НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №3
з дисципліни «Системи глибинного навчання»
на тему: “Нейромережеве розпізнавання кібератак”
Варіант 4 – Розпізнавання мережевої кібератаки типу teardrop на базі PNN

Виконав:
студент групи КМ-01
Шолоп Л. О.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості

Ймовірнісна нейронна мережа (PNN) - це тип нейронної мережі, який використовується в задачах класифікації. Вона має шарувату структуру, яка складається з чотирьох шарів: вхідного, шаблонів, додавання та вихідного.

В даній мережі щільність ймовірності приналежності класам оцінюється за допомогою ядерної апроксимації. Це означає, що PNN використовує метод найближчого сусіда для визначення класу вхідного вектора, але замість використання одного найближчого сусіда вона використовує всіх сусідів, враховуючи їх відстань до вхідного вектора.

Ці мережі використовуються в різних областях, таких як розпізнавання образів, прогнозування часових рядів, медична діагностика та багато інших. Вони відомі своєю здатністю швидко навчатися, а також тим, що вони не вимагають ітераційного процесу навчання, як багато інших типів нейронних мереж. Замість цього вони використовують простий алгоритм, який вимагає лише одного проходу через навчальний набір даних.

Основна частина

На основі датасету NLS-KDD треба навчити та протестувати нейронну мережу PNN.

Назви колонок у датасеті:

'duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_creations','num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','attack','level'

Колонка - 'attack' відповідає за вид атаки.

Якщо normal то це не атака

Імпортуємо всі необхідні бібліотеки

Створюємо клас мережі PNN, який приймає такі параметри:

```
"""
Ініціалізація параметрів моделі PNN.

Parameters:
- input_size (int): Розмір вхідного вектора.
- output_size (int): Кількість вихідних класів.
"""
```

```
"""
Здійснює прогноз за допомогою моделі PNN.

Parameters:
- X_train (numpy.ndarray): Масив навчальних даних.
- X_test (numpy.ndarray): Масив тестових даних.
- y (numpy.ndarray): Вектор міток класів для навчальних даних.
- sigma (float): Параметр розподілу Гаусса для визначення ширини ядра.

Returns:
- numpy.ndarray: Вектор прогнозованих класів.
"""
```

Завантажуємо датасет із онлайн ресурсі Kaggle.

Оскільки датасети вже розбиті на тестовий та навчальний виведемо розмір наборів:

```
Розмір навчального набору: 54569
Розмір тестового набору: 23388
```

Подивимось, скільки шуканих атак у кожному наборі:

```
Записів із 'teardrop' у навчальному наборі: 632
Записів із 'normal' у навчальному наборі: 53937

Записів із 'teardrop' у тестовому наборі: 272
Записів із 'normal' у тестовому наборі: 23116
```

Подивимось тип кожної з колонок:

```
duration                        int64    count                          int64
protocol_type                  object    srv_count                      int64
service                        object    serror_rate                  float64
flag                           object    srv_serror_rate              float64
src_bytes                       int64    rerror_rate                  float64
dst_bytes                       int64    srv_rerror_rate              float64
land                            int64    same_srv_rate                float64
wrong_fragment                  int64    diff_srv_rate                float64
urgent                          int64    srv_diff_host_rate           float64
hot                             int64    dst_host_count                 int64
num_failed_logins               int64    dst_host_srv_count             int64
logged_in                       int64    dst_host_same_srv_rate       float64
num_compromised                 int64    dst_host_diff_srv_rate       float64
root_shell                      int64    dst_host_same_src_port_rate  float64
su_attempted                    int64    dst_host_srv_diff_host_rate  float64
num_root                        int64    dst_host_serror_rate         float64
num_file_creations              int64    dst_host_srv_serror_rate     float64
num_shells                      int64    dst_host_rerror_rate         float64
num_access_files                int64    dst_host_srv_rerror_rate     float64
num_outbound_cmds               int64    attack                        object
is_host_login                   int64    level                          int64
is_guest_login                  int64    dtype: object
```

Оскільки маємо деякі колонки з типом object їх треба перекодувати для передачі в PNN використаємо one-hot encoding

