorzystanie jego zdolności do generowania tekstu w celu eksploracji i analizy danych dotyczących anomalii w sieciach komputerowych. ChatGPT może być użyty do generowania opisów anomalii, analizy logów zdarzeń, tworzenia raportów oraz interaktywnej eksploracji danych.

Jednym z głównych powodów wyboru ChatGPT jest jego wszechstronność i zdolność do generowania wysokiej jakości tekstów w różnych dziedzinach. Ponadto, jako model językowy, ChatGPT może być dostosowany do specyficznych potrzeb projektu poprzez dodanie dodatkowych danych treningowych lub dostosowanie hiperparametrów.

4.2. Opis zbioru danych

Jako zbiór danych przyjęto UNSW-NB 15, czyli akademicki zbiór danych anomalii, jego autorem jest dr. Nour Moustafa i dr. Jill Slay. Surowe pakiety sieciowe zbioru danych UNSW-NB 15 zostały utworzone za pomocą narzędzia IXIA PerfectStorm w laboratorium Cyber Range na UNSW w Canberze w celu wygenerowania hybrydy rzeczywistych, nowoczesnych, normalnych działań i syntetycznych, współczesnych zachowań ataku. Do przechwycenia 100 GB nieprzetworzonego ruchu (np. plików Pcap) wykorzystano narzędzie tcpdump. Ten zbiór danych obejmuje dziewięć typów ataków, a mianowicie Fuzzery, Analiza, Backdoory, DoS, Exploity, Generic, Reconnaissance, Shellcode i Worms. Wykorzystuje się narzędzia Argus, Bro-IDS i opracowano dwanaście algorytmów w celu wygenerowania łącznie 49 funkcji z etykietą klasy.

Partycja z tego zbioru danych jest skonfigurowana jako zbiór szkoleniowy i zbiór testowy, a mianowicie odpowiednio UNSW_NB15_training-set.csv i UNSW_NB15_testing-set.csv. Liczba rekordów w zestawie szkoleniowym wynosi 175 341 rekordów, a zestaw testowy to 82 332 rekordy różnych typów, ataku i normalnego.

Zbiór danych składa się z 49 unikalnych kolumn, które wraz z krótkim opisem zostały przedstawione poniżej:

1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR,...
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the ?ow packet size transmitted by the src
24	dmeansz	integer	Mean of the ?ow packet size transmitted by the dst
25	trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
26	res_bdy_len	integer	Actual uncompressed content size of the data transferred from the server$s http service.
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)

33	tcprtt	Float	TCP connection setup round-trip time, the sum of ﬠsynackﬠ and ﬠackdatﬠ.
34	synack	Float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
35	ackdat	Float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
36	is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variab...
37	ct_state_ttl	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10...
38	ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and Post in http service.
39	is_ftp_login	Binary	If the ftp session is accessed by user and password then 1 else 0.
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.
41	ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections acco...
42	ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections...
43	ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time...
44	ct_src_ ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26)...
45	ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections acc...
46	ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections acc...
47	ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections accor...
48	attack_cat	nominal	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoo...
49	Label	binary	0 for normal and 1 for attack records

Rys. 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10. Przedstawienie wszystkich kolumn zbioru danych wraz z krótkim opisem

4.3. Przygotowanie danych treningowych

Wykorzystany zbiór danych został już odpowiednio przygotowany i wyczyszczony oraz podzielony na plik train oraz test, czyli plik z danymi uczącymi oraz plik z danymi testującymi. W przygotowanym zbiorze danych występuje 0 brakujących rekordów a poniżej przedstawione zostało przygotowanie zbioru danych:

```
import numpy as np
import pandas as pd
```

Przydatne biblioteki do zaimportowania odpowiedzialne za algebrę liniową oraz przetwarzanie danych, zapisywanie do pliku csv.

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Zaimportowanie modułu os, wykorzystywanego do interakcji z systemem operacyjnym. Przeszukanie katalogu '/kaggle/input' w celu znalezienia wszystkich plików i katalogów w tym miejscu. Dla każdego katalogu, zwracane są trzy elementy: ścieżka katalogu (dirname), lista podkatalogów (pominięte w kodzie) oraz listę plików w danym katalogu (filenames). W uproszczeniu, ten kod służy do wyświetlenia pełnej ścieżki dostępu do wszystkich plików znajdujących się w katalogu '/kaggle/input' oraz jego podkatalogach.

```
/kaggle/input/unswnb15/UNSW_NB15_testing-set.parquet
/kaggle/input/unswnb15/UNSW_NB15_training-set.parquet
/kaggle/input/unswnb15/UNSW_NB15_testing-set.csv
/kaggle/input/unswnb15/UNSW_NB15_training-set.csv
```

Efekt danych zwróconych przez kod. Następnie pomocne będzie zaimportowanie biblioteki fastai i funkcji df_shrink.

```
from fastai.tabular.all import df_shrink
```

Przypisanie do zmiennej adresu zbioru danych uczących.

```
to_clean = '/kaggle/input/unswnb15/UNSW_NB15_training-set.csv'
```

Wczytanie pliku CSV do obiektu DataFrame, zastosowując określone opcje wczytania, takie jak pomijanie dodatkowych białych znaków na początku komórek i wybór odpowiedniego kodowania.

```
df = pd.read_csv(to_clean, skipinitialspace=True, low_memory=False
, encoding='utf-8')
```

Wyrzucenie kolumny 'id', ponieważ jest zupełnie niepotrzebna.

```
df.drop(columns=['id'], inplace=True)
```

Domyślne typy danych są albo niespecyficzne, albo zbyt duże. Funkcja df_shrink z biblioteki fastai pomaga lepiej sprecyzować typy danych. Poniżej przedstawiono je przed transformacją oraz po.

```
df.dtypes
```

dur	float64	dtcpb	int64
proto	object	dwin	int64
service	object	tcprtt	float64
state	object	synack	float64
spkts	int64	ackdat	float64
dpkts	int64	smean	int64
sbytes	int64	dmean	int64
dbytes	int64	trans_depth	int64
rate	float64	response_body_len	int64
sttl	int64	ct_srv_src	int64
dttl	int64	ct_state_ttl	int64
sload	float64	ct_dst_ltm	int64
dload	float64	ct_src_dport_ltm	int64
sloss	int64	ct_dst_sport_ltm	int64
dloss	int64	ct_dst_src_ltm	int64
sinpkt	float64	is_ftp_login	int64
dinpkt	float64	ct_ftp_cmd	int64
sjit	float64	ct_flw_http_mthd	int64
djit	float64	ct_src_ltm	int64
swin	int64	ct_srv_dst	int64
stcpb	int64	is_sm_ips_ports	int64
		attack_cat	object
		label	int64

Rys. 4.11. Przedstawienie wyniku polecenia df.dtypes

Użycie funkcji df_shrink w celu zmniejszenia rozmiaru generowanego przez niepotrzebnie użyty typy zmiennych.

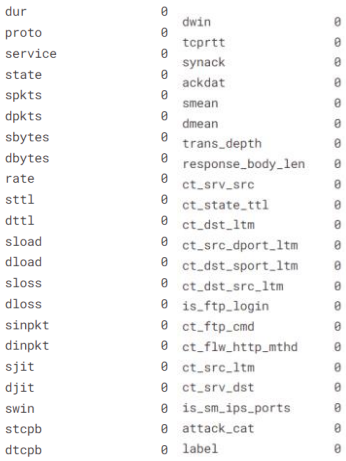
```
df = df_shrink(df, skip=[], obj2cat=True, int2uint=False)
df.dtypes
```

dur	float32	tcprtt	float32
proto	category	synack	float32
service	category	ackdat	float32
state	category	smean	int16
spkts	int16	dmean	int16
dpkts	int16	trans_depth	int16
sbytes	int32	response_body_len	int32
dbytes	int32	ct_srv_src	int8
rate	float32	ct_state_ttl	int8
sttl	int16	ct_dst_ltm	int8
dttl	int16	ct_src_dport_ltm	int8
sload	float32	ct_dst_sport_ltm	int8
dload	float32	ct_dst_src_ltm	int8
sloss	int16	is_ftp_login	int8
dloss	int16	ct_ftp_cmd	int8
sinpkt	float32	ct_flw_http_mthd	int8
dinpkt	float32	ct_src_ltm	int8
sjit	float32	ct_srv_dst	int8
djit	float32	is_sm_ips_ports	int8
swin	int16	attack_cat	category
stcpb	int64	label	int8
dtcpb	int64		
dwin	int16		

