МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| д-р. техн. наук, профессор |  |  |  | Ю. А. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ | | | | | |
| РАЗРАБОТКА АНСАМБЛЯ НЕЙРОННЫХ СЕТЕЙ ТИПА БЭГГИНГ НА ОСНОВЕ СВЕРТОЧНОЙ НЕЙРОСЕТЕВОЙ МОДЛИ С ПОЛНОСВЯЗНЫМ КЛАССИФИКАТОРОМ | | | | | |
| по дисциплине: ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ И ОБРАБОТКА ДАННЫХ | | | | | |
|  | | | | | |
|  | | | | | |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232М |  |  |  | В. Ф. Губайдулин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

СОДЕРЖАНИЕ

[РАЗРАБОТКА АНСАМБЛЯ НЕЙРОННЫХ СЕТЕЙ ТИПА БЭГГИНГ НА ОСНОВЕ СВЕРТОЧНОЙ НЕЙРОСЕТЕВОЙ МОДЛИ С ПОЛНОСВЯЗНЫМ КЛАССИФИКАТОРОМ 1](#_Toc151487183)

[Введение 3](#_Toc151487184)

# Введение

В качестве набора данных был получен набор данных сетевого трафика KDD Cup 1999 Data. KDD Cup 1999 Data – это набор данных, содержащий информацию о сетевом трафике, собранном в ходе конференции KDD-99 по знаниям и данным. Набор данных включает в себя различные типы трафика, такие как HTTP, SMTP, FTP и т.д., а также атрибуты, связанные с каждым соединением, такие как время начала и окончания соединения, количество переданных байтов и т.д. Этот набор данных часто используется для обучения алгоритмов обнаружения аномалий и иных задач в области информационной безопасности. С помощью данного набора необходимо реализовать приложение на языке Python и библиотек машинного и глубокого обучения на основе классификационной модели, которое будет призвано определять попытки сетевых вторжений.

При реализации приложения будет необходимо выполнить разведочный анализ данных, найти гиперпараметры модели с помощью генетического алгоритма и метода greedsearch, обучить модель с помощью автокодировщика.

Был выбран вариант 29: ансамбль нейронных сетей типа бэггинг на основе сверточной нейросетевой модели с полносвязным классификатором.

**Цель:** разработка приложения на языке Python и необходимых библиотек машинного и глубокого обучения на основе классификационной модели, которое определяет попытки сетевых вторжений на основе данных сетевого трафика.

**Задачи:**

* выполнить разведочный анализ;
* проверить набор данных на сбалансированность классов;
* для выбранной комбинации набора данных и предиктивной модели выполнить поиск гиперпараметров с помощью генетического алгоритма;
* для выбранной комбинации набора данных и предиктивной модели выполнить поиск гиперпараметров с помощью метода greedsearch;
* на неразмеченной части набора данных выполнить предобучение с помощью автокодировщика, построенного на основании базового нейросетевого классификатора, заданного в варианте, а затем выполнить точную настройку базовой сети на размеченном наборе данных.

# 1 Разведочный анализ данных

Были установлены библиотеки и пакеты необходимые для проведения разведочного анализа.

Листинг 1 – Библиотеки для проведения разведочного анализа

%matplotlib inline

import matplotlib.pyplot as plt

import os

import pandas as pd

from pandas.plotting import scatter\_matrix

import numpy as np

import seaborn as sns

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

from sklearn.preprocessing import StandardScaler, PolynomialFeatures, LabelEncoder

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

from sklearn.linear\_model import ElasticNet, LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

Были объявлены константы, необходимые для для проведения разведочного анализа.

Листинг 2 - Константы, которые необходимы для реализации для проведения разведочного анализа

\_INCLUDED\_CLASSES = ['normal.', 'smurf.']

Был загружен размеченный датасет.

