НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ**​** ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**​**

Факультет прикладної математики

Кафедра прикладної математики

Звіт

Лабораторна робота №3

із дисципліни «Методи штучного інтелекту»

Тема: « Нейромережеве розпізнавання кібератак»

Виконав:

студент групи КМ-71

Агафонов Д. С.

Керівник:

Терейковська Л.О.

Київ **​**—**​**2020

Зміст

[Постановка задачі 3](#_Toc56383351)

[Характеристика виборки 4](#_Toc56383352)

[Опис реалізації розробленого модуля 10](#_Toc56383353)

[Опис та результати експериментальних досліджень 13](#_Toc56383354)

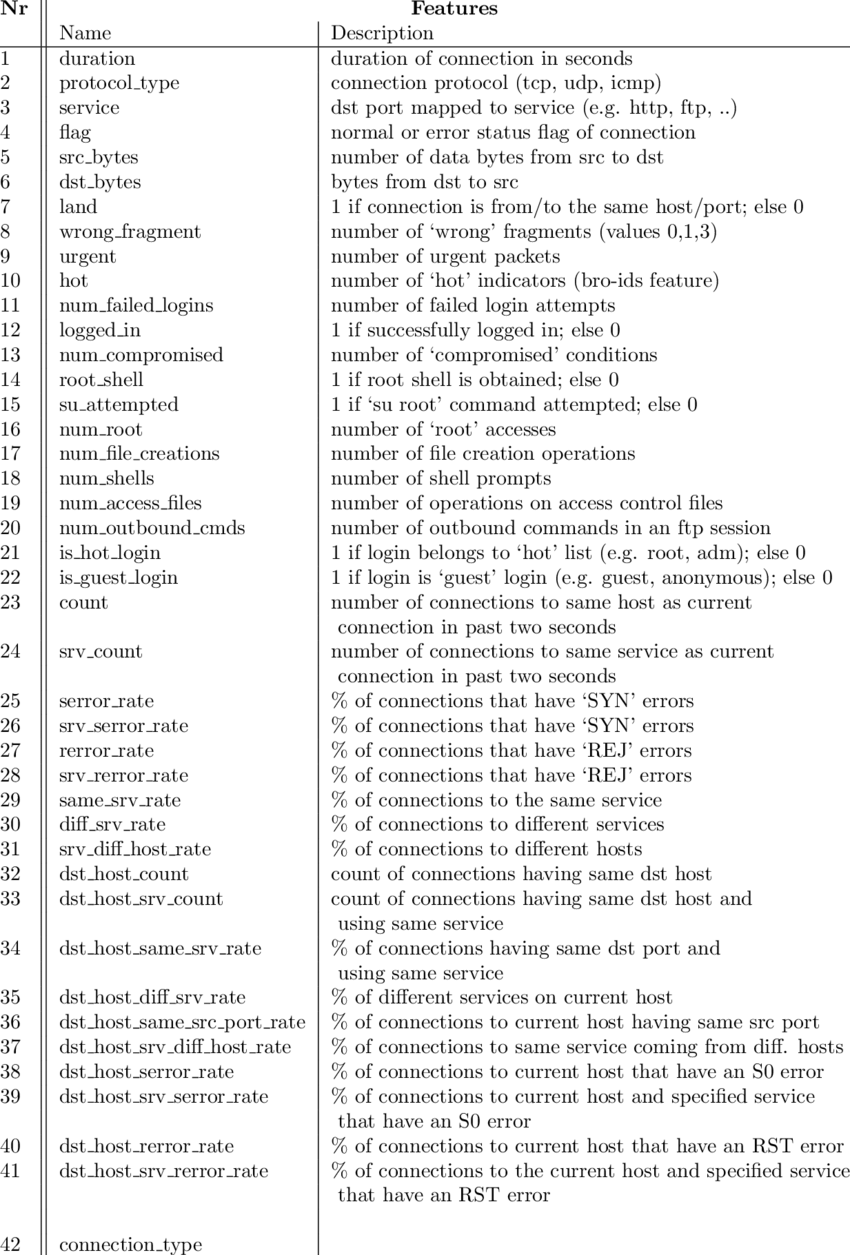
[Висновки 15](#_Toc56383355)

[Лістинг 16](#_Toc56383356)

# Постановка задачі

*Завдання*: Розробка програмного забезпечення для реалізації нейронної мережі типу ДШП, призначених для розпізнавання кібератак, сигнатури яких представлені в базі даних KDD-99.