Rys. 4.12. Przedstawienie wyniku polecenia df.dtypes

Poszukiwanie N/A wartości przy pomocy polecenia:

```
df.isna().sum()
```



dur	0	dwin	0
proto	0	tcprtt	0
service	0	synack	0
state	0	ackdat	0
spkts	0	smean	0
dpkts	0	dmean	0
sbytes	0	trans_depth	0
dbytes	0	response_body_len	0
rate	0	ct_srv_src	0
sttl	0	ct_state_ttl	0
dttl	0	ct_dst_ltm	0
sload	0	ct_src_dport_ltm	0
dload	0	ct_dst_sport_ltm	0
sloss	0	ct_dst_src_ltm	0
dloss	0	is_ftp_login	0
sinpkt	0	ct_ftp_cmd	0
dinpkt	0	ct_flw_http_mthd	0
sjit	0	ct_src_ltm	0
djit	0	ct_srv_dst	0
swin	0	is_sm_ips_ports	0
stcpb	0	attack_cat	0
dtcpb	0	label	0

Rys. 4.13. Przedstawienie wyniku polecenia df.isna().sum()

Dane wyjściowe na parquet, bardzo wydajny, binarny format tabelaryczny. W przypadku tego zbioru danych miejsce na dysku zostało zmniejszone o około 66%, a ładowanie każdego zbioru danych za pomocą pd.read_parquet() będzie niesamowicie szybkie. Kod poniżej służy do zapisania obiektu DataFrame do pliku w formacie Parquet o nazwie i lokalizacji określonej przez ścieżkę oraz wyświetlona zostanie lista plików i katalogów znajdujących się w lokalizacji '/kaggle/working/'.

```
df.to_parquet(f"/kaggle/working/{to_clean.split('/')[-1].replace('
.csv', '.parquet')}")
!ls -lth /kaggle/input/unswnb15
```

Efekt działania polecenia i wyświetlona lista plików i katalogów:

```
total 60M
-rw-r--r-- 1 nobody nogroup  31M Aug  6 13:04 UNSW_NB15_testing-set.csv
-rw-r--r-- 1 nobody nogroup 4.7M Aug  6 13:04 UNSW_NB15_training-set.parquet
-rw-r--r-- 1 nobody nogroup 9.9M Aug  6 13:04 UNSW_NB15_testing-set.parquet
-rw-r--r-- 1 nobody nogroup  15M Aug  6 13:04 UNSW_NB15_training-set.csv
```

Ten kod wylistuje pliki i katalogi w podanej lokalizacji '/kaggle/working' i wyświetli ich nazwy, rozmiary (w czytelnej formie) oraz daty modyfikacji. Pliki i katalogi będą posortowane według daty modyfikacji, z najnowszymi na początku.

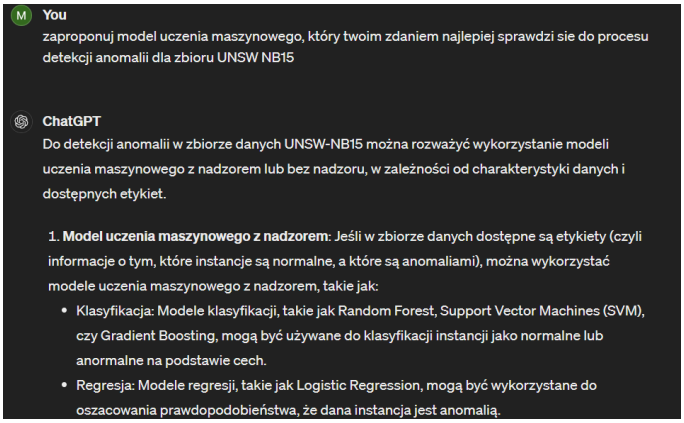
```
!ls -lth /kaggle/working
```

Efekt działania polecenia i wyświetlona lista plików i katalogów:

```
total 4.7M
---------- 1 root root  19K Aug  6 13:04 __notebook__.ipynb
-rw-r--r-- 1 root root 4.7M Aug  6 13:04 UNSW_NB15_training-set.parquet
```

4.4. Architektura proponowanego systemu

Model uczenia maszynowego zaproponowany przez generatywny model AI ChatGPT:



Rys. 4.14. Przedstawienie wyniku zwróconego przez ChatGPT

Kod programu napisanego przy użyciu narzędzia ChatGPT po kliku próbach i odpowiedniej modyfikacji względem danych, ma za zadanie nauczenie modelu RandomForest do rozpoznawania czy dane wskazują na atak:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

#Wczytanie danych treningowych i testowych
dftrain = pd.read_parquet('/content/drive/MyDrive/Colab
Notebooks/UNSW_NB15_training-set.parquet')
dftest = pd.read_parquet('/content/drive/MyDrive/Colab
Notebooks/UNSW_NB15_testing-set.parquet')

#Usunięcie kolumn 'label' i 'attack_cat' z danych treningowych
X_train = dftrain.drop(['label', 'attack_cat'], axis=1)
#Kolumny 'label' i 'attack_cat' jako etykiety dla danych treningowych
y_train = dftrain[['label', 'attack_cat']]

#Usunięcie kolumn 'label' i 'attack_cat' z danych testowych
X_test = dftest.drop(['label', 'attack_cat'], axis=1)
#Kolumny 'label' i 'attack cat' jako etykiety dla danych testowych
y_test = dftest[['label', 'attack_cat']]

#Przekształcenie zmiennych kategorycznych za pomocą kodowania one-hot
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)

X_trainf, X_testf, y_trainf, y_testf = train_test_split(X_train_encoded,
y_train, test_size=0.25, random_state=42)
```

```
X_trainff, X_testff, y_trainff, y_testff = train_test_split(X_test_encoded,
y_test, test_size=0.25, random_state=42)

#Inicjalizacja i trenowanie modelu Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_trainf, y_trainf['label'])

#Inicjalizacja i trenowanie modelu Random Forest
rf_model2 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model2.fit(X_trainff, y_trainff['label'])

#Predykcja na zbiorze testowym
y_pred = rf_model.predict(X_testf)

#Predykcja na zbiorze testowym
y_pred2 = rf_model2.predict(X_testff)

#Ocena modelu
print("Confusion Matrix:")
print(confusion_matrix(y_testf['label'], y_pred))
print("\nClassification Report:")
print(classification_report(y_testf['label'], y_pred))

#Ocena modelu
print("Confusion Matrix:")
print(confusion_matrix(y_testff['label'], y_pred2))
print("\nClassification Report:")
print(classification_report(y_testff['label'], y_pred2))
```

Kod po małych modyfikacjach, zmienione elementy widoczne są poniżej, mają na celu uczenie modelu RandomForest do rozpoznawania rodzaju ataku spośród kilku opcji:

```
#Inicjalizacja i trenowanie modelu Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_trainf, y_trainf['attack_cat'])

#Inicjalizacja i trenowanie modelu Random Forest
rf_model2 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model2.fit(X_trainff, y_trainff['attack_cat'])

#Predykcja na zbiorze testowym
y_pred = rf_model.predict(X_testf)

# Predykcja na zbiorze testowym
y_pred2 = rf_model2.predict(X_testff)

#Ocena modelu
print("Confusion Matrix:")
```

```
print(confusion_matrix(y_testf['attack_cat'], y_pred))
print("\nClassification Report:")
print(classification_report(y_testf['attack_cat'], y_pred))