Листинг 3 – Загрузка размеченного датасета

LABELED\_DATASET = 'C:\\Users\\Gubay\\OneDrive\\Documents\\Archive\_University\\Мага\_3\\ml\_course\_work\\datasets\\kdd\_10000\_labeled.csv'

labled\_data = pd.read\_csv(LABELED\_DATASET)

Было произведено ознакомление с данными с помощью методов описательной статистики. Размерность набора данных составила (10000, 42). Типы атрибутов в данных:

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

class object

dtype: object

Краткое описание данных:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 42 columns):

# Column Non-Null Count Dtype

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

0 duration 10000 non-null int64

1 protocol\_type 10000 non-null object

2 service 10000 non-null object

3 flag 10000 non-null object

4 src\_bytes 10000 non-null int64

5 dst\_bytes 10000 non-null int64

6 land 10000 non-null int64

7 wrong\_fragment 10000 non-null int64

8 urgent 10000 non-null int64

9 hot 10000 non-null int64

10 num\_failed\_logins 10000 non-null int64

11 logged\_in 10000 non-null int64

12 num\_compromised 10000 non-null int64

13 root\_shell 10000 non-null int64

14 su\_attempted 10000 non-null int64

15 num\_root 10000 non-null int64

16 num\_file\_creations 10000 non-null int64

17 num\_shells 10000 non-null int64

18 num\_access\_files 10000 non-null int64

19 num\_outbound\_cmds 10000 non-null int64

20 is\_host\_login 10000 non-null int64

21 is\_guest\_login 10000 non-null int64

22 count 10000 non-null int64

23 srv\_count 10000 non-null int64

24 serror\_rate 10000 non-null float64

25 srv\_serror\_rate 10000 non-null float64

26 rerror\_rate 10000 non-null float64

27 srv\_rerror\_rate 10000 non-null float64

28 same\_srv\_rate 10000 non-null float64

29 diff\_srv\_rate 10000 non-null float64

30 srv\_diff\_host\_rate 10000 non-null float64

31 dst\_host\_count 10000 non-null int64

32 dst\_host\_srv\_count 10000 non-null int64

33 dst\_host\_same\_srv\_rate 10000 non-null float64

34 dst\_host\_diff\_srv\_rate 10000 non-null float64

35 dst\_host\_same\_src\_port\_rate 10000 non-null float64

36 dst\_host\_srv\_diff\_host\_rate 10000 non-null float64

37 dst\_host\_serror\_rate 10000 non-null float64

38 dst\_host\_srv\_serror\_rate 10000 non-null float64

39 dst\_host\_rerror\_rate 10000 non-null float64

40 dst\_host\_srv\_rerror\_rate 10000 non-null float64

41 class 10000 non-null object

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

memory usage: 3.2+ MB

Листинг 4 – Получение данных с помощью методов описательной статистики

labled\_data.head() # первые 5 строк

labled\_data.shape # размерность набора данных

labled\_data.dtypes # типы атрибутов в данных

labled\_data.info() # получение краткого описания данных

Исходя из полученных первичных данных описательной статистики необходимо удалить значения в атрибуте *class*, которые не равны ‘normal’ или ‘smurf’. А также, необходимо перевести атрибут class в значение типа *int*, где 0 – ‘normal’, 1 – ‘smurf’. По итогу получились следующие типы данных:

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

class int32

dtype: object

Листинг 5 – Редактирование исходных данных в атрибуте *class*

# Необходимо удалить значения в атрибуте class,

# которые не равны 'normal.' или 'smurf.'

labled\_data = labled\_data[(labled\_data['class'] == \_INCLUDED\_CLASSES[0]) | (labled\_data['class'] == \_INCLUDED\_CLASSES[1])]

# Необходим перевод атриубта class в значение типа int, где

# 0 - normal, 1- smurf

labled\_data['class'] = labled\_data['class'].replace({

'normal.': 0,

'smurf.': 1})

labled\_data['class'] = labled\_data['class'].astype(int)

labled\_data.dtypes # типы атрибутов в данных

Возникла необходимость перекодировать значения в столбцах *protocol\_type*, *service*, *flag*.