Спочатку вдало перекодувати не вдалось, тому що наявність певних записів створює ‘зайві’ колонки:

```
Колонок у навчальному наборі: 78
Колонок у тестовому наборі: 77
Кількість спільних колонок: 76

Унікальні колонки в навчальному наборі: {'service_link', 'flag_RSTOS0'}
Унікальні колонки в тестовому наборі: {'service_remote_job'}
```

Оскільки їх небагато, ми їх видалимо:

```
Кількість входжень 'link' у навчальному наборі: 1
Кількість входжень 'RSTOS0' у навчальному наборі: 1
Кількість входжень 'remote_job' у тестовому наборі: 1
```

Повторне перекодування пройшло успішно:

```
Колонок у навчальному наборі: 76
Колонок у тестовому наборі: 76
Кількість спільних колонок: 76
```

Колонка ‘attack’ змінила свою назву після перекодування:

```
attack_teardrop
```

Далі нормалізовуємо дані

Утворюємо навчальні дані та мітки, та тестові дані та мітки

Створюємо об’єкт класу PNN із такими параметрами:

- input_size=76 (кількість колонок після перекодування)

- output_size=2 (розмірність вихідного шару (два класи))

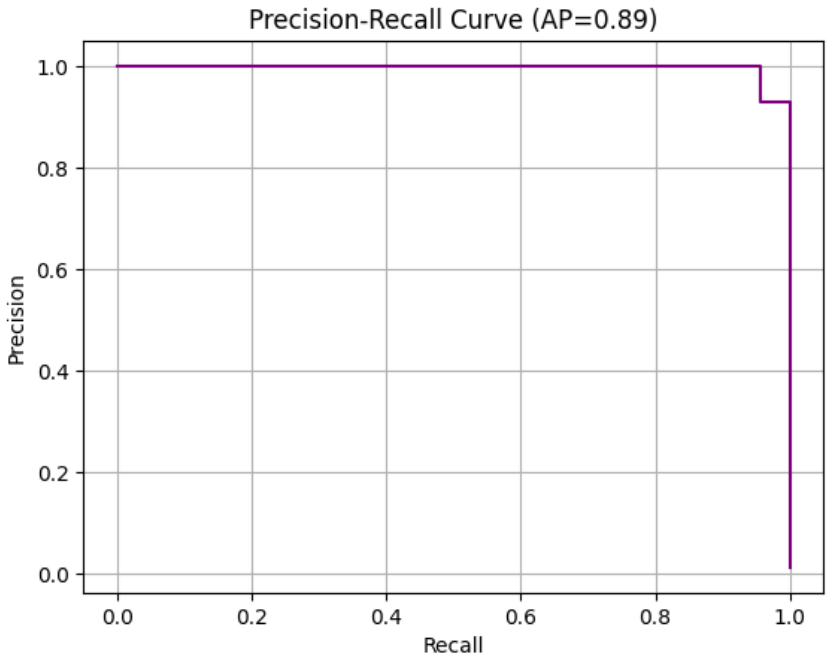
Тестуємо отриману модель:

```
sigma_value = 0.03
predictions = pnn.predict(X_train, X_test.values, y_train, sigma_value)
```