# Характеристика виборки



Джерело даних: *The UCI KDD Archive Information and Computer Science University of California, Irvine*

Типи даних з виборки:

dtypes: float64(15), int64(23), object(4)

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 duration 494020 non-null int64

1 protocol\_type 494020 non-null object

2 service 494020 non-null object

3 flag 494020 non-null object

4 src\_bytes 494020 non-null int64

5 dst\_bytes 494020 non-null int64

6 land 494020 non-null int64

7 wrong\_fragment 494020 non-null int64

8 urgent 494020 non-null int64

9 hot 494020 non-null int64

10 num\_failed\_logins 494020 non-null int64

11 logged\_in 494020 non-null int64

12 lnum\_compromised 494020 non-null int64

13 lroot\_shell 494020 non-null int64

14 lsu\_attempted 494020 non-null int64

15 lnum\_root 494020 non-null int64

16 lnum\_file\_creations 494020 non-null int64

17 lnum\_shells 494020 non-null int64

18 lnum\_access\_files 494020 non-null int64

19 lnum\_outbound\_cmds 494020 non-null int64

20 is\_host\_login 494020 non-null int64

21 is\_guest\_login 494020 non-null int64

22 count 494020 non-null int64

23 srv\_count 494020 non-null int64

24 serror\_rate 494020 non-null float64

25 srv\_serror\_rate 494020 non-null float64

26 rerror\_rate 494020 non-null float64

27 srv\_rerror\_rate 494020 non-null float64

28 same\_srv\_rate 494020 non-null float64

29 diff\_srv\_rate 494020 non-null float64

30 srv\_diff\_host\_rate 494020 non-null float64

31 dst\_host\_count 494020 non-null int64

32 dst\_host\_srv\_count 494020 non-null int64

33 dst\_host\_same\_srv\_rate 494020 non-null float64

34 dst\_host\_diff\_srv\_rate 494020 non-null float64

35 dst\_host\_same\_src\_port\_rate 494020 non-null float64

36 dst\_host\_srv\_diff\_host\_rate 494020 non-null float64

37 dst\_host\_serror\_rate 494020 non-null float64

38 dst\_host\_srv\_serror\_rate 494020 non-null float64

39 dst\_host\_rerror\_rate 494020 non-null float64

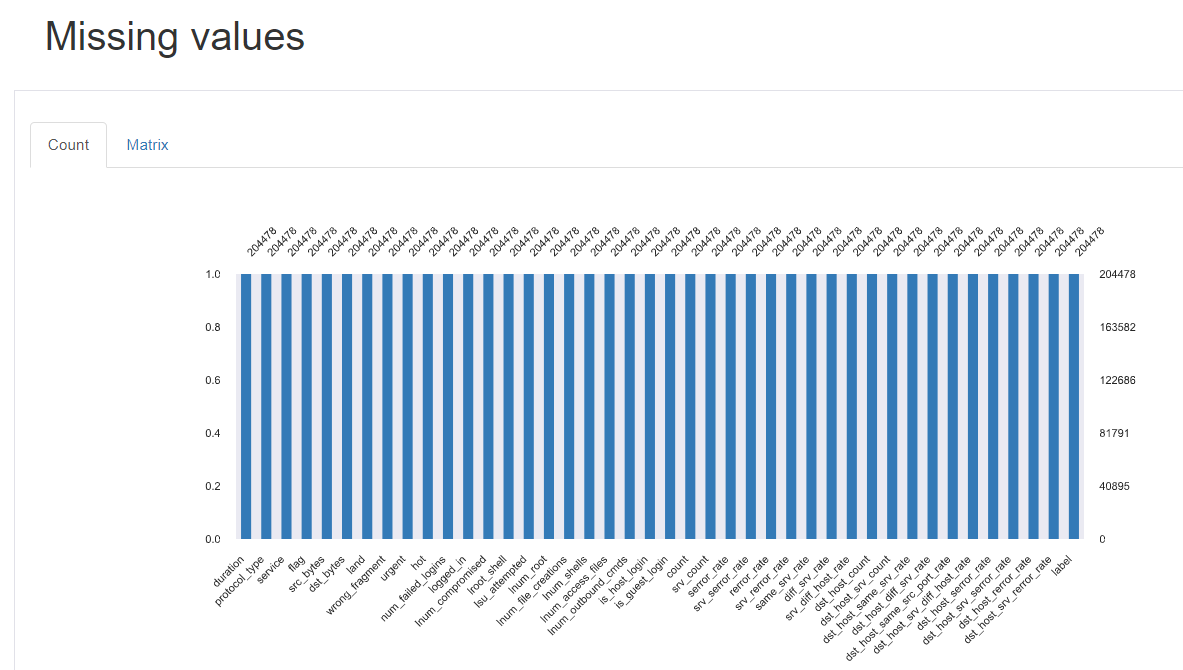
40 dst\_host\_srv\_rerror\_rate 494020 non-null float64