Листинг 6 – Перекодирование значений в столбцах *protocol\_type*, *service*, *flag*

# Необходимо перекодировать значения в столбцах protocol\_type, service, flag

labled\_data\_encoded = labled\_data.copy(deep=True)

label\_encoder = LabelEncoder()

labled\_data\_encoded['protocol\_type\_encoded'] = label\_encoder.fit\_transform(labled\_data\_encoded['protocol\_type'])

labled\_data\_encoded['service\_encoded'] = label\_encoder.fit\_transform(labled\_data\_encoded['service'])

labled\_data\_encoded['flag\_encoded'] = label\_encoder.fit\_transform(labled\_data\_encoded['flag'])

labled\_data\_encoded = labled\_data\_encoded.drop('protocol\_type', axis=1)

labled\_data\_encoded = labled\_data\_encoded.drop('service', axis=1)

labled\_data\_encoded = labled\_data\_encoded.drop('flag', axis=1)

labled\_data = labled\_data\_encoded.copy(deep=True)

Были проведены одномерная и многомерная визуализация.

Была выведены гистограмма значений каждого атрибута.

Листинг 7 – Вывод гистограммы значений каждого атрибута

labled\_data.hist(bins=25, figsize=(20,15)) # 1D-графики: гистограммы

plt.show()