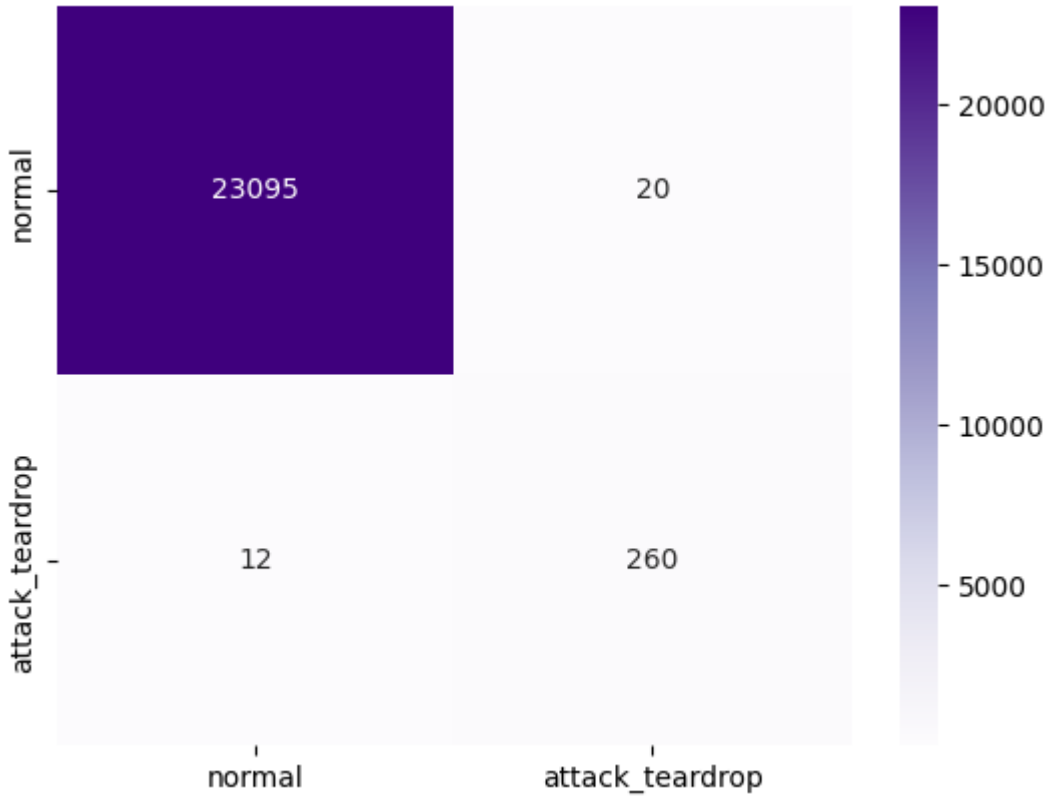
Такі параметри дали наступні результати:

```
Accuracy (Точність) : 99.86317 %
Precision (Точність) : 92.85714 %
Recall (Повнота) : 95.58824 %
F1 Score (F-міра) : 94.2029 %
```

PR крива:



Матриця помилок:



Досить актуальна на даний час проблема з детектом кібератаки. Всі метрики кажуть про те, що результат непоганий, але дивлячись яку задачу ми розв'язуємо.

Якщо наша задача це відловлювання всіх кібератак, а не більшість. То варто змінити гіперпараметри так, щоб рекол став 100%, але при цьому решта метрик не впали до 0 %.

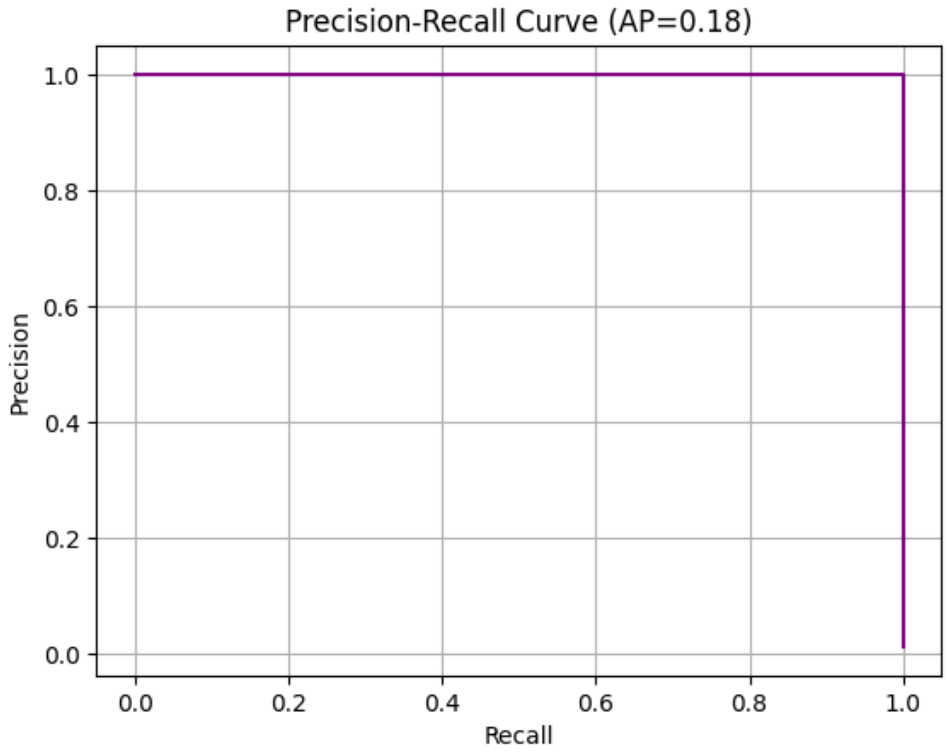
При такому значенні sigma:

```
sigma_value = 1.0
predictions = pnn.predict(X_train, X_test.values, y_train, sigma_value)
```