41 label 494020 non-null object

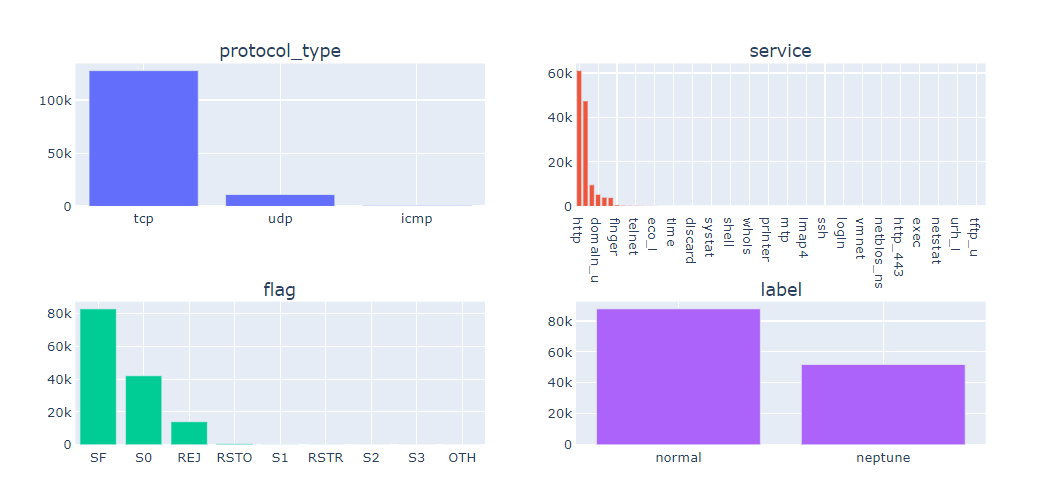
Типи змінних:

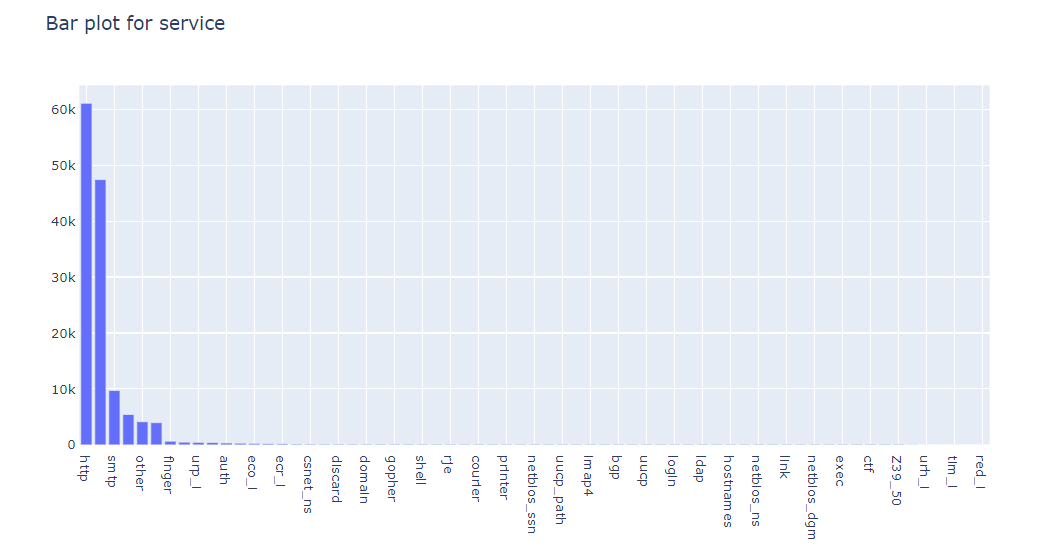
* номінальні бінарні: 7
* мультиномінальні: 3
* кількісні: 32

Зі звіту отриманого за допомогою бібліотеки pandas profiling пропущених значень в даних немає:

