МIНIСТЕРСТВО ОСВIТИ I НАУКИ УКРАЇНИ

НАЦIОНАЛЬНИЙ ТЕХНIЧНИЙ УНIВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛIТЕХНIЧНИЙ IНСТИТУТ»

Кафедра прикладної математики

Звіт

із лабораторної роботи №3

з дисципліни «Методи штучного інтелекту»

на тему:

«Розпізнавання кібератак за допомогою PNN»

|  |  |
| --- | --- |
| Виконав: | Керівник: |
| студент групи КМ-63 | *Терейковська Л. О.* |
| *Артеменко Я.К.* |  |

Київ — 2019

# **ЗМІСТ**

[**Завдання** 3](#_Toc26901682)

[**Опис** 3](#_Toc26901683)

[**Додаток** 4](#_Toc26901684)

[**Додаток А (код програми)** 4](#_Toc26901685)

[**Додаток Б (скріншоти результату)** 7](#_Toc26901686)

# **Завдання**

Варіант №1

Розпізнавання мережевої кібератаки типу neptune на базі PNN.

# **Опис**

Для даної роботи було використано базу даних KDD-99, яка містить дані про кібератаки та умови, при яких вони відбуваються.

Для виконання даної лабораторної роботи, дані з бази даних необхідно попередньо нормалізувати. Для цього в кожному стовпці було знайдено максимальний та мінімальний елементи, за допомогою яких проводилась нормалізація даних. Також текстові поля були замінені на відповідні числові еквіваленти. Також нормалізація проводиться і для вхідних даних.

Дана нейронна мережа складається з 4-ох шарів:

1. Містить 40 нейронів, так як маємо 40 вхідних сигналів;
2. Містить 9228 нейронів. які ініціалізовані ваговими коефіцієнтами, взятими з початкової вибірки;
3. Містить 2 нейрони:  
   - 1 клас відноситься до нормального підключення  
   - 2 клас відноситься до кібератаки neptune
4. Містить 1 нейрон, в якому відбувається порівняння даних, які прийшли з 3 шару.

Результат виконання на основі наших вхідних даних виводиться на екран.

# **Додаток**

# **Додаток А (код програми)**

from math import exp  
  
type = "neptune"  
  
f = open("kddcup.data\_10\_percent\_corrected", "r")  
data = f.read()  
f.close()  
q = data.split(".\n")  
for i in range(len(q)):  
 q[i] = q[i].split(",")  
  
  
def norm\_text(index):  
 to\_list = list(normal[index])  
 l = len(to\_list) - 1  
 word\_norm = {}  
 for i in range(len(to\_list)):  
 word\_norm[to\_list[i]] = round(i\*(1/l), 2)  
 return word\_norm  
  
  
normal = [{"min": int(q[0][0]), "max": int(q[0][0])}, set(), set(), set(), {"min": int(q[4][0]), "max": int(q[4][0])},  
 {"min": int(q[5][0]), "max": int(q[5][0])}, {"min": int(q[6][0]), "max": int(q[6][0])}, {"min": int(q[7][0]), "max": int(q[7][0])},  
 {"min": int(q[8][0]), "max": int(q[8][0])}, {"min": int(q[9][0]), "max": int(q[9][0])}, {"min": int(q[10][0]), "max": int(q[10][0])},  
 {"min": int(q[11][0]), "max": int(q[11][0])}, {"min": int(q[12][0]), "max": int(q[12][0])}, {"min": int(q[13][0]), "max": int(q[13][0])},  
 {"min": int(q[14][0]), "max": int(q[14][0])}, {"min": int(q[15][0]), "max": int(q[15][0])}, {"min": int(q[16][0]), "max": int(q[16][0])},  
 {"min": int(q[17][0]), "max": int(q[17][0])}, {"min": int(q[18][0]), "max": int(q[18][0])}, {"min": int(q[19][0]), "max": int(q[19][0])},  
 {"min": int(q[20][0]), "max": int(q[20][0])}, {"min": int(q[21][0]), "max": int(q[21][0])}, {"min": int(q[22][0]), "max": int(q[22][0])},  
 {"min": int(q[23][0]), "max": int(q[23][0])}, {"min": float(q[24][0]), "max": float(q[24][0])}, {"min": float(q[25][0]), "max": float(q[25][0])},  
 {"min": float(q[26][0]), "max": float(q[26][0])}, {"min": float(q[27][0]), "max": float(q[27][0])}, {"min": float(q[28][0]), "max": float(q[28][0])},  
 {"min": float(q[29][0]), "max": float(q[29][0])}, {"min": float(q[30][0]), "max": float(q[30][0])}, {"min": int(q[31][0]), "max": int(q[31][0])},  
 {"min": int(q[32][0]), "max": int(q[32][0])}, {"min": float(q[33][0]), "max": float(q[33][0])}, {"min": float(q[34][0]), "max": float(q[34][0])},  
 {"min": float(q[35][0]), "max": float(q[35][0])}, {"min": float(q[36][0]), "max": float(q[36][0])}, {"min": float(q[37][0]), "max": float(q[37][0])},  
 {"min": float(q[38][0]), "max": float(q[38][0])}, {"min": float(q[39][0]), "max": float(q[39][0])}, {"min": float(q[40][0]), "max": float(q[40][0])}, set()]  
  