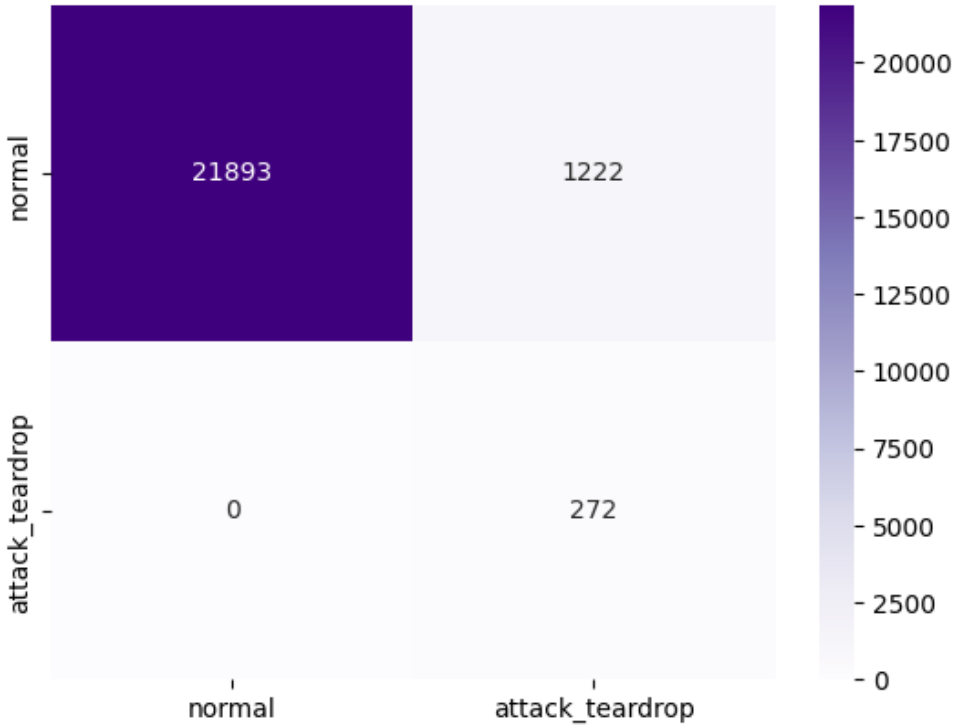
Ми отримуємо наступні результати:

```
Accuracy (Точність) : 94.77487 %
Precision (Точність) : 18.20616 %
Recall (Повнота) : 100.0 %
F1 Score (F-міра) : 30.80408 %
```

PR крива:



Матриця помилок:



Ціль досягнута, рекол = 100%

Тому зміна гіперпараметрів може бути чудовим інструментом для адаптації до конкретної задачі класифікації.

Додаток А – Код програми

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import precision_recall_curve, average_precision_score,
confusion_matrix
```

```
class PNN:
    def __init__(self, input_size, output_size):
        """
        Ініціалізація параметрів моделі PNN.

        Parameters:
        - input_size (int): Розмір вхідного вектора.
        - output_size (int): Кількість вихідних класів.
        """
        self.input_size = input_size
        self.output_size = output_size
        self.mean_vectors = None
        self.sigma = None
        self.weights = None


    def predict(self, X_train, X_test, y, sigma=1.0):
        """
        Здійснює прогноз за допомогою моделі PNN.

