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 16 – Розпізнавання мережевої кібератаки типу rootkit на базі PNN

Виконав:
студент групи КМ-01
Романецький М.С.

Викладач:
Професор кафедри ПМА
Терейковський І. А.

Київ — 2023

Зміст

Теоретичні відомості 3
Основна частина 6
Перелік посилань 12
Додаток А – Код програми 13

Теоретичні відомості

Rootkit

Кібератака типу rootkit є однією з найбільш складних і небезпечних форм кіберзлочинності. Rootkit - це програмне забезпечення, призначене для таємного доступу або контролю над комп'ютерною системою без відома чи згоди її власника. Цей тип атаки використовується для отримання привілеїв "root" або "адміністратора" на зараженій системі, надаючи зловмиснику повний контроль над нею.

Rootkit володіє рядом характеристик, які роблять його вкрай небезпечним для жертви:

1. Таємність:
   Rootkit намагається приховати свою присутність в системі, маскуючись перед антивірусами та іншими засобами виявлення. Це може включати приховування файлів, процесів та мережевого трафіку, пов'язаного з атакою.
2. Стійкість:
   Багато rootkit-ів намагаються залишитися невиявленими та уникнути видалення, вбудовуючи свої компоненти у ядро операційної системи або в інші системні файли.
3. Привілеї адміністратора:
   Rootkit, як правило, спрямований на отримання найвищого рівня привілеїв в системі, що дозволяє зловмисникові виконувати будь-які дії та маніпуляції з комп'ютером.
4. Персистентність:
   Rootkit може залишатися в системі тривалий час, регенеруючись після перезавантаження системи або інших утручань.
5. Можливості вторгнення:
   Rootkit може використовувати різноманітні вектори вторгнення, включаючи зараження через заражені файли, використання вразливостей у мережевих протоколах або соціально-інженерні методи.

Захист від атаки rootkit включає в себе регулярну оновлення антивірусного програмного забезпечення, моніторинг мережевої активності, а також використання фаєрволів та інших засобів безпеки. Розуміння принципів функціонування rootkit-ів дозволяє краще захищати системи від цього типу кіберзагроз.

Ймовірнісна нейронна мережа PNN

Ймовірнісна нейронна мережа PNN - це вид штучних нейронних мереж, який використовує Баєсову статистику для виконання певних завдань, таких як класифікація або регресія. Ймовірнісна нейронна мережа була розроблена Дональдом Спехтом у 1990 році. Основна ідея PNN полягає в тому, що виходи мережі можна інтерпретувати як оцінки ймовірності належності вхідного об’єкта певному класу або значенню функції. Для цього PNN використовує метод ядерних оцінок густини ймовірності, який базується на використанні Гаусових функцій.

Структура PNN складається з чотирьох шарів: вхідного, прихованого, сумувального та вихідного. Вхідний шар містить нейрони, які передають вхідний вектор до прихованого шару. Прихований шар містить нейрони, які обчислюють відстань від вхідного вектора до кожного з навчальних прикладів, які належать до певного класу. Сумувальний шар містить нейрони, які підсумовують виходи прихованого шару для кожного класу. Вихідний шар містить нейрони, які нормалізують виходи сумувального шару та видають оцінки ймовірності для кожного класу.

Для навчання PNN не потрібно використовувати жодних алгоритмів оптимізації, таких як градієнтний спуск або зворотне поширення помилки. Навчання PNN полягає в тому, що для кожного навчального прикладу створюється нейрон у прихованому шарі, який має центр у цьому прикладі та параметр ширини, який визначається емпірично. Таким чином, PNN має просту та швидку процедуру навчання, але водночас вимагає багато пам’яті та обчислювальних ресурсів для зберігання та обробки всіх навчальних прикладів.