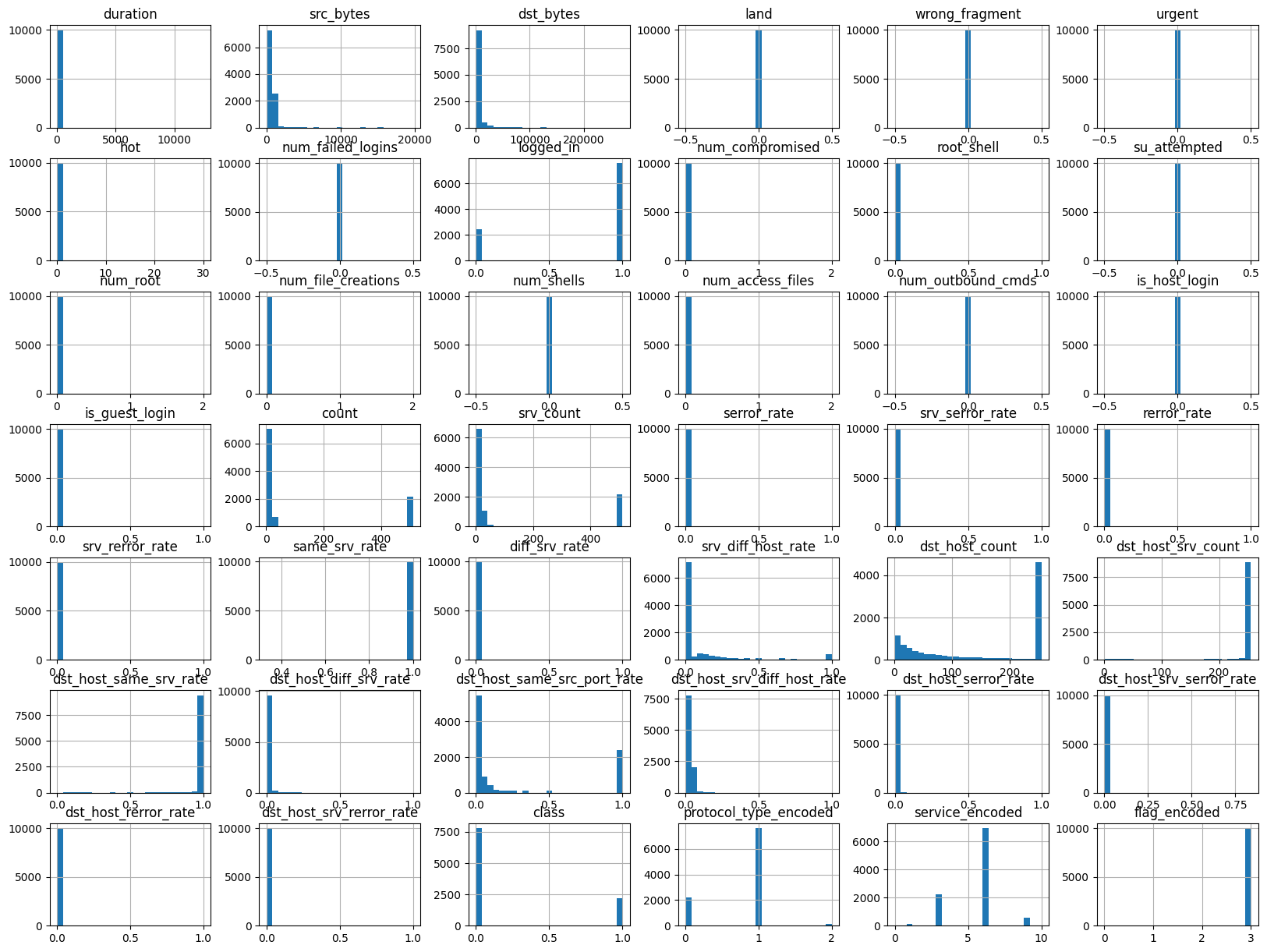


Рисунок 1 – Гистограмма значений каждого атрибута

Была построена гистограмма отдельно выбранных значений, которые имеют изменения в представленном датасете.

Листинг 8 – Отбор атрибутов и построение гистограммы

# Были выбраны следующие значения для детальной визуалзиаци, т.к.

# их значения изменяются в датасете

sub\_init\_labled\_data = pd.DataFrame({

'protocol\_type\_encoded': labled\_data['protocol\_type\_encoded'],

'service\_encoded': labled\_data['service\_encoded'],

'src\_bytes' : labled\_data["src\_bytes"],

'dst\_bytes': labled\_data["dst\_bytes"],

'logged\_in': labled\_data["logged\_in"],

'count': labled\_data["count"],

'srv\_count': labled\_data["srv\_count"],

'srv\_diff\_host\_rate': labled\_data["srv\_diff\_host\_rate"],

'srv\_diff\_host\_rate': labled\_data["srv\_diff\_host\_rate"],

'dst\_host\_count': labled\_data['dst\_host\_count'],

'dst\_host\_srv\_count': labled\_data['dst\_host\_srv\_count'],

'dst\_host\_same\_srv\_rate': labled\_data['dst\_host\_same\_srv\_rate'],

'dst\_host\_diff\_srv\_rate': labled\_data['dst\_host\_diff\_srv\_rate'],

'dst\_host\_same\_src\_port\_rate': labled\_data['dst\_host\_same\_src\_port\_rate'],

'dst\_host\_srv\_diff\_host\_rate': labled\_data['dst\_host\_srv\_diff\_host\_rate'],

'class': labled\_data['class']

})

sub\_init\_labled\_data.hist(bins=50, figsize=(10,5))

plt.show()

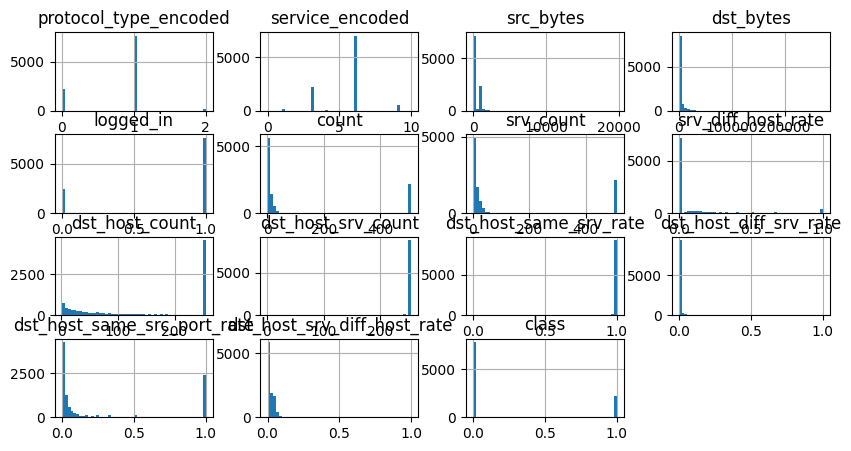


Рисунок 2 – Гистограмма отдельно выбранных значений

Был построен график распределения для уже отобранных значений.