#Ocena modelu
print("Confusion Matrix:")
print(confusion_matrix(y_testff['attack_cat'], y_pred2))
print("\nClassification Report:")
print(classification_report(y_testff['attack_cat'], y_pred2))
```

W zbiorze danych, były już pliki z zawartymi oddzielnie danymi uczącymi oraz testującymi, niestety ChatGPT nie poradził sobie z odpowiednim dopasowaniem się do danych więc, dane zostały dwa razy podzielone na dane uczące oraz testowe dla tych dwóch plików i zostały przeprowadzone dwa procesy uczenia. Dane zostały podzielone w proporcjach 75% dane uczące, 25% dane testowe.

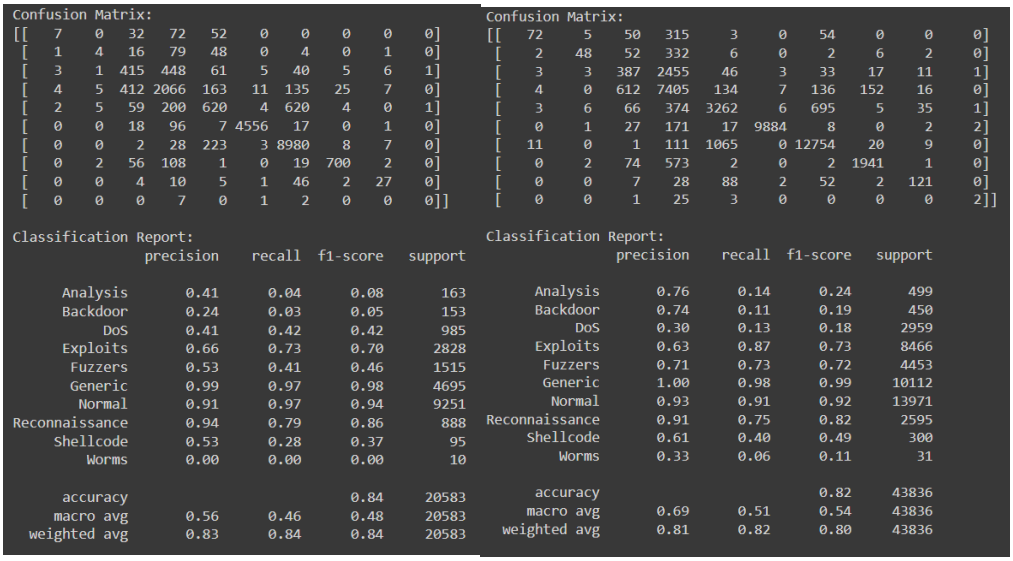
4.5. Proces detekcji anomalii

Oto efekty przeprowadzonego procesu uczenia dla danych z dwóch plików, można zauważyć, że w drugim pliku zawiera się więcej przykładów, przedstawione zostały confussion matrix oraz raporty klasyfikacji. Dokładność jest na zadowalającym poziomie 0.95 w obu przypadkach, precyzja także przekracza wartość 0.92.

```
Confusion Matrix:
[[ 8887   364]
 [  753 10579]]

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      9251
           1       0.97      0.93      0.95     11332

    accuracy                           0.95     20583
   macro avg       0.94      0.95      0.95     20583
weighted avg       0.95      0.95      0.95     20583

Confusion Matrix:
[[12531  1440]
 [  690 29175]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.90      0.92     13971
           1       0.95      0.98      0.96     29865

    accuracy                           0.95     43836
   macro avg       0.95      0.94      0.94     43836
weighted avg       0.95      0.95      0.95     43836
```

Rys. 4.15. Przedstawienie wyników dla modelu rozpoznającego czy wystąpił atak czy nie

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model RandomForest poradził sobie zdecydowanie gorzej, mimo tego dokładność dwa razy przekroczyła 80%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: generic, normal, reconnaissance. Widoczny jest także wpływ ilości danych dla konkretnego przypadku na zdolność modelu po nauczeniu się do jego rozpoznania. Przypadki z dużą ilością danych testowych wypadają znacznie lepiej niż przykładowo atak worms.



```
Confusion Matrix:
[[   7    0   32   72   52    0    0    0    0    0]
 [   1    4   16   79   48    0    4    0    1    0]
 [   3    1  415  448   61    5   40    5    6    1]
 [   4    5  412 2066  163   11  135   25    7    0]
 [   2    5   59  200  620    4  620    4    0    1]
 [   0    0   18   96    7 4556   17    0    1    0]
 [   0    0    2   28  223    3 8980    8    7    0]
 [   0    2   56  108    1    0   19  700    2    0]
 [   0    0    4   10    5    1   46    2   27    0]
 [   0    0    0    7    0    1    2    0    0    0]]

Classification Report:
                precision    recall  f1-score   support

      Analysis       0.41      0.04      0.08       163
      Backdoor       0.24      0.03      0.05       153
           DoS       0.41      0.42      0.42       985
      Exploits       0.66      0.73      0.70      2828
       Fuzzers       0.53      0.41      0.46      1515
       Generic       0.99      0.97      0.98      4695
        Normal       0.91      0.97      0.94      9251
Reconnaissance       0.94      0.79      0.86       888
     Shellcode       0.53      0.28      0.37        95
         Worms       0.00      0.00      0.00        10

      accuracy                           0.84     20583
     macro avg       0.56      0.46      0.48     20583
  weighted avg       0.83      0.84      0.84     20583
```

```
Confusion Matrix:
[[   72     5    50   315     3     0    54     0     0     0]
 [    2    48    52   332     6     0     2     6     2     0]
 [    3     3   387  2455    46     3    33    17    11     1]
 [    4     0   612  7405   134     7   136   152    16     0]
 [    3     6    66   374  3262     6   695     5    35     1]
 [    0     1    27   171    17  9884     8     0     2     2]
 [   11     0     1   111  1065     0 12754    20     9     0]
 [    0     2    74   573     2     0     2  1941     1     0]
 [    0     0     7    28    88     2    52     2   121     0]
 [    0     0     1    25     3     0     0     0     0     2]]

Classification Report:
                precision    recall  f1-score   support

      Analysis       0.76      0.14      0.24       499
      Backdoor       0.74      0.11      0.19       450
           DoS       0.30      0.13      0.18      2959
      Exploits       0.63      0.87      0.73      8466
       Fuzzers       0.71      0.73      0.72      4453
       Generic       1.00      0.98      0.99     10112
        Normal       0.93      0.91      0.92     13971
Reconnaissance       0.91      0.75      0.82      2595
     Shellcode       0.61      0.40      0.49       300
         Worms       0.33      0.06      0.11        31

      accuracy                           0.82     43836
     macro avg       0.69      0.51      0.54     43836
  weighted avg       0.81      0.82      0.80     43836
```

Rys. 4.16. i 4.17. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku

5. Porównanie innych modeli do detekcji anomalii

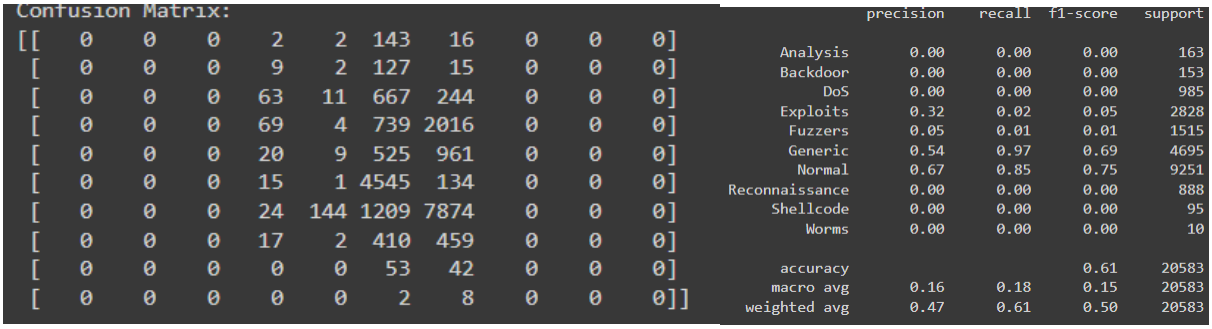
W celu porównania wyników modelu zaproponowanego przez ChatGPT do innych została zmieniona tylko niewielka część kodu odpowiedzialna za inicjalizację modelu.

5.1. Model Regresji Logistycznej

Kod programu napisanego dla modelu Regresji Logistycznej:

```
from sklearn.linear_model import LogisticRegression
#Inicjalizacja i trenowanie modelu LR
rf_model = LogisticRegression(
    random_state=42,  #Ustawienie ziarna losowości dla powtarzalności wyników
    max_iter=1000,  #Zwiększenie maksymalnej liczby iteracji dla zapewnienia
zbieżności
    solver='lbfgs',  # stawienie solvera na 'lbfgs' (Algorytm Quasi-Newtona)
    multi_class='auto'  #Określenie metody dla wielu klas ('auto' automatycznie
wybiera metodę)
)
rf_model.fit(X_trainf, y_trainf['attack_cat'])

