

Analiza danych strumieniowych z użyciem uczenia maszynowego

Problemy rozwiązywane z użyciem uczenia maszynowego:

1) Rozpoznawanie ataków zewnętrznych (neptune) w ruchu sieciowym TCP/IP

Dane historyczne są danymi rzeczywistymi i zostały pobrane ze strony:

<https://www.kaggle.com/datasets/anushonkar/network-anomaly-detection/data?select=Network+Anomaly+Detection.docx> i dotyczą ruchu TCP/IP.

Ruch TCP/IP, czyli Transmission Control Protocol/Internet Protocol to zbiór protokołów komunikacyjnych wykorzystywanych do przesyłania danych w sieciach komputerowych, w tym w globalnej sieci - Internet. Ruch TCP/IP jest bardzo narażony na ataki - może być przechwytywany przez atakujących, przez co dochodzi często do wycieku poufnych informacji.

Atak Neptune

Atak sieciowy Neptune to odmiana ataku typu DoS (Denial of Service), który ma na celu przeciążenie serwera i uniemożliwienie jego normalnej pracy. W szczególności atak Neptune jest rodzajem ataku SYN flood, w którym napastnik wysyła dużą liczbę pakietów SYN do serwera, inicjując proces otwierania połączenia TCP, ale nigdy go nie kończąc.

2) Przewidywanie zużycia energii przez urządzenie

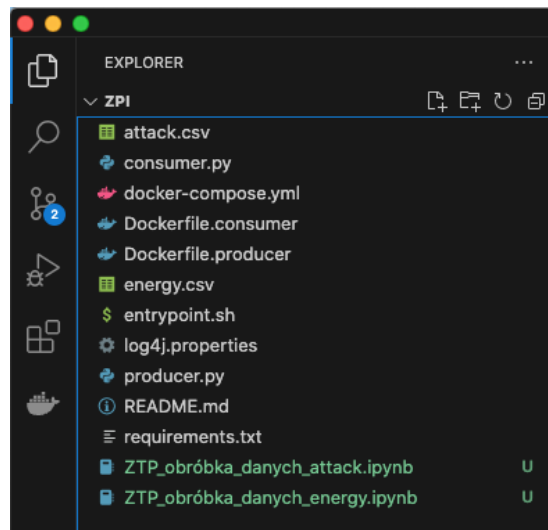
Dane historyczne są danymi rzeczywistymi i zostały pobrane ze strony:

<https://www.kaggle.com/datasets/sohommajumder21/appliances-energy-prediction-data-set> i zużycia energii przez urządzenia w budynku niskoenergetycznym.

Obróbka danych

W pierwszym przypadku skupiłam się na rozpoznawaniu ataku "neptune" oraz na ruchu normalnym - bez anomalii. W tym celu pozbyłam się rekordów, które dotyczyły innych ataków. Dla ruchu normalnego było o wiele więcej wierszy, więc dokonałam zbalansowania tej klasy. Kolejną obróbką w obu bazach danych było usunięcie duplikatów i pustych rekordów. Następną obróbką w projekcie było zredukowanie znacznej ilości kolumn, które odznaczały się słabą entropią, a na koniec dokonałam normalizacji danych. Za pomocą Google Colab wygenerowałam pliki attack.csv oraz energy.csv.

Budowa projektu (cały system działa jako obrazy dockerowe)



Budowanie modelu jest możliwe albo z zadaniem harmonogramem (co 20s) albo na żądanie.

```
$ entrypoint.sh ×
$ entrypoint.sh
1  #!/bin/sh
2
3  # Ścieżka do pliku wskaźnikowego
4  FLAG_FILE=/app/.first_run_completed
5
6  # Sprawdzenie zmiennej środowiskowej DELAY_SECONDS
7  if [ ! -f "$FLAG_FILE" ]; then
8      echo "Pierwsze uruchomienie, brak opóźnienia."
9      touch "$FLAG_FILE"
10 else
11     if [ -n "$DELAY_SECONDS" ]; then
12         echo "Opóźnienie uruchomienia o $DELAY_SECONDS sekund..."
13         sleep $DELAY_SECONDS
14     fi
15 fi
16
17 # Uruchomienie głównej aplikacji
18 exec python consumer.py
19
```

```
consumer:
  build:
    context: .
    dockerfile: Dockerfile.consumer
  depends_on:
    - kafka
  environment:
    - KAFKA_BROKER=kafka:9092
    - MODEL_ATTACK=true
    - MODEL_ENERGY=false
    - DELAY_SECONDS=20 # Opóźnienie
  restart: always
```

producer.py - generuje dane dla systemu Kafka (pobiera dane i wysyła strumieniowe)