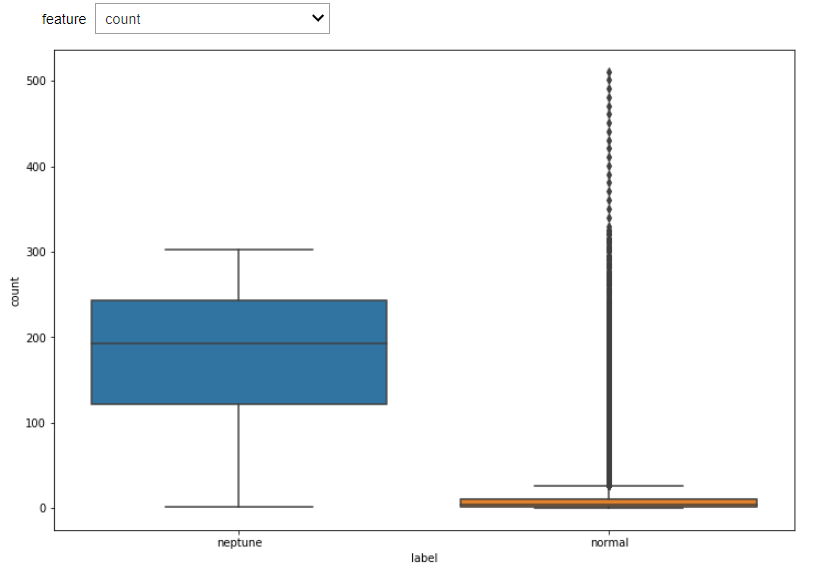


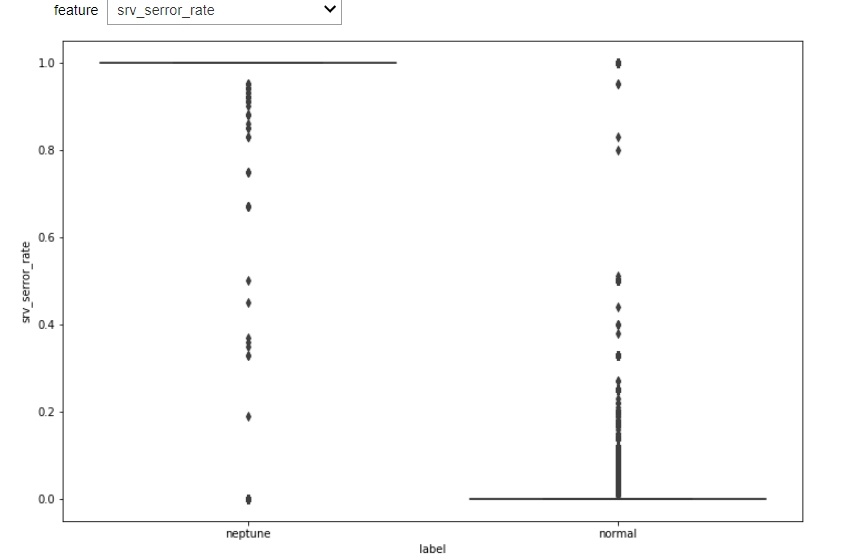
Стовбчикові діаграми ознак:

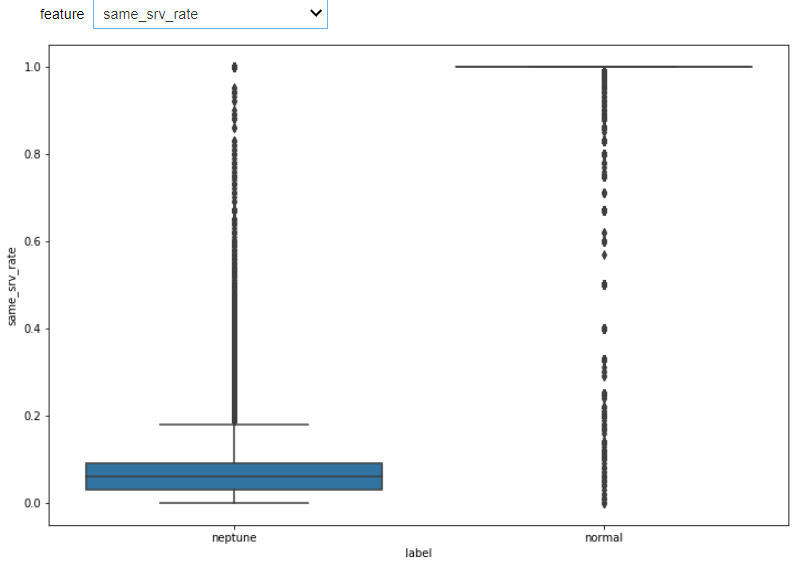




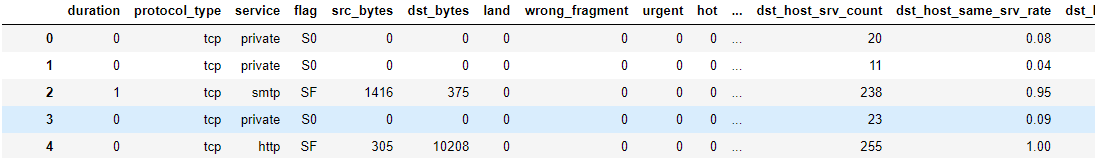
Box plot для деяких ознак:

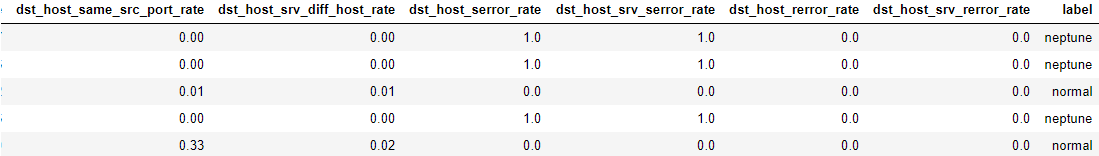






Початковий вигляд даних:





Label Encoding та One-hot Encoding

Використаємо Label кодування для переведення значень ознак та міток зі строкового в числовий вигляд:

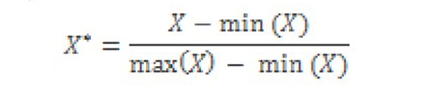
0: 'neptune', 1: 'normal'

One-hot кодування створює окремі стовбці-ознаки для перекодованих категоріальних даних.