#Inicjalizacja i trenowanie modelu LR
rf_model2 = LogisticRegression(
    random_state=42,  #Ustawienie ziarna losowości dla powtarzalności wyników
    max_iter=1000,  #Zwiększenie maksymalnej liczby iteracji dla zapewnienia
zbieżności
    solver='lbfgs',  #Ustawienie solvera na 'lbfgs' (Algorytm Quasi-Newtona)
    multi_class='auto'  #Określenie metody dla wielu klas ('auto' automatycznie
wybiera metodę)
)
rf_model2.fit(X_trainff, y_trainff['attack_cat'])
```

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model Logistic Regression poradził sobie zdecydowanie gorzej, dokładność dwa razy przekroczyła 50%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: generic, normal.

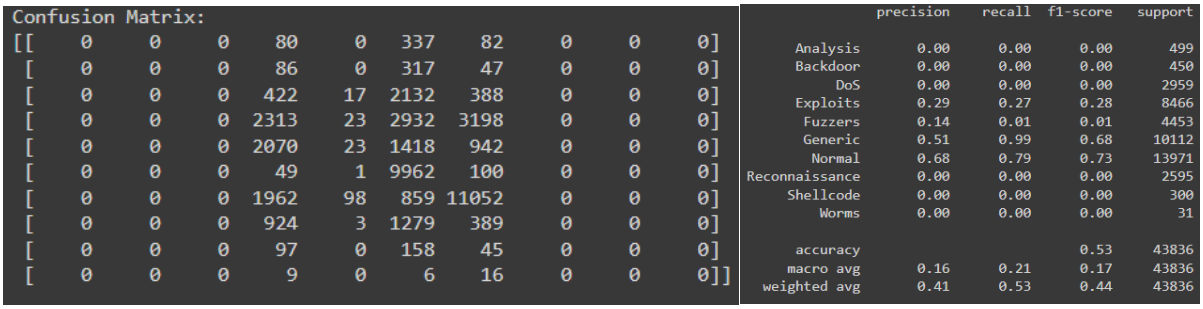


Confusion Matrix:

```
[[   0    0    0    2    2  143   16    0    0    0]
 [   0    0    0    9    2  127   15    0    0    0]
 [   0    0    0   63   11  667  244    0    0    0]
 [   0    0    0   69    4  739 2016    0    0    0]
 [   0    0    0   20    9  525  961    0    0    0]
 [   0    0    0   15    1 4545  134    0    0    0]
 [   0    0    0   24  144 1209 7874    0    0    0]
 [   0    0    0   17    2  410  459    0    0    0]
 [   0    0    0    0    0   53   42    0    0    0]
 [   0    0    0    0    0    2    8    0    0    0]]
```

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	163
Backdoor	0.00	0.00	0.00	153
DoS	0.00	0.00	0.00	985
Exploits	0.32	0.02	0.05	2828
Fuzzers	0.05	0.01	0.01	1515
Generic	0.54	0.97	0.69	4695
Normal	0.67	0.85	0.75	9251
Reconnaissance	0.00	0.00	0.00	888
Shellcode	0.00	0.00	0.00	95
Worms	0.00	0.00	0.00	10
accuracy			0.61	20583
macro avg	0.16	0.18	0.15	20583
weighted avg	0.47	0.61	0.50	20583

Rys. 5.1. i 5.2. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 1



Confusion Matrix:

```
[[   0    0    0   80    0  337   82    0    0    0]
 [   0    0    0   86    0  317   47    0    0    0]
 [   0    0    0  422   17 2132  388    0    0    0]
 [   0    0    0 2313   23 2932 3198    0    0    0]
 [   0    0    0 2070   23 1418  942    0    0    0]
 [   0    0    0   49    1 9962  100    0    0    0]
 [   0    0    0 1962   98  859 11052    0    0    0]
 [   0    0    0  924    3 1279  389    0    0    0]
 [   0    0    0   97    0  158   45    0    0    0]
 [   0    0    0    9    0    6   16    0    0    0]]
```

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	499
Backdoor	0.00	0.00	0.00	450
DoS	0.00	0.00	0.00	2959
Exploits	0.29	0.27	0.28	8466
Fuzzers	0.14	0.01	0.01	4453
Generic	0.51	0.99	0.68	10112
Normal	0.68	0.79	0.73	13971
Reconnaissance	0.00	0.00	0.00	2595
Shellcode	0.00	0.00	0.00	300
Worms	0.00	0.00	0.00	31
accuracy			0.53	43836
macro avg	0.16	0.21	0.17	43836
weighted avg	0.41	0.53	0.44	43836

Rys. 5.3. i 5.4. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 2

5.2. Model K-nearest Neighbors

Kod programu napisanego dla modelu KNC:

```
from sklearn.neighbors import KNeighborsClassifier
#Inicjalizacja i trenowanie modelu KNN
rf_model = KNeighborsClassifier()
rf_model.fit(X_trainf, y_trainf['attack_cat'])


#Inicjalizacja i trenowanie modelu KNN
rf_model2 = KNeighborsClassifier()
rf_model2.fit(X_trainff, y_trainff['attack_cat'])
```

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model KNC poradził sobie trochę lepiej niż regresja, dokładność dwa razy przekroczyła 65%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: generic, normal, reconnaissance.

```
Confusion Matrix:
[[   2    3   43   85   20    0   10    0    0    0]
 [  13    0   23   79   27    0   11    0    0    0]
 [  26    6  278  433   84    1  156    1    0    0]
 [  32    9  312 1050  206    5 1195   18    1    0]
 [  20   12  102  411  284    7  667   11    1    0]
 [   0    1   12   49   14 4531   86    2    0    0]
 [   6    2   78 1042  379   16 7702   24    2    0]
 [   3    4   36  186   32    3  288  336    0    0]
 [   0    1    4   15   15    2   52    4    2    0]
 [   0    0    0    3    2    0    5    0    0    0]]
```

	precision	recall	f1-score	support
Analysis	0.02	0.01	0.02	163
Backdoor	0.00	0.00	0.00	153
DoS	0.31	0.28	0.30	985
Exploits	0.31	0.37	0.34	2828
Fuzzers	0.27	0.19	0.22	1515
Generic	0.99	0.97	0.98	4695
Normal	0.76	0.83	0.79	9251
Reconnaissance	0.85	0.38	0.52	888
Shellcode	0.33	0.02	0.04	95
Worms	0.00	0.00	0.00	10
accuracy			0.69	20583
macro avg	0.38	0.31	0.32	20583
weighted avg	0.68	0.69	0.68	20583

Rys. 5.5. i 5.6. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 1

```
Confusion Matrix:
[[   77     2   237   127    22     0    29     5     0     0]
 [   48     4   229   136    17     0    15     1     0     0]
 [  222    12  1481   966   103     1   139    26     9     0]
 [  359    34  2179  3788   932     3  1004   158     9     0]
 [   64     9   311  1644  1551     5   746    93    28     2]
 [    5     0    70    94    38  9872    27     5     1     0]
 [   33     9    97  1686   968     7 11081    81     8     1]
 [   48     5   325   804   223     2   277   904     6     1]
 [    3     2    27    74   100     3    42     9    40     0]
 [    0     0     1    17     3     1     9     0     0     0]]

Classification Report:
                precision    recall  f1-score   support

      Analysis       0.09      0.15      0.11       499
      Backdoor       0.05      0.01      0.02       450
           DoS       0.30      0.50      0.37      2959
      Exploits       0.41      0.45      0.43      8466
       Fuzzers       0.39      0.35      0.37      4453
       Generic       1.00      0.98      0.99     10112
        Normal       0.83      0.79      0.81     13971
Reconnaissance       0.71      0.35      0.47      2595
     Shellcode       0.40      0.13      0.20       300
         Worms       0.00      0.00      0.00        31

      accuracy                           0.66     43836
     macro avg       0.42      0.37      0.38     43836
  weighted avg       0.68      0.66      0.66     43836
```

Rys. 5.7. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 2

5.3. Model Naive Bayes Classifier

Kod programu napisanego dla modelu GaussianNB:

```
from sklearn.naive_bayes import GaussianNB

#Inicjalizacja i trenowanie modelu NB

rf_model = GaussianNB()

rf_model.fit(X_trainf, y_trainf['attack_cat'])


#Inicjalizacja i trenowanie modelu NB

rf_model2 = GaussianNB()

rf_model2.fit(X_trainff, y_trainff['attack_cat'])
```

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model GaussianNB poradził sobie bardzo źle, ponieważ dokładność dwa razy nie przekroczyła nawet 50%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: normal, exploits.