Листинг 9 ­– Построение графика распределения

sub\_init\_labled\_data.plot(kind='density', subplots=True, sharex=False,\

legend=True, fontsize=1, figsize=(20,10))

plt.show()

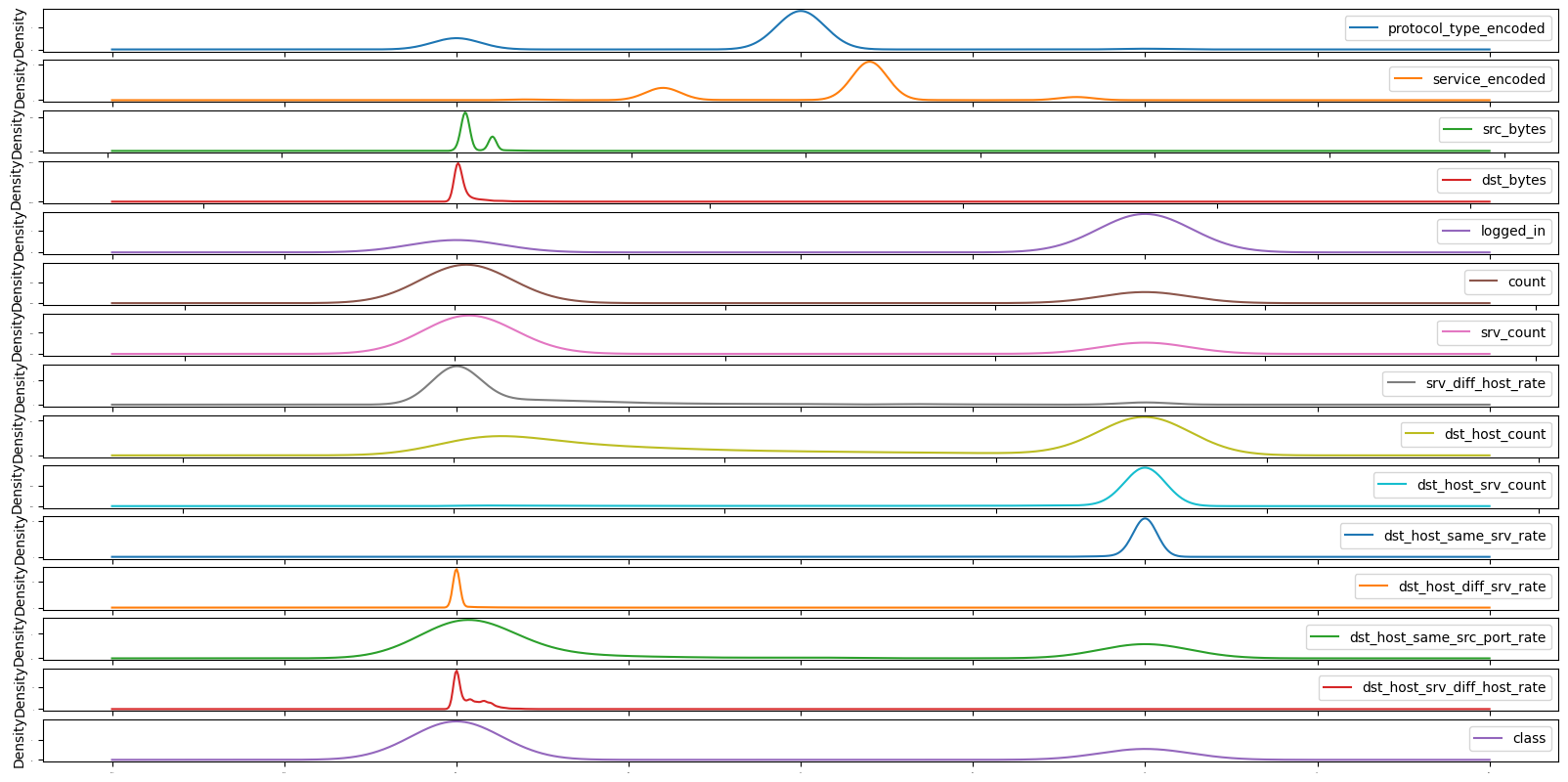


Рисунок 3 – Диаграмма распределения

Был построен график корреляционной матрицы.