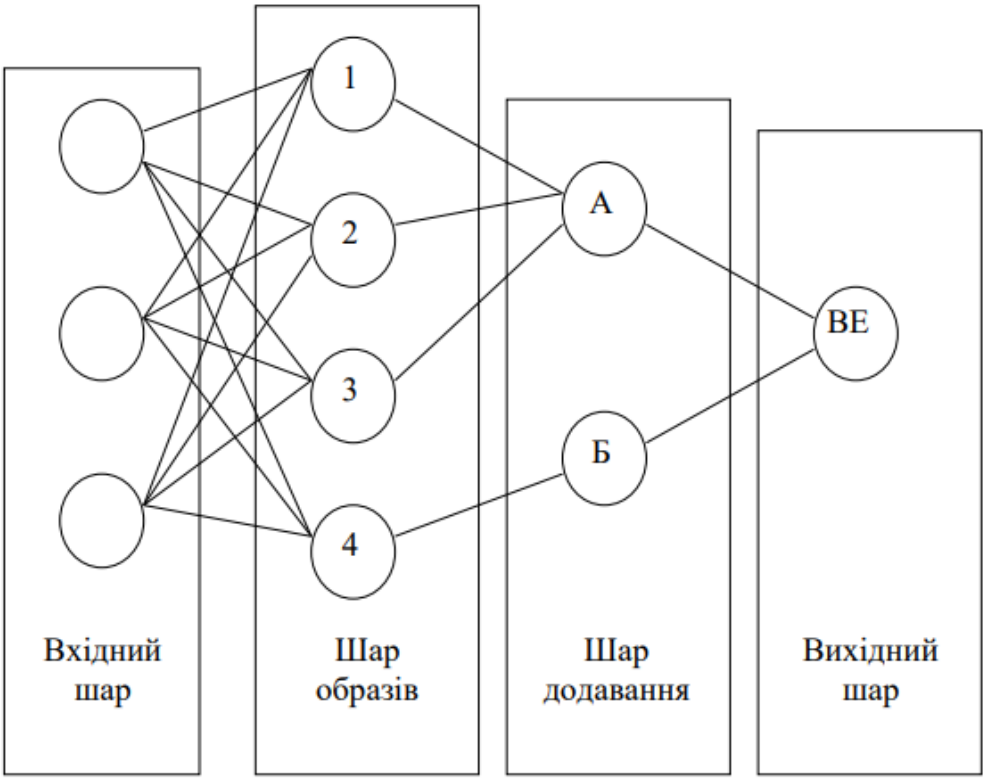

Рис. 1 – Архітектурна схема мережі PNN

https://ela.kpi.ua/bitstream/123456789/50135/1/ANN.pdf ст. 64

Основна частина

За індивідуальним варіантом завдання треба розробити нейронну мережу на базі PNN, яка буде розпізнавати мережеві кібератаки типу ‘rootkit’.

Для цього нам знадобиться датасет NSL-KDD [4] який містить дані про мережеві кібератаки та типи цих атак (normal, якщо атака не підтвердилась). Датасет було взято з інтернет ресурсу Kaggle.

Датасет містить такі назви колонок:

```
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'])
```

Мітками типів атак є колонка 'attack'.

Оскільки є готові файли з даними для тренування та тестування, скористаємось ними. Перевіримо скільки всього записів які містять ‘rootkit’ або ‘normal’ у них.

```
Кількість записів у датасеті 'train_data': 125972
Кількість записів у датасеті 'test_data' : 22543

Кількість входжень 'rootkit' у 'train_data': 10
Кількість входжень 'rootkit' у 'test_data':  13
Кількість входжень 'normal'  у 'train_data': 67342
Кількість входжень 'normal'  у 'test_data':  9711
```

Як бачимо, даних про кібератаки дуже багато, але в той же час даних саме про атаки типу ‘rootkit’ дуже мало. Тому наступним кроком приберемо з датасету всі дані які мають тип атаки не ‘rootkit’ або не ‘normal’. Таким чином тренувальний датасет містить 67352 записи, а тестовий 9724 записи.