Pobieranie danych historycznych rzeczywistych z pliku .csv

```
def generate_data_attack(start_row=0):
    label_mapping = {'neptune': 1, 'normal': 0}
    with open('attack.csv', 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        for _ in range(start_row):
            next(csvreader)
        for row in csvreader:
            X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11 = row
            feature_11_numeric = label_mapping.get(X11, 0)
            data = {
                "feature_1": float(X1),
                "feature_2": float(X2),
                "feature_3": float(X3),
                "feature_4": float(X4),
                "feature_5": float(X5),
                "feature_6": float(X6),
                "feature_7": float(X7),
                "feature_8": float(X8),
                "feature_9": float(X9),
                "feature_10": float(X10),
                "feature_11": feature_11_numeric
            }
            return json.dumps(data)

def generate_data_energy(start_row=0):
    with open('energy.csv', 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        for _ in range(start_row):
            next(csvreader)
        for row in csvreader:
            X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14, X15, X16 = row

            data = {
                "feature_2": float(X2),
                "feature_3": float(X3),
                "feature_4": float(X4),
                "feature_5": float(X5),
                "feature_6": float(X6),
                "feature_7": float(X7),
                "feature_8": float(X8),
                "feature_9": float(X9),
                "feature_10": float(X10),
                "feature_11": float(X11),
                "feature_12": float(X12),
                "feature_13": float(X13),
                "feature_14": float(X14),
                "feature_15": float(X15),
                "feature_16": float(X16)
            }
            return json.dumps(data)
```

Ciągłe wysyłanie wiadomości do Kafki

```
#Tworzy obiekt producenta Kafka (Producer) zdefiniowany wcześniej konfiguracją.
p = Producer(conf)

#Główna pętla
#symuluje ciągłą generację danych i wysyłanie ich do Kafki za pomocą producenta
start_row1 = 1
while True:
    try:
        print("Producent: generuję nowe dane", flush=True)
        print(start_row1, flush=True)
        #Generuje nowe dane

        if (os.getenv('MODEL_ENERGY', 'false').lower() == 'true'):
            activity = generate_data_energy(start_row1)
            start_row1 = start_row1 + 1
            #Wysyła wygenerowane dane do tematu Kafka o nazwie 'historic_data'
            p.produce('historic_data', activity)
            #Wywołuje p.flush() w celu zapewnienia wysłania wszystkich wiadomości do brokera Kafka.
            p.flush()

        elif (os.getenv('MODEL_ATTACK', 'false').lower() == 'true'):
            activity = generate_data_attack(start_row1)
            start_row1 = start_row1 + 1
            #Wysyła wygenerowane dane do tematu Kafka o nazwie 'historic_data'
            p.produce('historic_data', activity)
            #Wywołuje p.flush() w celu zapewnienia wysłania wszystkich wiadomości do brokera Kafka.
            p.flush()

        #Oczekuje przez sekundę przed ponownym wykonaniem pętli.
        time.sleep(1)
    #W przypadku wystąpienia błędu, wyświetla komunikat i czeka 3 sekundy przed ponownym wykonaniem pętli.
    except Exception as e:
        print(f"Błąd: {e}")
        time.sleep(3)
```

consumer.py

Funkcje związane z odbieraniem wiadomości strumieniowo:

```
22
23     time.sleep(100)
24     #print("KONSUMENT")
25
26     conf = {
27         'bootstrap.servers': 'kafka:9092',
28         'group.id': 'mygroup',
29         'auto.offset.reset': 'earliest',
30         'client.id': socket.gethostname()
31     }
32
33     #pobiera najnowsze dostępne offsety dla wszystkich partycji określonego tematu w Kafka, co pozwala na monitorowanie, jakie wiadomości są najnowsze w danym temacie.
34     def get_latest_offsets(consumer, topic):
35         #inicjalizacja słownika latest_offsets, będzie przechowywał najnowsze offsety dla każdej partycji
36         latest_offsets = {}
37         #pobiera listę tematów dostępnych w Kafka, a następnie wybiera temat podany jako argument (topic)
38         partitions = consumer.list_topics(topic).topics[topic].partitions
39         #iteracja przez partycje: #iteruje przez wszystkie partycje tematu.
40         for partition in partitions:
41             #Dla każdej partycji tworzony jest obiekt TopicPartition, który łączy temat z partycją.
42             tp = TopicPartition(topic, partition)
43             #pobiera zakres offsetów (najniższy i najwyższy) dla danej partycji (tp).
44             #low to najniższy offset (pierwsza dostępna wiadomość).
45             #high to najwyższy offset (najnowsza wiadomość).
46             low, high = consumer.get_watermark_offsets(tp, timeout=10)
47             #Przypisanie wysokiego offsetu do partycji: Najwyższy offset (high) jest przypisywany do odpowiedniej partycji w słowniku latest_offsets.
48             latest_offsets[partition] = high
49         #Zwracanie słownika z najnowszymi offsetami:
50         return latest_offsets
51
```