        Parameters:
        - X_train (numpy.ndarray): Масив навчальних даних.
        - X_test (numpy.ndarray): Масив тестових даних.
        - y (numpy.ndarray): Вектор міток класів для навчальних даних.
        - sigma (float): Параметр розподілу Гаусса для визначення ширини ядра.

        Returns:
        - numpy.ndarray: Вектор прогнозованих класів.
        """
        self.mean_vectors = []

        # Обчислюємо середні вектори для кожного класу
        for class_label in range(self.output_size):
            class_samples = X_train[y == class_label]
            mean_vector = np.mean(class_samples, axis=0)
            self.mean_vectors.append(mean_vector)
```

```
        self.mean_vectors = np.array(self.mean_vectors)

        # Обчислюємо ваги для кожного класу
        self.weights = np.ones(self.output_size) / self.output_size
        predictions = []

        for sample in X_test:
            probabilities = []

            # Розраховуємо ймовірності для кожного класу
            for class_label in range(self.output_size):
                mean_vector = self.mean_vectors[class_label]
                sample = sample.astype(float)
                activation = np.exp(-0.5 * np.sum((sample - mean_vector) ** 2) /
(sigma ** 2))
                probability = activation * self.weights[class_label]
                probabilities.append(probability)

            # Визначаємо клас з найвищою ймовірністю
            predicted_class = np.argmax(probabilities)
            predictions.append(predicted_class)

        return np.array(predictions)
```

```
train_data = pd.read_csv('KDDTrain+.txt')
test_data = pd.read_csv('KDDTest+.txt')

columns =
(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong
_fragment','urgent','hot',
            'num_failed_logins','logged_in','num_compromised','root_shell','su_atte
mpted','num_root',
            'num_file_creations','num_shells','num_access_files','num_outbound_cmds
','is_host_login','is_guest_login',
            'count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_
rerror_rate','same_srv_rate',
            'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_cou
nt','dst_host_same_srv_rate',
            'dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_di
ff_host_rate','dst_host_serror_rate',
            'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_
rate','attack','level'])

train_data.columns = columns
test_data.columns = columns

# Поєднання двох датафреймів
combined_data = pd.concat([train_data, test_data], axis=0)
```

```
# Розділення за типом атаки
attack_data = combined_data[combined_data['attack'] == 'teardrop']
normal_data = combined_data[combined_data['attack'] == 'normal']

# Розбиття на навчальний та тестовий набір
train_attack, test_attack = train_test_split(attack_data, test_size=0.3,
random_state=42)
train_normal, test_normal = train_test_split(normal_data, test_size=0.3,
random_state=42)

# Об'єднання навчальних та тестових наборів
train_data = pd.concat([train_attack, train_normal], axis=0)
test_data = pd.concat([test_attack, test_normal], axis=0)

# Виведення розмірів навчального та тестового наборів
print(f'Розмір навчального набору: {train_data.shape[0]}')
print(f'Розмір тестового набору: {test_data.shape[0]}')
```

```
count_train_teardrop = (train_data['attack'] == 'teardrop').sum()
count_train_normal = (train_data['attack'] == 'normal').sum()
count_test_teardrop = (test_data['attack'] == 'teardrop').sum()
count_test_normal = (test_data['attack'] == 'normal').sum()

print(f"Записів із 'teardrop' у навчальному наборі: {count_train_teardrop}")
print(f"Записів із 'normal' у навчальному наборі: {count_train_normal}\n")
print(f"Записів із 'teardrop' у тестовому наборі: {count_test_teardrop}")
print(f"Записів із 'normal' у тестовому наборі: {count_test_normal}")
```

```
train_data = train_data[(train_data['attack'] == 'teardrop') |
(train_data['attack'] == 'normal')]
test_data  = test_data[ (test_data['attack']  == 'teardrop') |
(test_data['attack']  == 'normal')]
```