```
Confusion Matrix:
[[    0     0     0     0     2   141     0    13     7     0]
 [    0     0     0     0     0   136     1    13     3     0]
 [    0     0     6    19    13   744    14   176    11     2]
 [    0     0     9    86    39   879   293  1491    17    14]
 [    0     0     0     3   162   495     0   848     7     0]
 [    0     0     2     2    12  4563    25    91     0     0]
 [    0     0    25    32   271  1676  3198  4010     3    36]
 [    0     0     0     0     2   450     3   431     2     0]
 [    0     0     0     0     6    46     0    41     2     0]
 [    0     0     0     0     0     2     1     6     0     1]]
```

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	163
Backdoor	0.00	0.00	0.00	153
DoS	0.14	0.01	0.01	985
Exploits	0.61	0.03	0.06	2828
Fuzzers	0.32	0.11	0.16	1515
Generic	0.50	0.97	0.66	4695
Normal	0.90	0.35	0.50	9251
Reconnaissance	0.06	0.49	0.11	888
Shellcode	0.04	0.02	0.03	95
Worms	0.02	0.10	0.03	10
accuracy			0.41	20583
macro avg	0.26	0.21	0.16	20583
weighted avg	0.64	0.41	0.40	20583

Rys. 5.8. i 5.9. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 1

```
Confusion Matrix:
[[    0     2     0     0    50   346     0    91    10     0]
 [    0     0     0     2    19   360     3    58     8     0]
 [    0     5     7    39   155  2335    37   292    83     6]
 [    0    13     8   168  1455  3374   701  2599   114    34]
 [    0     1     0    32  1458  1290     0  1655    17     0]
 [    0     2     4     6    47  9955    37    57     3     1]
 [    0     0    15    12  1610  1922  7395  2927     1    89]
 [    0    55     1     2   390  1309     2   820    16     0]
 [    0     0     0     0    71   141     0    86     2     0]
 [    0     0     0     0     9     4     3    12     0     3]]
```

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	499
Backdoor	0.00	0.00	0.00	450
DoS	0.20	0.00	0.00	2959
Exploits	0.64	0.02	0.04	8466
Fuzzers	0.28	0.33	0.30	4453
Generic	0.47	0.98	0.64	10112
Normal	0.90	0.53	0.67	13971
Reconnaissance	0.10	0.32	0.15	2595
Shellcode	0.01	0.01	0.01	300
Worms	0.02	0.10	0.04	31
accuracy			0.45	43836
macro avg	0.26	0.23	0.18	43836
weighted avg	0.57	0.45	0.41	43836

Rys. 5.10. i 5.11. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 2

5.4. Model Decision Tree Classifier

Kod programu napisanego dla modelu DTC:

```
from sklearn.tree import DecisionTreeClassifier
#Inicjalizacja i trenowanie modelu DTC
rf_model = DecisionTreeClassifier(random_state=42)
rf_model.fit(X_trainf, y_trainf['attack_cat'])


#Inicjalizacja i trenowanie modelu DTC
rf_model2 = DecisionTreeClassifier(random_state=42)
rf_model2.fit(X_trainff, y_trainff['attack_cat'])
```

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model DTC poradził sobie zadowalająco, ponieważ dokładność dwa razy przekroczyła 80%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: general, exploits, normal, reconnaissance, exploits.

```
Confusion Matrix:
[[  10    2   41   69   40    1    0    0    0    0]
 [   2    4   19   85   39    0    2    1    1    0]
 [   5    3  517  351   54   11   33    4    6    1]
 [  11   15  600 1808  175   55  100   52    9    3]
 [  10    7   73  216  664   10  522    4    8    1]
 [   1    1   22   62    5 4588   11    3    1    1]
 [   0    2   31   87  547    9 8527   19   28    1]
 [   0    8   74   82    5    1   15  701    2    0]
 [   0    1    7   12    9    3   33    2   27    1]
 [   0    0    2    4    0    1    1    0    0    2]]

Classification Report:
                precision    recall  f1-score   support

      Analysis       0.26      0.06      0.10       163
      Backdoor       0.09      0.03      0.04       153
           DoS       0.37      0.52      0.44       985
      Exploits       0.65      0.64      0.65      2828
      Fuzzers       0.43      0.44      0.43      1515
      Generic       0.98      0.98      0.98      4695
       Normal       0.92      0.92      0.92      9251
Reconnaissance       0.89      0.79      0.84       888
     Shellcode       0.33      0.28      0.31        95
         Worms       0.20      0.20      0.20        10

      accuracy                           0.82     20583
     macro avg       0.51      0.49      0.49     20583
  weighted avg       0.82      0.82      0.82     20583
```

```
Confusion Matrix:
[[   85     6    66   303     3     0    36     0     0     0]
 [    2    55    67   307     4     0     2     8     4     1]
 [    3     2   541  2309    28    10    38    12    16     0]
 [   16    13  1023  6880   108    44   136   212    23    11]
 [    4    10    93   372  2998     7   914     3    51     1]
 [    0     2    44   119    10  9918    10     2     5     2]
 [   38     3    34   162  1066     4 12615    19    30     0]
 [    0     6    98   548     3     2    10  1924     3     1]
 [    0     0    14    27    69     1    51     2   136     0]
 [    0     1     3    15     1     0     0     0     1    10]]

Classification Report:
                precision    recall  f1-score   support

      Analysis       0.57      0.17      0.26       499
      Backdoor       0.56      0.12      0.20       450
           DoS       0.27      0.18      0.22      2959
      Exploits       0.62      0.81      0.71      8466
      Fuzzers       0.70      0.67      0.69      4453
      Generic       0.99      0.98      0.99     10112
       Normal       0.91      0.90      0.91     13971
Reconnaissance       0.88      0.74      0.81      2595
     Shellcode       0.51      0.45      0.48       300
         Worms       0.38      0.32      0.35        31

      accuracy                           0.80     43836
     macro avg       0.64      0.54      0.56     43836
  weighted avg       0.80      0.80      0.79     43836
```

Rys. 5.12. i 5.13. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 1 i 2

5.5. Model Multi-layer Perceptron Classifier

Kod programu napisanego dla modelu MLPC:

```python
from sklearn.neural_network import MLPClassifier
#Inicjalizacja i trenowanie modelu MLPC
rf_model = MLPClassifier(random_state=42)
rf_model.fit(X_trainf, y_trainf['attack_cat'])


#Inicjalizacja i trenowanie modelu MLPC
rf_model2 = MLPClassifier(random_state=42)
rf_model2.fit(X_trainff, y_trainff['attack_cat'])
```

Oto efekty przeprowadzonego procesu uczenia dla rozpoznania rodzaju ataku dla dwóch plików. Na podstawie confussion matrix oraz raportów klasyfikacji można zauważyć, że w tym przypadku model MLPC nie poradził sobie zadowalająco, ponieważ dokładność dwa razy nie przekroczyła 55%. Niestety precyzja jest zadowalająca tylko dla kilku rodzajów ataków takich jak: general, normal, fuzzers.