consumer.py

```
51
52
53 #Funkcja consume_messages_to_latest jest odpowiedzialna za konsumowanie wiadomości z topiku Kafka do momentu osiągnięcia najnowszego
54 def consume_messages_to_latest():
55     #Tworzy instancję konsumenta Kafka za pomocą dostarczonej konfiguracji conf.
56     c = Consumer(conf)
57     #Subskrybuje konsumenta do określonego topiku.
58     topic = 'historic_data'
59     c.subscribe([topic])
60     #Inicjalizuje pustą listę do przechowywania skonsumowanych wiadomości.
61     messages = []
62
63     try:
64         #pobrać najnowsze offsety dla wszystkich partycji w topiku.
65         latest_offsets = get_latest_offsets(c, topic)
66
67         #Tworzy listę obiektów TopicPartition dla każdej partycji z offsetem początkowym równym 0.
68         partitions = [TopicPartition(topic, p, 0) for p in latest_offsets]
69         #Przypisuje konsumenta do określonych partycji.
70         c.assign(partitions)
71
72         #Rozpoczyna nieskończoną pętlę do konsumowania wiadomości.
73         while True:
74             #Pobiera wiadomość z topiku z czasem oczekiwania 1 sekundy.
75             msg = c.poll(timeout=1.0)
76             #Jeśli nie ma wiadomości, przechodzi do następnej iteracji.
77             if msg is None:
78                 continue
79             #Sprawdza, czy wiadomość zawiera błąd.
80             if msg.error():
81                 #Jeśli kod błędu to _PARTITION_EOF, oznacza to koniec partycji, więc przechodzi do następnej iteracji.
82                 if msg.error().code() == KafkaError._PARTITION_EOF:
83                     continue
84                 else:
85                     #W przeciwnym razie, rzuca wyjątek KafkaException.
86                     raise KafkaException(msg.error())
87
88             #Pobiera numer partycji wiadomości.
89             partition = msg.partition()
90             #Pobiera offset wiadomości.
91             offset = msg.offset()
92             #Dekoduje i dodaje wiadomość do listy messages.
93             messages.append(json.loads(msg.value().decode('utf-8')))
94
95             #Sprawdza, czy obecny offset jest równy najnowszemu offsetowi minus 1.
96             if offset == latest_offsets[partition] - 1:
97                 #suwa partycję z listy latest_offsets, jeśli osiągnięto najnowszy offset.
98                 del latest_offsets[partition]
99             #Jeśli lista latest_offsets jest pusta, przerywa pętlę.
100             if not latest_offsets:
101                 break
102
103     #Obsługuje wszelkie wyjątki, drukując komunikat o błędzie.
104     except Exception as e:
105         print(f"Błąd: {e}")
106     #Zamyka konsumenta niezależnie od wyniku operacji.
107     finally:
108         c.close()
109
110     #Zwraca listę skonsumowanych wiadomości.
111     return messages
112
113
```

Budowa modelu uczenia maszynowego przy użyciu DASK wraz z ewaluacją modelu

LinearRegression

```
#Budowanie modelu REGRESJI LINIOWEJ
def build_model_energy():
    try:
        messages = consume_messages_to_latest()
    except Exception as e:
        print(f"Błąd: {e}")

    #Dane z Kafki są najpierw wczytywane do Pandas DataFrame,
    df = pd.DataFrame(messages)
    # a następnie konwertowane do Dask DataFrame przy użyciu dd.from_pandas().
    # Dask DataFrame umożliwia rozproszone przetwarzanie dużych zbiorów danych,
    df = dd.from_pandas(pd.DataFrame(df), npartitions=8)

    # Wybieramy kolumny do X i y
    X = df[['feature_3', 'feature_4', 'feature_5', 'feature_6', 'feature_7', 'feature_8', 'feature_9']]
    y = df['feature_2']

    # Dzielimy dane na zestawy treningowe i testowe
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

    #Inicjalizujemy modelu regresji liniowej
    clf = LinearRegression()

    # Trenujemy model
    clf.fit(X_train.values.compute(), y_train.values.compute())

    # Przewidujemy wartości
    y_pred = clf.predict(X_test.values.compute())

    # Obliczamy metryki regresji
    mae = mean_absolute_error(y_test.compute(), y_pred)
    mse = mean_squared_error(y_test.compute(), y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test.compute(), y_pred)
    explained_variance = explained_variance_score(y_test.compute(), y_pred)
    mape = np.mean(np.abs((y_test.compute() - y_pred) / y_test.compute())) * 100

    print(f"Mean Absolute Error: {mae}")
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
    print(f"R-squared: {r2}")
    print(f"Explained Variance: {explained_variance}")
    print(f"Mean Absolute Percentage Error: {mape}")
    print(f"Ilość odebranych wiadomości: {len(df)}")
```