Кодування очікуваного вихідного сигналу:

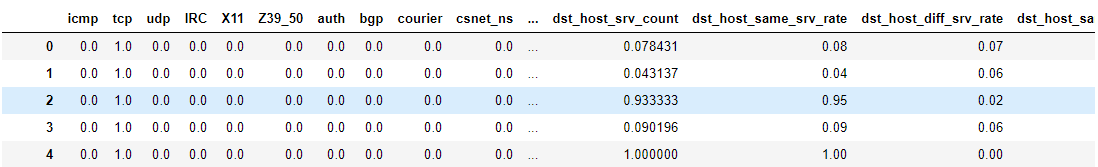
(0..0,5] – є кібератака; (0,5..1) – немає кібератаки.

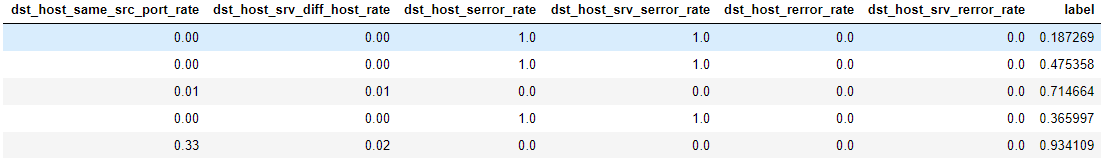
Нормалізація даних за допомогою MinMaxScaler: дані [0, 1]



де X\* - це нормалізоване значення, X – значення до нормалізації, min(X) – мінімальне значення параметра, max(X) – максимальне значення параметра.

Фінальний вигляд даних:





# Опис реалізації розробленого модуля

Для розбиття даних на тренувальну та тестову виборку використовуємо функцію train\_test\_split

X, y = dataset.drop(columns='label'), dataset[['label']]

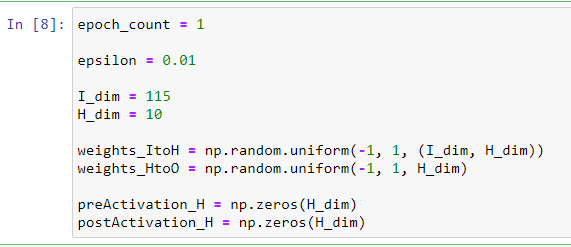
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

Отримуємо 4 виборки X\_train, X\_test, y\_train, y\_test. На X\_train та y\_train навчається ДШП, X\_test подається на вхід нейронної мережі, відбувається класифікація даних , результати порівнюються з y\_test.

Для активації ми використовуємо функцію логістичної сигмоиди.



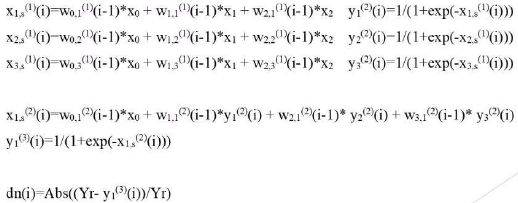
Ми вибираємо, розмірність вхідного шару, розмірність прихованого шару і кількість епох.



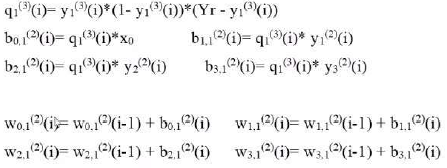
Функція np.random.uniform () заповнює матриці ваг випадковими значеннями від -1 до +1. Створюємо порожні масиви для значень преактіваціі і постактивации в прихованому шарі.

Перший цикл for дозволяє нам проходити через кілька епох. Усередині кожної епохи, в другому циклі for, по черзі проходячи по вибірках, ми обчислюємо вихідне значення для кожної вибірки (тобто сигнал постактивации вихідного вузла). У третьому циклі for ми звертаємося індивідуально до кожного прихованого вузлу, використовуючи скалярний добуток для генерування сигналу преактіваціі і функцію активації для генерування сигналу постактивации.