```
Confusion Matrix:
[[   0    2    1    0    8  143    4    5    0    0]
 [   0    3    4    5    7  125    7    2    0    0]
 [   0   20   20   36  132  638   88   51    0    0]
 [   0   17   15   71  845  693  742  445    0    0]
 [   0    6    6    4  617  368  299  215    0    0]
 [   0    2    0   11   64 4532   56   30    0    0]
 [   0    1    3   18 2583  804 5002  840    0    0]
 [   0    6    1    7  230  406  135  103    0    0]
 [   0    0    0    0   59   15   13    8    0    0]
 [   0    0    0    0    4    1    2    3    0    0]]
```

```
                precision    recall  f1-score   support

      Analysis       0.00      0.00      0.00       163
      Backdoor       0.05      0.02      0.03       153
           DoS       0.40      0.02      0.04       985
      Exploits       0.47      0.03      0.05      2828
      Fuzzers       0.14      0.41      0.20      1515
      Generic       0.59      0.97      0.73      4695
       Normal       0.79      0.54      0.64      9251
Reconnaissance       0.06      0.12      0.08       888
     Shellcode       0.00      0.00      0.00        95
         Worms       0.00      0.00      0.00        10

      accuracy                           0.50     20583
     macro avg       0.25      0.21      0.18     20583
  weighted avg       0.58      0.50      0.48     20583
```

Rys. 5.14. i 5.15. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 1

```
Confusion Matrix:
[[    0     0    13   347     1    85    36    17     0     0]
 [    0     0    12   345     1    68    18     6     0     0]
 [    0     7    98  2177     4   489   157    27     0     0]
 [    0    10   687  3637    17  2582  1238   295     0     0]
 [    4     8   398   795    55  1799  1132   262     0     0]
 [    0     0    14    81     0  9952    59     6     0     0]
 [    2     1   505   644     7  2294 10273   245     0     0]
 [    0     1   177   581     0  1503   215   118     0     0]
 [    0     0    19   102     0    98    64    17     0     0]
 [    0     0     3     4     0    15     7     2     0     0]]
```

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	499
Backdoor	0.00	0.00	0.00	450
DoS	0.05	0.03	0.04	2959
Exploits	0.42	0.43	0.42	8466
Fuzzers	0.65	0.01	0.02	4453
Generic	0.53	0.98	0.69	10112
Normal	0.78	0.74	0.76	13971
Reconnaissance	0.12	0.05	0.07	2595
Shellcode	0.00	0.00	0.00	300
Worms	0.00	0.00	0.00	31
accuracy			0.55	43836
macro avg	0.25	0.22	0.20	43836
weighted avg	0.53	0.55	0.49	43836

Rys. 5.16. i 5.17. Przedstawienie wyników dla modelu rozpoznającego rodzaj ataku dla pliku 2

6. Zastosowanie autoenkodera

Autoenkoder to rodzaj sieci neuronowej, która jest trenowana do kopiowania swoich danych wejściowych na wyjście. Składa się z dwóch głównych części:

- **Enkoder**: Przekształca dane wejściowe na niższy wymiar (reprezentację latentną), kompresując informacje.
- **Dekoder**: Rekonstruuje oryginalne dane wejściowe z reprezentacji latentnej.

Podany poniżej kod buduje i trenuje autoenkoder na danych sieciowych w celu kompresji i rekonstrukcji danych. Pierwszym krokiem był import bibliotek.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from sklearn.preprocessing import StandardScaler
from sklearn.model selection import train test split
from sklearn.metrics import mean_squared_error
```

Wczytanie danych z pliku ‘parquet’.

```
#Wczytanie danych treningowych
dftrain = pd.read_parquet('/content/drive/MyDrive/Colab
Notebooks/UNSW_NB15_training-set.parquet')
```

Przygotowanie cech poprzez usunięcie kolumn ‘label’ oraz ‘attack_cat’ w celu utworzenia macierzy cech ‘X_train’.

```
#Usunięcie kolumn 'label' i 'attack_cat' z danych treningowych
X_train = dftrain.drop(['label', 'attack_cat'], axis=1)
```

Przekształcenie z.miennych kategorycznych na zmienne binarne za pomocą kodowania one-hot.

```
#Przekształcenie zmiennych kategorycznych za pomocą kodowania one-hot
X_train_encoded = pd.get_dummies(X_train)
```

Podział danych na zbiory treningowe i testowe w stosunku 75%/25%.

```
#Podział danych na zbiory treningowe i testowe
X_trainf, X_testf = train_test_split(X_train_encoded, test_size=0.25,
random_state=42)
```

Skalowanie danych aby miały średnią 0 i odchylenie standardowe 1.

```
#Skalowanie danych
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_trainf)
X_test_scaled = scaler.transform(X_testf)
```

Ustawienie parametrów modelu czyli wymiaru wejściowego i wymiaru warstwy latentnej.

```
#Parametry modelu
input_dim = X_train_scaled.shape[1]
encoding_dim = 32  #Rozmiar warstwy latentnej
```

Budowa modelu autoenkodera, czyli definicja warstw enkodera i dekodera oraz stworzenie modeli: autoenkodera, enkodera i dekodera.

```
#Budowa modelu autoenkodera
input_layer = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_layer)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_layer, decoded)
encoder = Model(input_layer, encoded)

encoded input = Input(shape=(encoding dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))
```

Kompilacja modelu autoenkodera z optymalizatorem ‘adam’ I funkcją straty ‘mse’.

```
#Kompilacja modelu
autoencoder.compile(optimizer='adam', loss='mse')
```

Trenowanie modelu na danych treningowych z walidacją na danych testowych.

```
#Trenowanie modelu
autoencoder.fit(X_train_scaled, X_train_scaled,
                epochs=50,
                batch_size=256,
                shuffle=True,
```

```
                validation_data=(X_test_scaled, X_test_scaled))
```

Użycie enkodera do kodowania danych i dekodera do rekonstrukcji danych z reprezentacji latentnej.

```
#Wykorzystanie enkodera do kodowania danych
encoded_train = encoder.predict(X_train_scaled)
encoded_test = encoder.predict(X_test_scaled)

#Wykorzystanie dekodera do dekodowania danych
decoded_train = decoder.predict(encoded_train)
decoded_test = decoder.predict(encoded_test)
```

Obliczenie średniego kwadratu błędu (MSE) dla danych treningowych i testowych oraz wypisanie wyników.

```
#Ocena jakości rekonstrukcji
train_mse = mean_squared_error(X_train_scaled, decoded_train)
test_mse = mean_squared_error(X_test_scaled, decoded_test)

print(f'Train MSE: {train_mse}')
print(f'Test MSE: {test_mse}')
```

Ten kod umożliwia occnę, jak dobrze autoencoder nauczył się kompresować i rekonstruować dane sieciowe, co może być użyteczne w wykrywaniu anomalii.

```
Train MSE: 0.8910008500806065
Test MSE: 0.9287464426350197
```

Rys. 6.1. Wyniki oceny rekonstrukcji autoenkodera dla pliku 1

```
Train MSE: 0.8749031987007986
Test MSE: 0.8407633102476777
```

Rys. 6.2. Wyniki oceny rekonstrukcji autoenkodera dla pliku 2

- Train MSE (0.8910) jest nieco niższy niż Test MSE (0.9287). To może sugerować, że model jest dobrze dopasowany do danych treningowych, ale nieco gorzej radzi sobie z danymi testowymi, co może być oznaką lekkiego przeuczenia.

- Train MSE (0.8740) jest nieco niższy niż Test MSE (0.8407). W tym przypadku, zarówno wartości Train MSE, jak i Test MSE są niższe niż w przypadku Pliku 1, co może wskazywać na lepszą jakość rekonstrukcji.

MSE w granicach 0.8 - 0.9 może być uznane za umiarkowanie dobre, jednak ostateczna ocena zależy od specyfiki danych i kontekstu zastosowania. W przypadku wykrywania anomalii, niższe wartości MSE są zawsze pożądane, ponieważ sugerują bardziej precyzyjne odwzorowanie normalnych danych.

7. Zastosowanie VAEs'a oraz API OpenAI

VAEs (Wariacyjne Autoenkodery) to generatywne modele probabilistyczne, które uczą się struktury danych poprzez zakodowanie ich w zmienne latentne (ukryte) i dekodowanie ich z powrotem do oryginalnych danych.

VAEs mogą być używane do detekcji anomalii, ponieważ są uczone na danych normalnych i stają się bardzo efektywne w ich rekonstrukcji. Anomalie, które znacznie odbiegają od normalnych danych, będą miały wyższe błędy rekonstrukcji, co pozwala na ich wykrycie poprzez analizę wartości błędu rekonstrukcji (np. Mean Squared Error).

Z zastosowaniem VAES wiązały się komplikacje i jego implementacja nie przebiegła pomyślnie. Wpływ na to może mieć forma zbioru danych i jego poszczególnych wartości.