Model klasyfikacji - LogisticRegression

```
#Budowanie modelu KLASYFIKACJI
def build_model_attack():
    try:
        messages = consume_messages_to_latest()
    except Exception as e:
        print(f"Błąd: {e}")

    #Dane z Kafki są najpierw wczytywane do Pandas DataFrame,
    df = pd.DataFrame(messages)
    # a następnie konwertowane do Dask DataFrame przy użyciu dd.from_pandas().
    # Dask DataFrame umożliwia rozproszone przetwarzanie dużych zbiorów danych,
    df = dd.from_pandas(pd.DataFrame(df), npartitions=8)

    # Wybieramy kolumny do X i y
    X = df[['feature_1', 'feature_2', 'feature_3', 'feature_4', 'feature_5', 'feature_6', 'feature_7', 'feature_8', 'feature_9', 'feature_10']]
    y = df['feature_11']

    # Dzielimy dane na zestawy treningowe i testowe
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, shuffle=False)

    # Inicjalizujemy model klasyfikacyjny
    clf = LogisticRegression()

    # Trenujemy model
    clf.fit(X_train.values.compute(), y_train.values.compute())

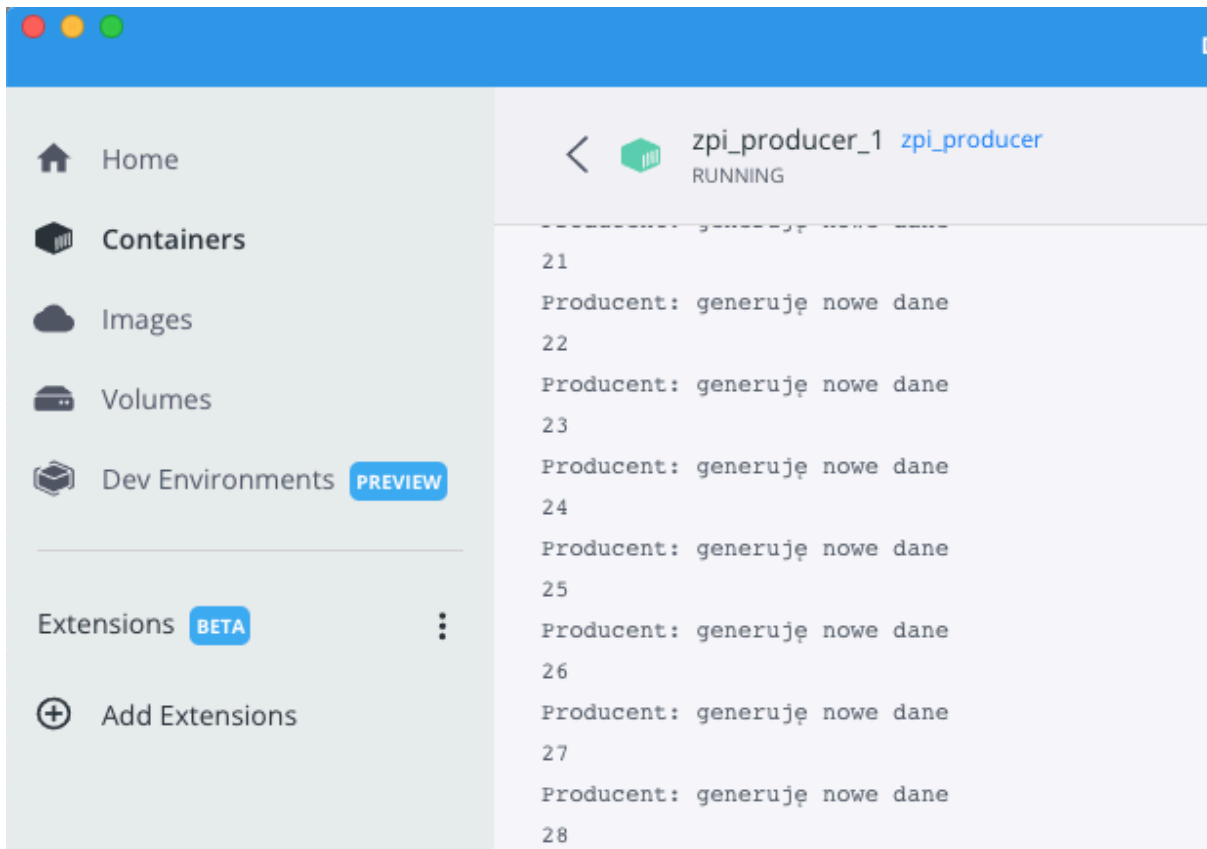
    # Użycie decision_function do uzyskania odległości od hiperpowierzchni decyzyjnej
    clf.decision_function(X_train.values.compute())

    # Przewidujemy wartości
    y_pred = clf.predict(X_test.values.compute())

    # Prawdopodobieństwa dla klas (potrzebne do obliczenia AUC i Log Loss)
    y_prob = clf.predict_proba(X_test.values.compute())
```

