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: | Викладач: |
| студент групи КМ-01 | Професор кафедри ПМА |
| Шолоп Л. О. | Терейковський І. А. |

Київ — 2023

Зміст

[Теоретичні відомості 3](#_Toc153483023)

[Основна частина 4](#_Toc153483024)

[Додаток А – Код програми 9](#_Toc153483025)

## 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

protocol\_type object

service object

flag object

src\_bytes int64

dst\_bytes int64

land int64

wrong\_fragment int64

urgent int64

hot int64

num\_failed\_logins int64

logged\_in int64

num\_compromised int64

root\_shell int64

su\_attempted int64

num\_root int64

num\_file\_creations int64

num\_shells int64

num\_access\_files int64

num\_outbound\_cmds int64

is\_host\_login int64

is\_guest\_login int64

count int64

srv\_count int64

serror\_rate float64

srv\_serror\_rate float64

rerror\_rate float64

srv\_rerror\_rate float64

same\_srv\_rate float64

diff\_srv\_rate float64

srv\_diff\_host\_rate float64

dst\_host\_count int64

dst\_host\_srv\_count int64

dst\_host\_same\_srv\_rate float64

dst\_host\_diff\_srv\_rate float64

dst\_host\_same\_src\_port\_rate float64

dst\_host\_srv\_diff\_host\_rate float64

dst\_host\_serror\_rate float64

dst\_host\_srv\_serror\_rate float64

dst\_host\_rerror\_rate float64

dst\_host\_srv\_rerror\_rate float64

attack object

level 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:

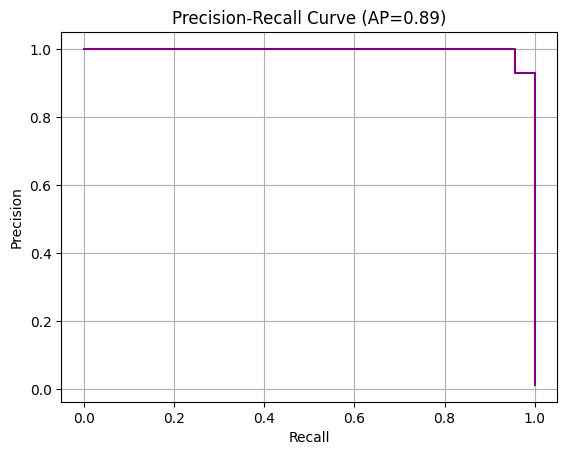
Accuracy (Точність) : 99.86317 %

Precision (Точність) : 92.85714 %

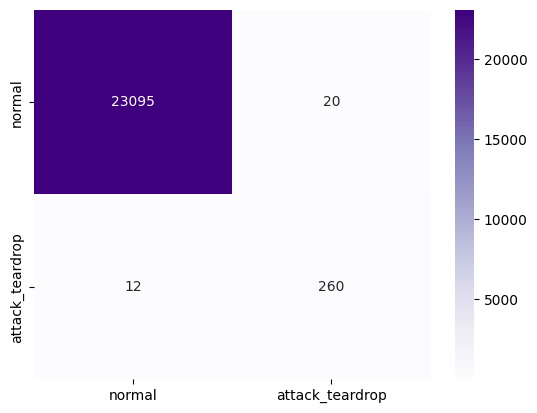
Recall (Повнота) : 95.58824 %

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:

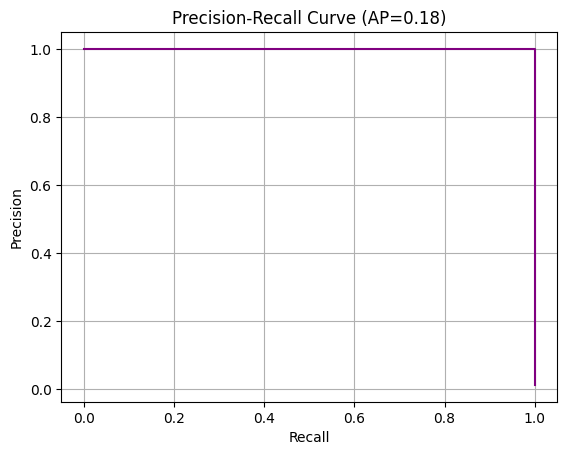
Accuracy (Точність) : 94.77487 %

Precision (Точність) : 18.20616 %

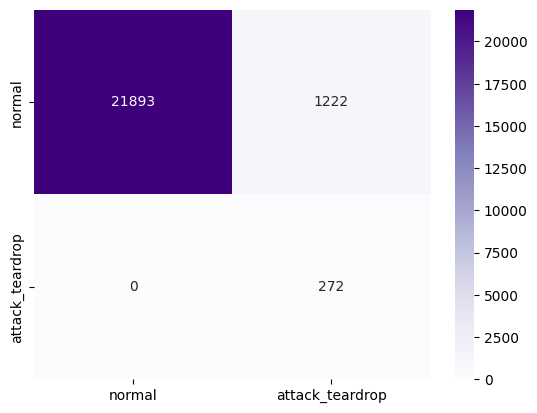
Recall (Повнота) : 100.0 %

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\_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'])

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()