Також варто зауважити, що ми будемо передавати ці дані в нейронну мережу, а отже треба перекодувати категорійні дані в чисельні.

Після перекодування варто переконатись, що назви колонок співпадають та кількість колонок однакова для тренувального та тестового набору даних.

```
Колонок у 'filtered_train_data': 76
Колонок у 'filtered_test_data':  70
Кількість спільних колонок: 67

Унікальні колонки в filtered_train_data:
{'flag_REJ', 'service_urh_i', 'service_ssh', 'service_imap4',
'flag_SH', 'service_shell', 'flag_S0', 'service_domain',
'service_red_i'}

Унікальні колонки в filtered_test_data:
{'flag_RSTOS0', 'service_remote_job', 'service_link'}

Отже, назви та кількість не співпадають
```

Тепер чітко видно, що тренувальний датасет містить 9 колонок, яких не містить тестовий датасет, та 3 колонки містить тестовий датасет, яких немає в тренувальному. Це сталось через те що функція pd.get_dummies() перекодовує стовпчики шляхом створення нових і ставить 1 якщо запис містив потрібну категорію, і 0 всім іншим категоріям.

Подивимось, наскільки багато цих даних, які створюють окремі стовпчики:

```
Кількість входжень 'red_i'  у 'train_data': 8
Кількість входжень 'ssh'    у 'train_data': 5
Кількість входжень 'domain' у 'train_data': 38
Кількість входжень 'urh_i'  у 'train_data': 10
Кількість входжень 'shell'  у 'train_data': 4
Кількість входжень 'imap4'  у 'train_data': 3
Кількість входжень 'REJ'    у 'train_data': 2693
Кількість входжень 'SH'     у 'train_data': 2
Кількість входжень 'S0'     у 'train_data': 354

Кількість входжень 'link'   у 'test_data': 1
Кількість входжень 'job'    у 'test_data': 1
Кількість входжень 'RSTOS0' у 'test_data': 1
```

Оскільки тренувальний датасет містить 67352 записи, а тестовий 9724 записи. Можемо видалити таку кількість записів, адже це не має сильно вплинути на результат досліджень.

Видаляємо дані та знову перевіряємо чи співпадають назви колонки та їх кількість:

```
Колонок у 'filtered_train_data': 67
Колонок у 'filtered_test_data':  67
Кількість спільних колонок: 67

Отже назви колонок у датафреймах повністю однакові
```

Тепер коли в нас усі дані переведені в числовий формат, їх треба нормалізувати. Для цього використаємо MinMaxScaler() із бібліотеки sklearn.preprocessing

Далі створюємо клас для ймовірнісної нейронної мережі PNN аналогічно до того як це було зроблено в лабораторній роботі № 2. Тобто є функція тренування та розпізнавання.

Оскільки маємо 2 датафрейми **filtered_train_data** та **filtered_test_data** потрібно розділити їх на X_train, X_test, y_train, y_test

Ініціалізуємо модель PNN. Із вхідним шаром у якому кількість нейронів буде дорівнювати кількості стовпців у датасеті, та двома вихідними шарами, які будуть відповідати за те чи була це атака типу ‘rootkit’ чи ні.

Тренуємо PNN на навчальних даних.

```
pnn.train(X_train, y_train, sigma=0.077)
```

Параметр сігма підібраний емпіричним шляхом, критерієм зупинки саме на цьому значенні стало те, що вручну підібрати краще значення не вдалось.