Листинг 10 – Построение графика корреляционной матрицы

# многомерные графики

# график корреляционной матрицы

correlation = sub\_init\_labled\_data.corr()

plt.figure(figsize=(10, 10))

plt.title('Correlation matrix')

sns.heatmap(correlation, vmax=1, square=True, annot=True, cmap='cubehelix')

plt.show()

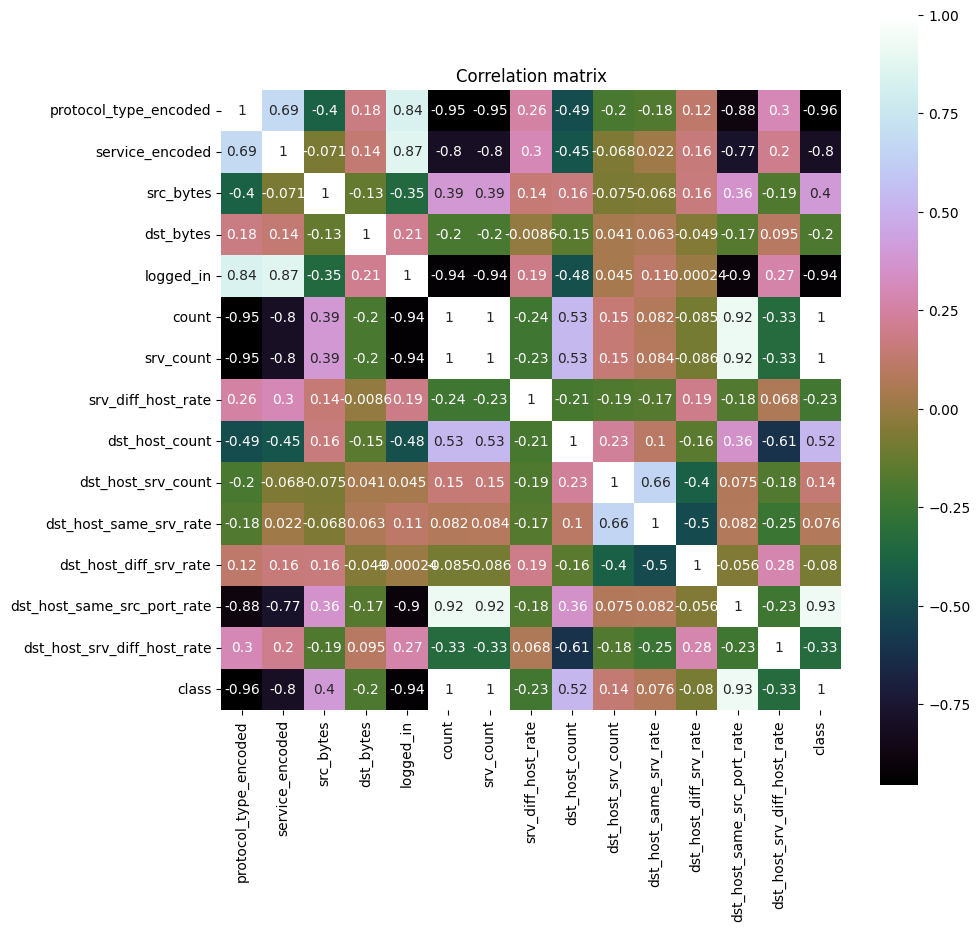


Рисунок 4 – График корреляционной матрицы

Были построены диаграмма рассеяния и диаграмма рассеяния с диагональю kde.