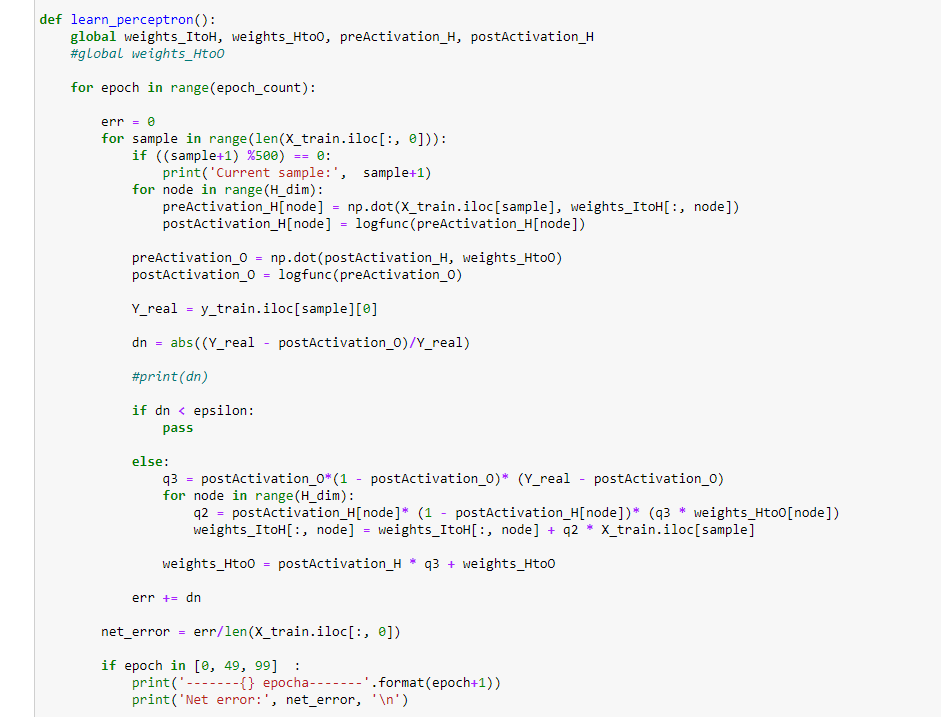
Після цього ми готові обчислити сигнал преактіваціі для вихідного вузла (знову використовуючи скалярний добуток), і ми застосовуємо функцію активації для генерування сигналу постактивации. Потім обчислюємо помилку ми віднімаємо цільове значення із значення отриманого сигналу постактивации вихідного вузла: dn = abs((Y\_real - postActivation\_O)/Y\_real).



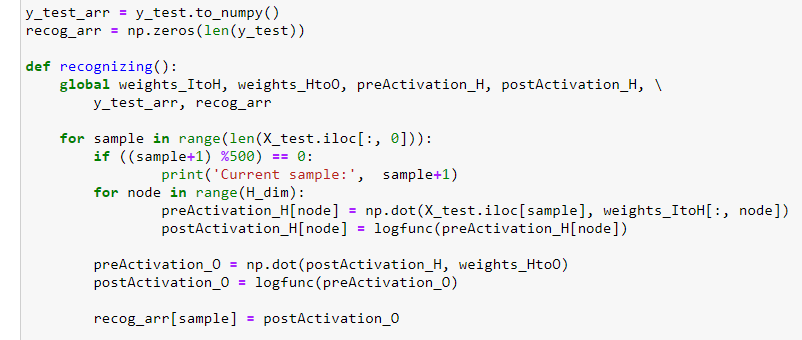
Якщо помилка <= епсілон, виконується переход на наступну строку виборки. Інакше виконуємо алгоритм зворотнього розповсюдження помилки за формулами:



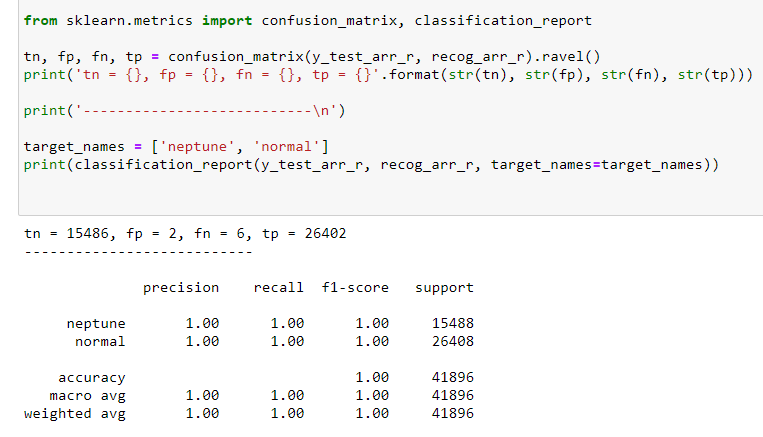
Алгоритм навчання:



Алгоритм розпізнавання:



Метрики:

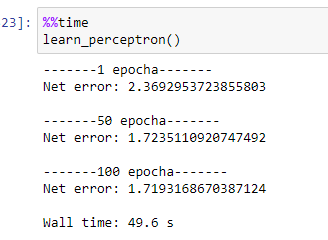


# Опис та результати експериментальних досліджень

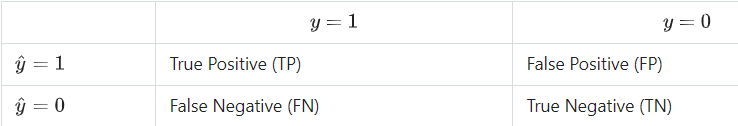
Навчання проводилося за 1 епоху з послідовним корегуванням вагів на кожному рядку з виборки.