Викликаємо функцію розпізнавання для тестового набору даних:

```
predictions = pnn.predict(X_test.values)
```

Щоб зрозуміти наскільки добре модель справляється з поставленою задачею виведемо метрики accuracy, precision, recall, f1-score.

```
Accuracy (Точність) : 93.22086 %
Precision (Точність): 1.93452 %
Recall (Повнота)    : 100.0 %
F1 Score (F-міра)   : 3.79562 %
```

Проаналізувавши ці результати можемо дійти таких висновків:

1. Модель вдало розпізнала всі атаки ‘rootkit’, які дійсно були такими, про це каже recall = 100%
2. Загальна точність accuracy висока, тобто модель добре класифікувала як ‘атаку’ так і ‘не атаку’
3. Низький precision каже про те, що модель погано впоралась із тим, що помилково визначала за атаку, коли насправді це була не атака. Але на мою думку, це дає більше роботи людям, які перевіряють і захищають певну систему, проте краще зайвий раз перевірити, ніж пропустити шкідливе ПЗ
4. F-міра досить мала, це не добре, але це через те що precision дав погані показники, тому збудуємо також ROC-криву та подивимось, як ця метрика оцінить результати

Також візуалізуємо отримані результати у вигляді гістограми та матриці помилок.

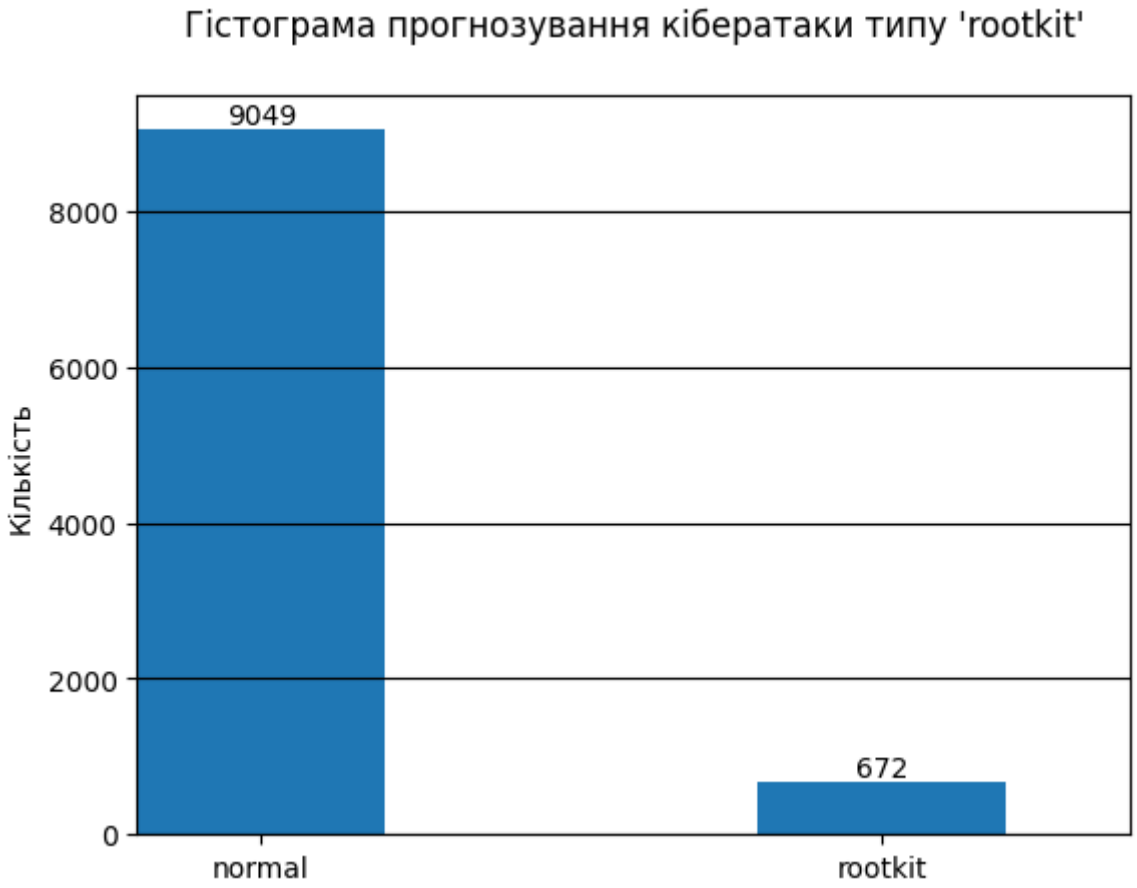

Рис. 2 – Гістограма прогнозування кібератаки типу ‘rootkit’