OpenAI API, z modelem GPT-3 lub nowszym, może być używane do detekcji anomalii w danych poprzez analizę kontekstową i klasyfikację tekstową. Niestety wykorzystanie OpenAI API wykazało problem dla typowego użytkownika, ponieważ dostęp do takiego limitu słów dla własnego API jest niemożliwy bez wykupienia odpowiedniej opcji.

Podany poniżej kod wykonuję detekcję anomalii w danych dotyczących ruchu sieciowego przy użyciu OpenAI API, w szczególności modelu "gpt-3.5-turbo". Na początku następuje import potrzebnych bibliotek.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import openai
```

Wczytanie danych z pliku '.parquet' zawierającego dane ruchu sieciowego.

```
#Wczytanie danych
df = pd.read_parquet('/content/drive/MyDrive/Colab
Notebooks/UNSW_NB15_training-set.parquet')
```

Wybór cech i etykiet poprzez usunięcie kolumn 'label' i 'attack_cat' z danych, 'attack_cat' jest używany jako etykieta do klasyfikacji.

```
#Wybór cech i etykiet
X = df.drop(['label', 'attack_cat'], axis=1)
y = df['attack_cat']
```

Przekształcenie zmiennych kategorycznych za pomocą kodowania one-hot, co pozwala na przekształcenie danych kategorycznych na formę numeryczną.

```
#Przekształcenie zmiennych kategorycznych za pomocą kodowania one-hot
X_encoded = pd.get_dummies(X)
```

Podział danych na zbiory treningowe (75%) i testowe (25%).

```
#Podział danych na zbiory treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
test_size=0.25, random_state=42)
```

Skalownie danych co zapewnia, że wszystkie cechy mają średnią 0 i odchylenie standardowe 1.

```
#Skalowanie danych
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Ustawienie klucza API OpenAI, który jest niezbędny do komunikacji z OpenAI API.

```
#Ustawienie klucza API OpenAI
openai.api_key = 'sk-proj-MKPIAaMWcZusErGfeTaFT3BlbkFJ4A0fNGDbavKCRKfkDGno'
```

Funkcja tworząca prompty, które będą używane do trenowania i testowania modelu OpenAI.

```
#Funkcja do tworzenia promptów
def create_prompt(sample, label=None):
    sample_text = ", ".join(map(str, sample))
    prompt = f"Given the following network traffic data: {sample_text}\nIs this
traffic normal or anomalous?"
    if label:
        prompt += f" It is {label}."
    return prompt
```

Przygotowanie promptów dla danych treningowych z uwzględnieniem etykiet.

```
#Przygotowanie promptów treningowych (few-shot learning)
train_prompts = []
for i in range(len(X_train_scaled)):
    sample = X_train_scaled[i]
    label = y_train.iloc[i]
    prompt = create_prompt(sample, label)
    train_prompts.append(prompt)
```

Przygotowanie promptów dla danych testowych bez etykiet.

```
#Przygotowanie promptów testowych
test_prompts = []
for i in range(len(X_test_scaled)):
    sample = X_test_scaled[i]
    prompt = create_prompt(sample)
    test_prompts.append(prompt)
```

Klasyfikacja danych testowych poprzez wysyłanie promptów testowych do OpenAI API I uzyskiwanie odpowiedzi dotyczących klasyfikacji ruchu sieciowego jako normalnego lub abnormalnego.

```
#Klasyfikacja danych testowych
predictions = []
for prompt in test_prompts:
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}]
    )
    prediction = response['choices'][0]['message']['content'].strip().lower()
    if "normal" in prediction:
        predictions.append("Normal")
    else:
        predictions.append("Anomalous")
```

Ocena dokładności modelu i wygenerowanie raportu klasyfikacji.

```
#Ocena wyników
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions)

print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{report}")
```

Kod ten wczytuje dane, przetwarza je i klasyfikuje za pomocą modelu GPT-3.5, wysyłając dane sieciowe jako prompty do API OpenAI i oceniając wyniki na podstawie dokładności oraz raportu klasyfikacji.

```
---------------------------------------------------------------------------
RateLimitError                            Traceback (most recent call last)
<ipython-input-2-8732ac02b119> in <cell line: 54>()
     53 predictions = []
     54 for prompt in test_prompts:
---> 55     response = openai.ChatCompletion.create(
     56         model="gpt-3.5-turbo",
     57         messages=[{"role": "user", "content": prompt}]

                                  4 frames
/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py in _interpret_response_line(self, rbody, rcode, rheaders, stream)
    763         stream_error = stream and "error" in resp.data
    764         if stream_error or not 200 <= rcode < 300:
--> 765             raise self.handle_error_response(
    766                 rbody, rcode, resp.data, rheaders, stream_error=stream_error
    767             )

RateLimitError: You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.
```

Rys. 7.1. Komunikat zwrócony przez program

Błąd RateLimitError oznacza, że liczba zapytań wysłanych do API OpenAI przekroczyła ustalony limit. Limity te mogą być dzienne, miesięczne lub na minutę/godzinę, w zależności od planu subskrypcji lub ustawień konta.

8. Zastosowanie Gans'a (Generative Adversarial Network)

Generative Adversarial Networks (GANs) to klasa algorytmów uczenia maszynowego zaprojektowanych przez Iana Goodfellowa i jego współpracowników w 2014 roku. GANy składają się z dwóch sieci neuronowych: generatora i dyskryminatora, które rywalizują ze sobą w grze o sumie zerowej.

- **Generator**: Tworzy próbki danych, które próbują naśladować rzeczywiste dane.
- **Dyskryminator**: Stara się odróżnić wygenerowane przez generator próbki od rzeczywistych danych.

Proces treningowy polega na tym, że generator stara się oszukać dyskryminatora, generując coraz bardziej realistyczne dane, podczas gdy dyskryminator stara się poprawić swoje umiejętności odróżniania prawdziwych danych od fałszywych.

Pokazany poniżej kod implementuje Generative Adversarial Network (GAN) do generowania syntetycznych danych podobnych do danych o ruchu sieciowym z zestawu UNSW-NB15. Wykorzystuje on TensorFlow i Keras do budowy i trenowania modelu GAN, który składa się z dwóch głównych części: generatora i dyskryminatora. Pierwszym krokiem było zaimportowanie potrzebnych bibliotek oraz narzędzi do przetwarzania danych:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, LeakyReLU
from tensorflow.keras.models import Model
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Dane treningowe są wczytywane z pliku 'parquet', kod ogranicza się do pierwszych 50 000 wierszy, aby przyspieszyć trening.

```
#Wczytanie danych treningowych (wczytaj tylko część danych dla przyspieszenia)
dftrain = pd.read_parquet('/content/drive/MyDrive/Colab
Notebooks/UNSW_NB15_testing-set.parquet')
#Ograniczenie do pierwszych 50000 wierszy
dftrain = dftrain.head(50000)

#Usunięcie kolumn 'label' i 'attack_cat' z danych treningowych
X_train = dftrain.drop(['label', 'attack_cat'], axis=1)
```

Zmienne kategoryczne są przekształcane na postać numeryczną za pomocą one-hot encoding a następnie dzielone na zbiór treningowy (75%) oraz zbiór testowy (25%).

```
#Przekształcenie zmiennych kategorycznych za pomocą kodowania one-hot
X_train_encoded = pd.get_dummies(X_train)

#Podział danych na zbiory treningowe i testowe
X_trainf, X_testf = train_test_split(X_train_encoded, test_size=0.25,
random_state=42)
```

Dane są skalowane przy użyciu ‘StandardScaler’.

```
#Skalowanie danych
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_trainf)
X_test_scaled = scaler.transform(X_testf)
```

Definicja modelu generatora, generator tworzy próbki danych z losowych wektorów (latent space). Składa się z warstw Dense i LeakyReLU.

```
#Parametry modelu
input_dim = X_train_scaled.shape[1]
encoding_dim = 32  #Rozmiar warstwy latentnej

#Budowa modelu generatora
generator_input = Input(shape=(encoding_dim,))
x = Dense(64)(generator_input)
x = LeakyReLU(alpha=0.2)(x)
generator_output = Dense(input_dim, activation='sigmoid')(x)

generator = Model(generator_input, generator_output)
```