На вхід подається датасет з 90 рядків та 41 ознак. Кількість hidden neurons дорівнює 3.

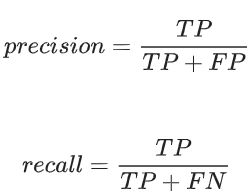
Алгоритм проходить 100 епох, помилки на 1, 50 та 100 епосі:

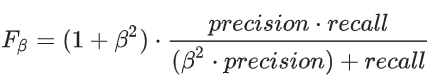


В результаті розпізнавання даних в ДШП можна отримати confusion matrix (матрицю помилок)



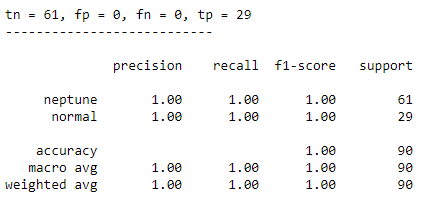
та класифікаційний зміст (precision, recall, f1-score)





beta = 1

Результат:



Отже, модель дуже точна (accuracy =1), дуже гарно знаходить класи (recall = 1) та дуже гарно класифікує об’єкти по класам (precision =1)

Hardware: Intel Core i5-5300U 2.3 ГГц

Термін навчання:

* learning

49s

* recognizing

75ms

# Висновки

В ході виконання лабораторної роботи виконана розробка програмного забезпечення для реалізації нейронної мережі типу ДШП, призначених для розпізнавання кібератак, сигнатури яких представлені в базі даних KDD-99. Проведена попередня обробка даних, кодування. На тренувальній вибірці навчено ДШП та проведено процедуру розпізнавання на тестовій. Отримані метрики.

# Лістинг

lab3\_data.ipynb

import pandas as pd

import numpy as np

pandas\_profiling

import random

%matplotlib inline

import matplotlib.pyplot as plt

from pylab import rcParams

rcParams['figure.figsize'] = 12, 8

import warnings

warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, MinMaxScaler

from sklearn.compose import ColumnTransformer

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('data2\kddcup99\_csv.csv')

data.info()

df1, df2 = data[data['label'] =='normal'].reset\_index(drop=True), data[data['label'] =='neptune'].reset\_index(drop=True)

df = df1.append(df2)

df =df.sample(frac=1, random\_state=42).reset\_index(drop=True) #mix data

df

profile = pandas\_profiling.ProfileReport(df, minimal=True)

profile.to\_file('output.html')

df.info()

dup = df[df.duplicated()]

dup.index

df = df.drop\_duplicates().reset\_index(drop=True)

from plotly.subplots import make\_subplots

import plotly.graph\_objects as go

categorical\_columns = df.select\_dtypes(include='object').columns.tolist()

numerical\_columns = df.select\_dtypes(include=np.number).columns.tolist()

fig = make\_subplots(rows=2, cols=2,

subplot\_titles=(categorical\_columns))

for i in range(len(categorical\_columns)):

fig.add\_trace(go.Bar(x=list(df[categorical\_columns[i]].value\_counts().to\_dict().keys()), y=list(df[categorical\_columns[i]].value\_counts().to\_dict().values())

),

i//2 +1, i%2+1)

fig.update\_layout( showlegend=False)

fig.show()

fig = go.Figure([go.Bar(x=list(df[categorical\_columns[1]].value\_counts().to\_dict().keys()),

y=list(df[categorical\_columns[1]].value\_counts().to\_dict().values())

)], layout=go.Layout(

title=go.layout.Title(text='Bar plot for '+categorical\_columns[1])))

#fig.update\_layout(subplot\_title = categorical\_columns[1])

fig.show()

from ipywidgets import interact #, interactive, fixed, interact\_manual, IntSlider

import seaborn as sns

excess\_features = ['land', 'wrong\_fragment', 'urgent', 'logged\_in', 'lroot\_shell', 'lsu\_attempted', 'lnum\_shells', 'is\_host\_login', 'is\_guest\_login' ]

numerical\_columns\_to\_plot = list(filter(lambda x: x not in excess\_features, numerical\_columns))

@interact

def test(feature = numerical\_columns\_to\_plot):

# y0 = df[col1].where(df['label'] == 'normal').dropna()

# y1 = df[col1].where(df['label'] == 'neptune').dropna()

sns.boxplot(y = feature, data = df, x='label')

label\_encoder = LabelEncoder()

df['label'] = label\_encoder.fit\_transform(df['label'])

dict(enumerate(label\_encoder.classes\_))

features\_encoder = LabelEncoder()

categorical\_columns = df.columns[df.dtypes == 'object']#.union(['service'])

label\_df = df[categorical\_columns]

categorial\_classes = []

for column in categorical\_columns:

label\_df[column] = features\_encoder.fit\_transform(df[column])

categorial\_classes.extend(features\_encoder.classes\_)

columns\_except\_categorical = list(filter(lambda x: x not in categorical\_columns, list(df.columns)))

new\_column\_names = categorial\_classes + columns\_except\_categorical

#len(new\_column\_names)