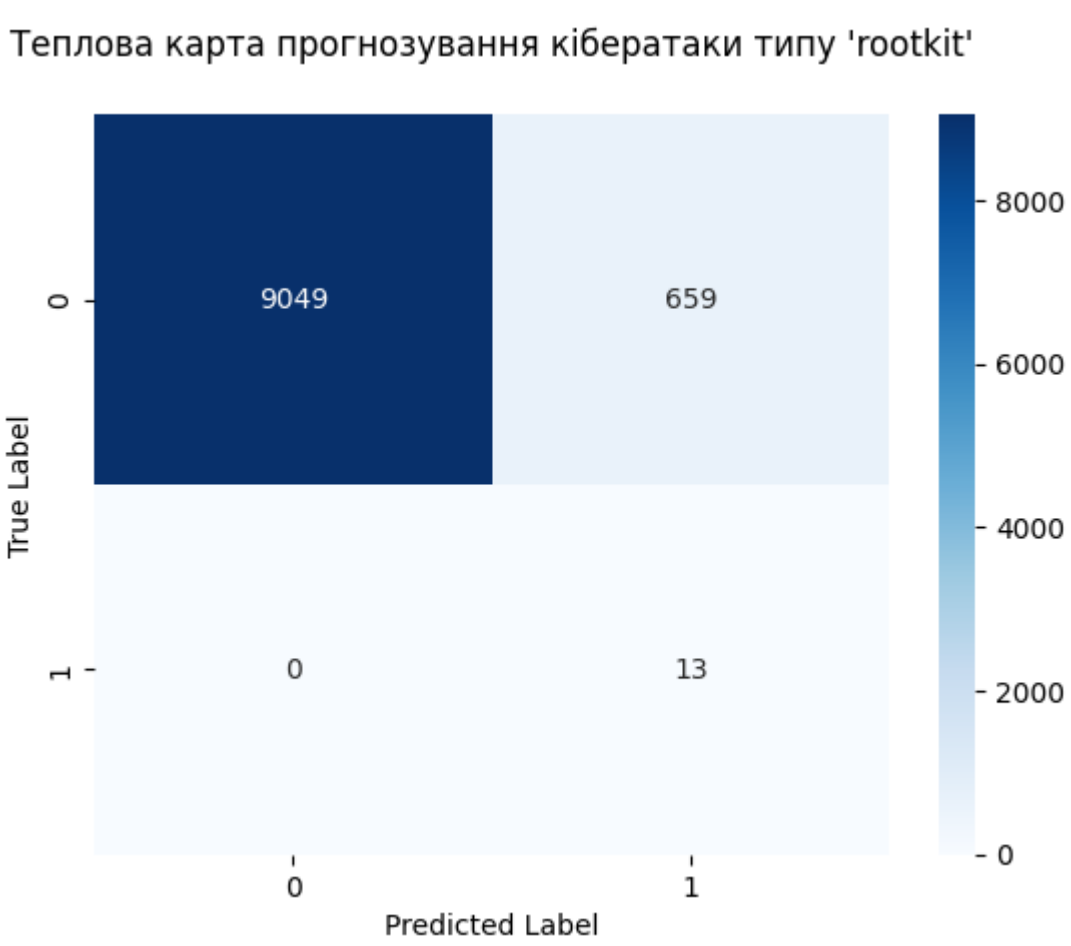

Рис. 3 – Теплова карта прогнозування кібератаки типу ‘rootkit’