Definicja modelu dyskryminatora, dyskryminator stara się odróżnić wygenerowane przez generator dane od rzeczywistych, także składa się z warstw Dense oraz LeakyReLU.

```
#Budowa modelu dyskryminatora
discriminator_input = Input(shape=(input_dim,))
y = Dense(64)(discriminator_input)
y = LeakyReLU(alpha=0.2)(y)
y = Dense(1, activation='sigmoid')(y)

discriminator = Model(discriminator input, y)
discriminator.compile(optimizer='adam', loss='binary_crossentropy')  #Dodana
kompilacja
```

Kompilacja modelu GAN, łączenie generatora i dyskryminatora, ustawiono dyskryminator jako nieuczący się podczas trenowania GAN.

```
#Kompilacja modelu generatora
discriminator.trainable = False
gan_input = Input(shape=(encoding_dim,))
gan output = discriminator(generator(gan input))
gan = Model(gan_input, gan_output)
gan.compile(optimizer='adam', loss='binary_crossentropy')
```

Trenowanie modelu GAN, w każdej epoce generator tworzy fałszywe dane, a dyskryminator trenuje się na rzeczywistych i wygenerowanych danych, następnie GAN trenuje generator.

```
#Trenowanie GAN
epochs = 50
batch_size = 256

for epoch in range(epochs):
    for _ in range(len(X_train_scaled) // batch_size):
        #Generowanie losowych punktów latentnych
        noise = np.random.normal(0, 1, size=(batch_size, encoding_dim))

        #Generowanie danych fałszywych z generatora
        generated_data = generator.predict(noise)

        #Losowanie próbek danych rzeczywistych
        idx = np.random.randint(0, X_train_scaled.shape[0], batch_size)
        real_data = X_train_scaled[idx]

        #Tworzenie zbioru treningowego dla dyskryminatora
        X = np.concatenate([real_data, generated_data])
        y_dis = np.zeros(2 * batch_size)
        y_dis[:batch_size] = 1  #Oznaczenie danych rzeczywistych jako 1

        #Trenowanie dyskryminatora
        discriminator.trainable = True
        d_loss = discriminator.train_on_batch(X, y_dis)

        #Tworzenie zbioru treningowego dla GAN
        noise = np.random.normal(0, 1, size=(batch_size, encoding_dim))
        y_gan = np.ones(batch_size)  #Oznaczenie wygenerowanych danych jako 1

        #Trenowanie GAN
        discriminator.trainable = False
        g_loss = gan.train_on_batch(noise, y_gan)

    print(f'Epoch: {epoch + 1}, Discriminator Loss: {d_loss}, Generator Loss:
{g_loss}')
```

Generowanie nowych danych i ocena jakości, generator jest używany do tworzenia nowych danych, które są następnie oceniane pod kątem jakości przy użyciu Mean Squared Error (MSE).

```
#Wykorzystanie generatora do generowania danych
generated_data = generator.predict(np.random.normal(0, 1,
size=(len(X_test_scaled), encoding_dim)))
#Ocena jakości generowanych danych
mse = mean_squared_error(X_test_scaled, generated_data)
print(f'Test MSE: {mse}')
```

Proces obejmuje przygotowanie danych, zbudowanie i trenowanie modelu GAN oraz ocenę jakości wygenerowanych danych przy użyciu MSE. GAN składa się z dwóch modeli (generatora i dyskryminatora), które współzawodniczą, aby generator nauczył się tworzyć realistyczne dane.

```
Epoch: 50, Discriminator Loss: 1.1817468475783244e-05, Generator Loss: 10.79804515838623
391/391 [==============================] - 1s 2ms/step
Test MSE: 1.0381340152879173
```

Rys. 8.1. Wyniki MSE Gans dla pliku 1

```
Epoch: 50, Discriminator Loss: 2.002749897656031e-05, Generator Loss: 10.310808181762695
391/391 [==============================] - 1s 3ms/step
Test MSE: 0.7743150408345608
```

Rys. 8.2. Wyniki MSE Gans dla pliku 2

Discriminator Loss:

W obu przypadkach loss dyskryminatora jest bardzo niski (rzędu e-05). Niski loss dyskryminatora wskazuje, że dyskryminator jest bardzo dobry w rozróżnianiu między rzeczywistymi a wygenerowanymi danymi.

Generator Loss:

Loss generatora jest stosunkowo wysoki w obu przypadkach (około 10.3 do 10.8). Wysoki loss generatora może wskazywać, że generator ma trudności z oszukiwaniem dyskryminatora. Jednakże, w przypadku GAN, wartości loss nie zawsze są bezpośrednio wskazujące na wydajność modelu, ze względu na dynamiczną naturę procesu treningowego.

Plik 2 uzyskał niższy Test MSE (0.774) w porównaniu do Pliku 1 (1.038). Oznacza to, że model dla Pliku 2 lepiej odtworzył dane testowe niż model dla Pliku 1. Wyniki MSE poniżej 1.0 są generalnie dobre, zwłaszcza jeśli bierzemy pod uwagę złożoność danych sieciowych.

Wnioski

W projekcie zastosowano różne metody AI, w tym autoenkodery, VAE, GAN, modele klasyczne oraz API OpenAI, do wykrywania anomalii w ruchu sieciowym, co pozwoliło na ocenę ich skuteczności i porównanie wyników. Autoenkodery wykazały umiarkowaną skuteczność w rekonstrukcji danych, co sugeruje, że mogą być użyteczne w detekcji anomalii, ale wymagają dalszej optymalizacji. Zbiór danych nie był wystarczająco przystosowany do VAE, natomiast GAN pokazał mieszane rezultaty, dyskryminatory były bardzo skuteczne w rozróżnianiu danych, ale generatory miały trudności z oszukiwaniem dyskryminatorów, co wskazuje na konieczność dalszej regulacji hiperparametrów. Zwykłe modele uczenia maszynowego, takie jak DecisionTree czy RandomForest, również zostały zastosowane i wykazały dobrą skuteczność w klasyfikacji anomalii, ale ich wyniki zależą w dużym stopniu od jakości i ilości danych treningowych oraz właściwej inżynierii cech. Wyniki uzyskane za pomocą API OpenAI były bardzo mocno ograniczone przez limity API, co wskazuje na konieczność bardziej efektywnego zarządzania zasobami lub wyboru alternatywnych podejść w przyszłych badaniach. Ogólnie, wyniki pokazują, że każdy z zastosowanych algorytmów ma swoje mocne i słabe strony, co sugeruje potrzebę dalszych badań i optymalizacji. Projekt wykazał, że AI, w tym zarówno zaawansowane modele jak autoenkodery, VAE i GAN-y, jak i tradycyjne modele uczenia maszynowego, może być skutecznie wykorzystane do wykrywania anomalii w ruchu sieciowym, ale wymagają odpowiedniego dostosowania i zrozumienia specyfiki każdego z podejść.

Literatura

- **Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014).** Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 27, 2672-2680.
- **Kingma, D.P., & Welling, M. (2014).** Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*.
- **Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986).** Learning representations by back-propagating errors. *Nature*, 323, 533-536.
- **Chalapathy, R., & Chawla, S. (2019).** Deep Learning for Anomaly Detection: A Survey. *arXiv preprint arXiv:1901.03407*.
- **Hinton, G.E., & Salakhutdinov, R.R. (2006).** Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786), 504-507.
- **Aggarwal, C.C. (2013).** Outlier Analysis. *Springer*.
- **Breunig, M.M., Kriegel, H.P., Ng, R.T., & Sander, J. (2000).** LOF: Identifying Density-Based Local Outliers. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 93-104.
- **Vapnik, V.N. (1995).** The Nature of Statistical Learning Theory. *Springer*.
- **Breiman, L. (2001).** Random Forests. *Machine Learning*, 45(1), 5-32.
- **Bishop, C.M. (2006).** Pattern Recognition and Machine Learning. *Springer*.

Dokumentacja i narzędzia

- **OpenAI.** API Reference. Dostęp: https://platform.openai.com/docs/api-reference.
- **TensorFlow.** TensorFlow Documentation. Dostęp: https://www.tensorflow.org/.
- **Scikit-learn.** Scikit-learn Documentation. Dostęp: https://scikit-learn.org/stable/.
- **Pandas.** Pandas Documentation. Dostęp: https://pandas.pydata.org/.
- **Numpy.** Numpy Documentation. Dostęp: https://numpy.org/.