```
train_data.dtypes  #  датафрейм test_data містить такі самі колонки та типи
```

```
# Використання one-hot encoding для категорійних змінних
filtered_train_data = pd.get_dummies(train_data, drop_first=True)
filtered_test_data  = pd.get_dummies(test_data,  drop_first=True)
```

```
print(f'Колонок у навчальному наборі: {filtered_train_data.shape[1]}')
print(f'Колонок у тестовому наборі: {filtered_test_data.shape[1]}')

# Отримання назв колонок для обох датафреймів
columns_train = set(filtered_train_data.columns)
columns_test = set(filtered_test_data.columns)
```

```
common_columns = columns_train.intersection(columns_test)  # Знаходження спільних назв колонок
print('Кількість спільних колонок:', len(common_columns))

# Знаходження унікальних колонок у кожному датафреймі
unique_columns_train = columns_train.difference(columns_test)
unique_columns_test = columns_test.difference(columns_train)

# Виведення назв унікальних колонок у кожному датафреймі
print(f'\nУнікальні колонки в навчальному наборі: {unique_columns_train}')
print(f'Унікальні колонки в тестовому наборі: {unique_columns_test}')
```

```
service_link = (train_data['service'] == 'link').sum()
flag_RSTOS0 = (train_data['flag'] == 'RSTOS0').sum()
service_remote_job = (test_data['service'] == 'remote_job').sum()

print(f"Кількість входжень 'link' у навчальному наборі: {service_link}")
print(f"Кількість входжень 'RSTOS0' у навчальному наборі: {flag_RSTOS0}")
print(f"Кількість входжень 'remote_job' у тестовому наборі: {service_remote_job}")
```

```
train_data = train_data[(train_data['service'] != 'link') & (train_data['flag'] != 'RSTOS0')]
test_data = test_data[(test_data['service'] != 'remote_job')]
```

```
filtered_train_data = pd.get_dummies(train_data, drop_first=True)
filtered_test_data  = pd.get_dummies(test_data,  drop_first=True)

print(f'Колонок у навчальному наборі: {filtered_train_data.shape[1]}')
print(f'Колонок у тестовому наборі: {filtered_test_data.shape[1]}')

# Отримання назв колонок для обох датафреймів
columns_train = set(filtered_train_data.columns)
columns_test = set(filtered_test_data.columns)

common_columns = columns_train.intersection(columns_test)  # Знаходження спільних назв колонок
print('Кількість спільних колонок:', len(common_columns))
```

```
print(filtered_test_data.columns[-1])
```

```
scaler = MinMaxScaler()

new_columns = filtered_train_data.columns

filtered_train_data[new_columns] = scaler.fit_transform(filtered_train_data[new_columns])
```

```
filtered_test_data[new_columns] = scaler.transform(filtered_test_data[new_columns])
```

```
X_train = filtered_train_data.drop(columns=['attack_teardrop'])
y_train = filtered_train_data['attack_teardrop']

X_test = filtered_test_data.drop(columns=['attack_teardrop'])
y_test = filtered_test_data['attack_teardrop']
```

```
input_size = filtered_train_data.shape[1]
output_size = 2  # розмірність вихідного шару (два класи)

pnn = PNN(input_size, output_size)
```

```
sigma_value = 0.03
# sigma_value = 1.0
predictions = pnn.predict(X_train, X_test.values, y_train, sigma_value)
```

```
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print(f"Accuracy (Точність) : {round(accuracy * 100, 5)} %")
print(f"Precision (Точність) : {round(precision * 100, 5)} %")
print(f"Recall (Повнота) : {round(recall * 100, 5)} %")
print(f"F1 Score (F-міра) : {round(f1 * 100, 5)} %")
```

```
precision, recall, _ = precision_recall_curve(y_test, predictions)
average_precision = average_precision_score(y_test, predictions)

plt.figure()
plt.step(recall, precision, color='purple')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (AP={:.2f})'.format(average_precision))
plt.grid()
plt.show()
```

```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples",
            xticklabels=['normal', 'attack_teardrop'],
            yticklabels=['normal', 'attack_teardrop'])
plt.show()
```