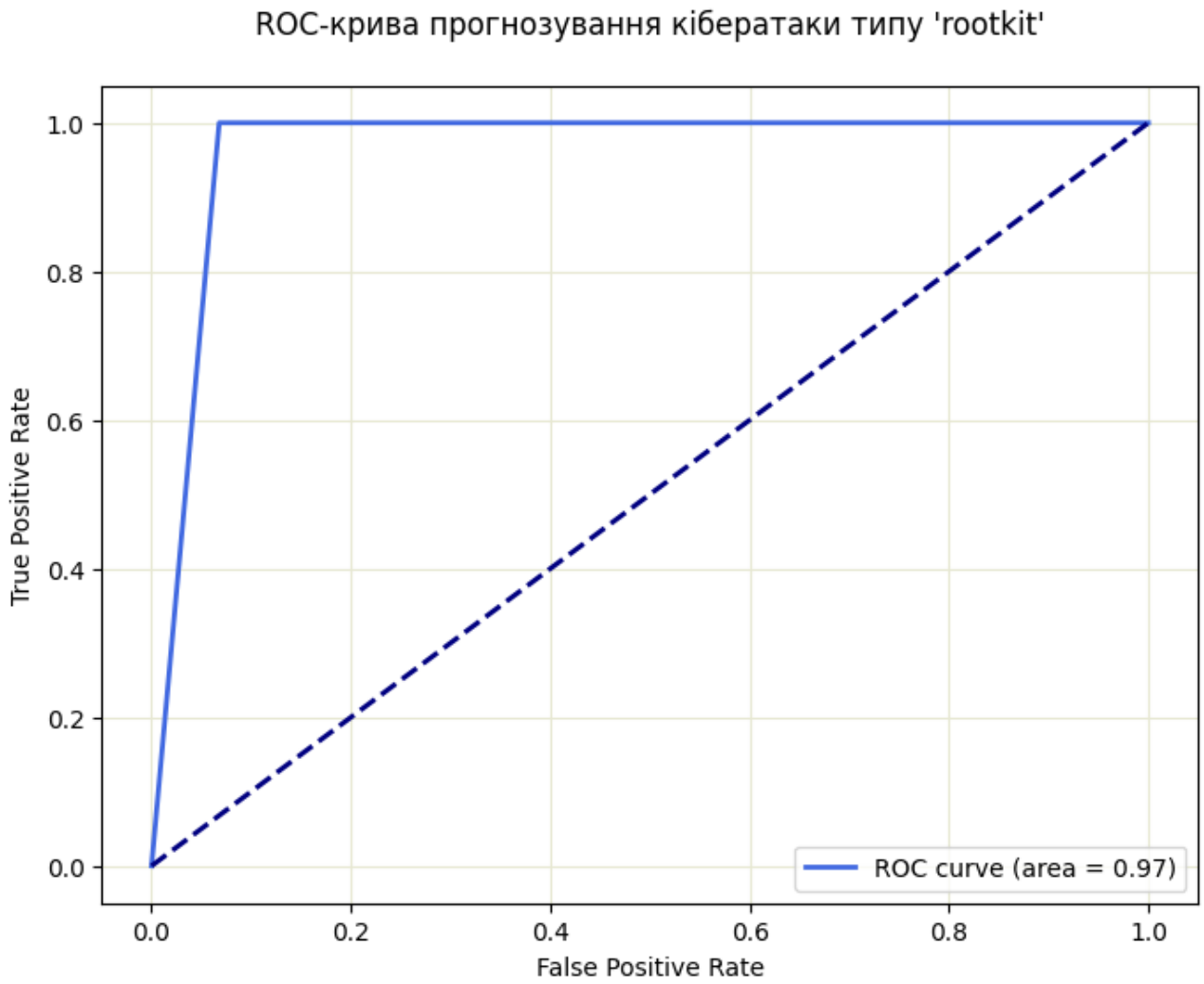

Рис. 4 – ROC-крива прогнозування кібератаки типу ‘rootkit’

Як бачимо, площа під ROC-кривою має дуже хороший показник (0.97), тому будемо вважати, що дана нейронна мережа справилась із поставленою задачею.