for i in range(len(q) - 1, -1, -1):  
 if q[i][41] == type or q[i][41] == "normal":  
 for j in range(len(q[i])):  
 if j != 1 and j != 2 and j != 3 and j != 41:  
 if (j != 24 and j != 25 and j != 26 and j != 27 and j != 28 and j != 29 and j != 30 and j != 33  
 and j != 34 and j != 35 and j != 36 and j != 37 and j != 38 and j != 39 and j != 40):  
 q[i][j] = int(q[i][j])  
 if normal[j]["max"] < q[i][j]:  
 normal[j]["max"] = q[i][j]  
 if normal[j]["min"] > q[i][j]:  
 normal[j]["min"] = q[i][j]  
 else:  
 q[i][j] = float(q[i][j])  
 if normal[j]["max"] < q[i][j]:  
 normal[j]["max"] = q[i][j]  
 if normal[j]["min"] > q[i][j]:  
 normal[j]["min"] = q[i][j]  
 else:  
 normal[j].add(q[i][j])  
 else:  
 del q[i]  
 continue  
  
normal[1] = norm\_text(1)  
normal[2] = norm\_text(2)  
normal[3] = norm\_text(3)  
  
  
for i in range(len(q)):  
 for j in range(len(q[i]) - 1):  
 if j != 1 and j != 2 and j != 3:  
 if normal[j]["max"] != normal[j]["min"]:  
 q[i][j] = (q[i][j] - normal[j]["min"])/(normal[j]["max"] - normal[j]["min"])  
 else:  
 q[i][j] = normal[j][q[i][j]]  
  
  
inputt = "0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal"  
input\_data\_all = inputt.split(",")  
input\_data = input\_data\_all[:-1]  
# input\_data = [0.0, 0.0, 0.0, 0.0, 0.0, 0.00016448555203911654, 0, 0, 0, 0.0, 0, 0.0, 0, 0, 0, 0, 0.0, 0, 0, 0, 0, 0.0, 0.6666666666666666, 0.6666666666666666, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.00784313725490196, 0.1111111111111111, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0]  
for j in range(len(input\_data)):  
 if j != 1 and j != 2 and j != 3:  
 if normal[j]["max"] != normal[j]["min"]:  
 input\_data[j] = (float(input\_data[j]) - normal[j]["min"])/(normal[j]["max"] - normal[j]["min"])  
 else:  
 input\_data[j] = float(input\_data[j])  
 else:  
 input\_data[j] = normal[j][input\_data[j]]  
  
  
  
class Input\_L:  
 def activation(self, x):  
 self.y = x \* 1  
 return self.y  
  
  
class Image\_L:  
 def \_\_init\_\_(self, w):  
 self.w = w  
  
 def activation(self, x):  
 sum = 0  
 for i in range(len(self.w)):  
 sum += exp(-(self.w[i] - x[i])\*\*2/0.3\*\*2)  
 return sum  
  
  
class Add\_L:  
 def activation(self, x):  
 sum = 0  
 for i in x:  
 sum += i  
 res = sum / len(x)  
 return res  
  
  
class Output\_L:  
 def activation(self, x):  
 clas = "normal"  
 val = x[clas]  
 for i in x.items():  
 if i[1] > val:  
 clas = i[0]  
 return clas  
  
  
input\_list = []  
for i in range(len(input\_data)):  
 input\_list.append(Input\_L)  
  
image\_list = []  
for i in range(len(q)):  
 image\_list.append(Image\_L(q[i][:-1]))  
  
class\_A = Add\_L()  
class\_B = Add\_L()  
  
res = Output\_L()  
  
y\_input = []  
y\_image = []  
y\_add = {}  
  
for i in range(len(input\_data)):  
 y\_input.append(input\_list[i].activation(input\_list[i], input\_data[i]))  
  
y\_image.append([])  
y\_image.append([])  
for i in range(len(q)):  
 if q[i][41] == "normal":  
 y\_image[0].append(image\_list[i].activation(y\_input))  
 else:  
 y\_image[1].append(image\_list[i].activation(y\_input))  
  
  
  
y\_add.update({"normal": class\_A.activation(y\_image[0])})  
y\_add.update({type: class\_B.activation(y\_image[1])})  
  
res = res.activation(y\_add)  
  
print("Класс",res)

# **Додаток Б (скріншоти результату)**