#from sklearn.compose import ColumnTransformer

col\_trans = ColumnTransformer([('encoder', OneHotEncoder(), categorical\_columns)], remainder='passthrough')

dataset = pd.DataFrame(col\_trans.fit\_transform(df), dtype = 'object')

old\_column\_names = np.array(dataset.columns)

dataset.rename(dict(zip(old\_column\_names, new\_column\_names)), axis=1, inplace=True)

#import random

np.random.seed(42)

label\_zero = dataset[['label']].where(dataset['label'] == 0).dropna()

label\_one = dataset[['label']].where(dataset['label'] == 1).dropna()

label\_zero['label'] = (0.5- 0)\*np.random.random(len(label\_zero))

label\_one['label'] = (1- 0.5)\*np.random.random(len(label\_one)) + 0.5

result = pd.concat([label\_one, label\_zero ]).sort\_index()

#result.sort\_index(inplace=True)

dataset['label'] = result

#from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

#new\_column\_names

dataset[new\_column\_names] = scaler.fit\_transform(dataset[new\_column\_names])

dataset

path ='path'

dataset.to\_csv(path +'final\_data.csv', index=False, header=True)

lab3\_percep.ipynb

import warnings

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split

%%time

dataset = pd.read\_csv(r'data2\final\_data.csv')

dataset

X, y = dataset.drop(columns='label'), dataset[['label']]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_train

def logfunc(x):

return 1.0/(1 + np.exp(-x))

epoch\_count = 1

epsilon = 0.01

I\_dim = 115

H\_dim = 10

weights\_ItoH = np.random.uniform(-1, 1, (I\_dim, H\_dim))

weights\_HtoO = np.random.uniform(-1, 1, H\_dim)

preActivation\_H = np.zeros(H\_dim)

postActivation\_H = np.zeros(H\_dim)

# learning

def learn\_perceptron():

global weights\_ItoH, weights\_HtoO, preActivation\_H, postActivation\_H

#global weights\_HtoO

for epoch in range(epoch\_count):

for sample in range(len(X\_train.iloc[:, 0])):

if ((sample+1) %500) == 0:

print('Current sample:', sample+1)

for node in range(H\_dim):

preActivation\_H[node] = np.dot(X\_train.iloc[sample], weights\_ItoH[:, node])

postActivation\_H[node] = logfunc(preActivation\_H[node])

preActivation\_O = np.dot(postActivation\_H, weights\_HtoO)

postActivation\_O = logfunc(preActivation\_O)

Y\_real = y\_train.iloc[sample][0]

dn = abs((Y\_real - postActivation\_O)/Y\_real)

if dn < epsilon:

pass

else:

q3 = postActivation\_O\*(1 - postActivation\_O)\* (Y\_real - postActivation\_O)

for node in range(H\_dim):

q2 = postActivation\_H[node]\* (1 - postActivation\_H[node])\* (q3 \* weights\_HtoO[node])

weights\_ItoH[:, node] = weights\_ItoH[:, node] + q2 \* X\_train.iloc[sample]

weights\_HtoO = postActivation\_H \* q3 + weights\_HtoO

%%time

learn\_perceptron()

#recognizing

y\_test\_arr = y\_test.to\_numpy()

recog\_arr = np.zeros(len(y\_test))

def recognizing():

global weights\_ItoH, weights\_HtoO, preActivation\_H, postActivation\_H, \

y\_test\_arr, recog\_arr

for sample in range(len(X\_test.iloc[:, 0])):

if ((sample+1) %500) == 0:

print('Current sample:', sample+1)

for node in range(H\_dim):

preActivation\_H[node] = np.dot(X\_test.iloc[sample], weights\_ItoH[:, node])

postActivation\_H[node] = logfunc(preActivation\_H[node])

preActivation\_O = np.dot(postActivation\_H, weights\_HtoO)

postActivation\_O = logfunc(preActivation\_O)

recog\_arr[sample] = postActivation\_O

%%time

recognizing()

# round

y\_test\_arr\_r = np.around(y\_test\_arr).reshape(-1)

recog\_arr\_r = np.around(recog\_arr)

# metrics

from sklearn.metrics import confusion\_matrix, classification\_report

tn, fp, fn, tp = confusion\_matrix(y\_test\_arr\_r, recog\_arr\_r).ravel()

print('tn = {}, fp = {}, fn = {}, tp = {}'.format(str(tn), str(fp), str(fn), str(tp)))

print('---------------------------\n')

target\_names = ['neptune', 'normal']

print(classification\_report(y\_test\_arr\_r, recog\_arr\_r, target\_names=target\_names))