На останок, характеристики системи та часові витрати на роботу програми:

```
Час роботи програми: 5.2 секунд

Процесор:   AMD Ryzen 5 3550H with Radeon Vega
ОЗП:        8.00 ГБ
ОС:         Windows 10 Pro
```

Перелік посилань

1. Навчальний посібник «Основні концепції нейроних мереж» Роберт Каллан – стор. 158-164.
2. https://ela.kpi.ua/bitstream/123456789/50135/1/ANN.pdf
3. Руденко О.Г. Штучні нейронні мережі. Навч. посіб. / О. Г. Руденко, Є. В. Бодянський.
4. NSL-KDD - https://www.kaggle.com/datasets/hassan06/nslkdd/data?select=KDDTrain%2B_20Percent.txt

Додаток А – Код програми

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_curve, auc, confusion_matrix
```

```
start_timer = time.time()
train_data = pd.read_csv('dataset\KDDTrain+.txt')
test_data = pd.read_csv('dataset\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
```

```
def check_rootkit_normal():
    count_train_rootkit = (train_data['attack'] == 'rootkit').sum()
    count_test_rootkit = (test_data['attack'] == 'rootkit').sum()
    count_train_normal = (train_data['attack'] == 'normal').sum()
    count_test_normal = (test_data['attack'] == 'normal').sum()

    print(f'Кількість записів у датасеті \'train_data\': {train_data.shape[0]}')
    print(f'Кількість записів у датасеті \'test_data\' : {test_data.shape[0]}\n')

    print(f"Кількість входжень 'rootkit' у 'train_data': {count_train_rootkit}")
```

```
    print(f"Кількість входжень 'rootkit' у 'test_data':  {count_test_rootkit}")
    print(f"Кількість входжень 'normal'  у 'train_data': {count_train_normal}")
    print(f"Кількість входжень 'normal'  у 'test_data':  {count_test_normal}")