Działanie systemu

Generowanie danych strumieniowo



Uruchomienie na żądanie

```
Mac-mini-admin-2:ZPI admin$
Mac-mini-admin-2:ZPI admin$ docker-compose up -d zookeeper kafka
Creating network "zpi_default" with the default driver
Creating zpi_zookeeper_1 ... done
Creating zpi_kafka_1 ... done
Mac-mini-admin-2:ZPI admin$ docker-compose up -d producer
zpi_zookeeper_1 is up-to-date
zpi_kafka_1 is up-to-date
Creating zpi_producer_1 ... done
Mac-mini-admin-2:ZPI admin$
Mac-mini-admin-2:ZPI admin$ docker-compose run --rm consumer
Creating zpi_consumer_run ... done
Pierwsze uruchomienie, brak opóźnienia.
Accuracy: 0.9847775175644028
Precision: 0.9829787234042553
Recall: 0.9892933618843683
F1 Score: 0.9861259338313768
Log Loss: 0.050464220760565406
Mean Absolute Error: 0.01522248243559719
Mean Squared Error: 0.01522248243559719
Root Mean Squared Error: 0.12337942468498217
Ilość odebranych wiadomości: 2113
Mac-mini-admin-2:ZPI admin$
```


Uruchomienie co 20 sekund

```
Mac-mini-admin-2:ZPI admin$  
Mac-mini-admin-2:ZPI admin$  
Mac-mini-admin-2:ZPI admin$ docker-compose up -d consumer  
zpi_zookeeper_1 is up-to-date  
zpi_kafka_1 is up-to-date  
Creating zpi_consumer_1 ... done  
Mac-mini-admin-2:ZPI admin$
```

Home

Containers


Images

Volumes

Dev Environments PREVIEW

Extensions BETA

Add Extensions

<  zpi_consumer_1 zpi_consumer

RUNNING

Pierwsze uruchomienie, brak opóźnienia.

Accuracy: 0.9843096234309623

Precision: 0.9812734082397003

Recall: 0.9905482041587902

F1 Score: 0.9858889934148637

Log Loss: 0.05516087688483126

Mean Absolute Error: 0.015690376569037656

Mean Squared Error: 0.015690376569037656

Root Mean Squared Error: 0.1252612333047925

Ilość odebranych wiadomości: 2231

Opóźnienie uruchomienia o 20 sekund...

Accuracy: 0.9890948745910578

Precision: 0.9919517102615694

Recall: 0.9879759519038076

F1 Score: 0.9899598393574297

Log Loss: 0.046650247516903845

Mean Absolute Error: 0.010905125408942203

Mean Squared Error: 0.010905125408942203

Root Mean Squared Error: 0.10442760846127906

Ilość odebranych wiadomości: 2257

Opóźnienie uruchomienia o 20 sekund...

Home


Containers

Images

Volumes

Dev Environments PREVIEW

Extensions BETA

<  zpi_consumer_1 zpi_consumer

RUNNING

Pierwsze uruchomienie, brak opóźnienia.

Mean Absolute Error: 50.797083707061006

Mean Squared Error: 4613.815148473942

Root Mean Squared Error: 67.92507010282685

R-squared: 0.7222916814160476

Explained Variance: 0.7563920416423596

Mean Absolute Percentage Error: 41.64654707852031

Ilość odebranych wiadomości: 25

Opóźnienie uruchomienia o 20 sekund...