check_rootkit_normal()
```

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

# train_data.dtypes
# test_data.dtypes
```

```
# 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 \'filtered_train_data\': {filtered_train_data.shape[1]}')
print(f'Колонок у \'filtered_test_data\':  {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)

# Знаходження спільних назв колонок
common_columns = columns_train.intersection(columns_test)

# Виведення назв та кількості спільних 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 filtered_train_data: \n{unique_columns_train}')
print(f'\nУнікальні колонки в filtered_test_data: \n{unique_columns_test}')
print(f'\nОтже, назви та кількість не співпадають')
```

```
a = (train_data['service'] == 'red_i').sum()
b = (train_data['service'] == 'ssh').sum()
c = (train_data['service'] == 'domain').sum()
d = (train_data['service'] == 'urh_i').sum()
e = (train_data['service'] == 'shell').sum()
f = (train_data['service'] == 'imap4').sum()
```

```
fl = (train_data['flag'] == 'REJ').sum()
fll = (train_data['flag'] == 'SH').sum()
flll = (train_data['flag'] == 'S0').sum()

ser1 = (test_data['service'] == 'link').sum()
ser2 = (test_data['service'] == 'remote_job').sum()
ser3 = (test_data['flag'] == 'RSTOS0').sum()

print(f"Кількість входжень 'red_i'  у 'train_data': {a}")
print(f"Кількість входжень 'ssh'    у 'train_data': {b}")
print(f"Кількість входжень 'domain' у 'train_data': {c}")
print(f"Кількість входжень 'urh_i'  у 'train_data': {d}")
print(f"Кількість входжень 'shell'  у 'train_data': {e}")
print(f"Кількість входжень 'imap4'  у 'train_data': {f}")

print(f"Кількість входжень 'REJ'    у 'train_data': {fl}")
print(f"Кількість входжень 'SH'     у 'train_data': {fll}")
print(f"Кількість входжень 'S0'     у 'train_data': {flll}")

print(f"\nКількість входжень 'link'   у 'test_data': {ser1}")
print(f"Кількість входжень 'job'    у 'test_data': {ser2}")
print(f"Кількість входжень 'RSTOS0' у 'test_data': {ser3}")
```

```
# Видалення записів з умовами у train_data
train_data = train_data[(train_data['service'] != 'red_i') &
                        (train_data['service'] != 'ssh') &
                        (train_data['service'] != 'domain') &
                        (train_data['service'] != 'urh_i') &
                        (train_data['service'] != 'shell') &
                        (train_data['service'] != 'imap4') &
                        (train_data['flag'] != 'REJ') &
                        (train_data['flag'] != 'SH') &
                        (train_data['flag'] != 'S0')]

# Видалення записів з умовами у test_data
test_data = test_data[(test_data['service'] != 'link') &
                      (test_data['service'] != 'remote_job') &
                      (test_data['flag'] != 'RSTOS0')]
```

```
# 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 \'filtered_train_data\': {filtered_train_data.shape[1]}')
print(f'Колонок у \'filtered_test_data\':  {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)

# Знаходження спільних назв колонок
common_columns = columns_train.intersection(columns_test)

# Виведення назв та кількості спільних колонок
print('Кількість спільних колонок:', len(common_columns))
print('\nОтже назви колонок у датафреймах повністю однакові')
```

```
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])
```

```
class PNN:
    def __init__(self, input_size, output_size):
        self.input_size = input_size
        self.output_size = output_size
        self.mean_vectors = None
        self.sigma = None
        self.weights = None


    def train(self, X, y, sigma=1.0):
        self.mean_vectors = []
        self.sigma = sigma

        # Обчислюємо середні вектори для кожного класу
        for class_label in range(self.output_size):
            class_samples = X[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


    def predict(self, X):
        predictions = []

        for sample in X:
            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) /
(self.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)
```

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

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

```
input_size = train_data.shape[1]
output_size = 2  # розмірність вихідного шару (два класи: rootkit та normal)

pnn = PNN(input_size, output_size)
```

```
pnn.train(X_train, y_train, sigma=0.077)
```

```
predictions = pnn.predict(X_test.values)
```

```
y_true = y_test
accuracy = accuracy_score(y_true, predictions)
precision = precision_score(y_true, predictions)
recall = recall_score(y_true, predictions)
f1 = f1_score(y_true, 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)} %")
```

```
cord = [0.1, 0.6]
for i, value in zip(cord, plt.hist(predictions, bins=2, width=0.2)[0]):
    plt.text(i, value + 0.1, str(int(value)), ha='center', va='bottom')
plt.title('Гістограма прогнозування кібератаки типу \'rootkit\'\n')
```

```
plt.ylabel('Кількість')
plt.xticks([0.1, 0.6], ['normal', 'rootkit'])
plt.xlim(0, 0.8)
plt.grid(axis='y',c='black')
plt.show()
```

```
cm = confusion_matrix(y_true, predictions)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title('Теплова карта прогнозування кібератаки типу \'rootkit\'\n')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
fpr, tpr, _ = roc_curve(y_true, predictions)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='royalblue', lw=2, label='ROC curve (area =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-крива прогнозування кібератаки типу \'rootkit\'\n')
plt.legend(loc='lower right')
plt.grid(c='#E7E8D2')
plt.show()
```

```
print(f'Час роботи програми: {round(time.time() - start_timer, 1)} секунд\n')
print(f'Процесор:   AMD Ryzen 5 3550H with Radeon Vega')
print(f'ОЗП:        8.00 ГБ')
print(f'ОС:         Windows 10 Pro')
```