Листинг 11 – Построение диаграммы рассеяния и диаграммы рассеяния с диагональю kde

scatter\_matrix(sub\_init\_labled\_data, figsize=(20, 20)) # диаграмма рассеяния

plt.show()

# диаграмма рассяния для всех атрибутов с диагональю kde

scatter\_matrix(sub\_init\_labled\_data, figsize=(20, 20), diagonal='kde') # диаграмма рассеяния

plt.show()

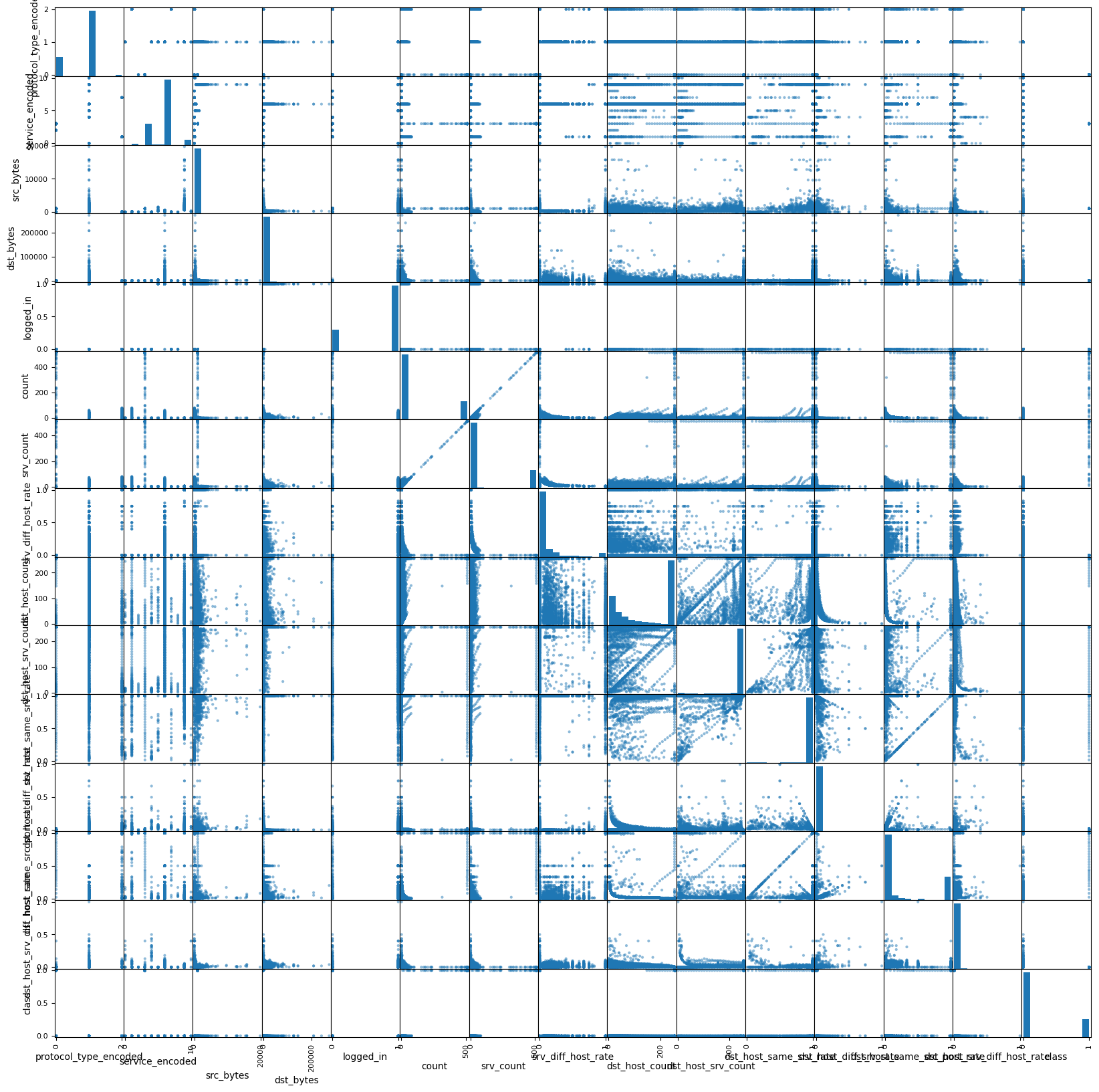


Рисунок 5 – Диаграмма рассеяния

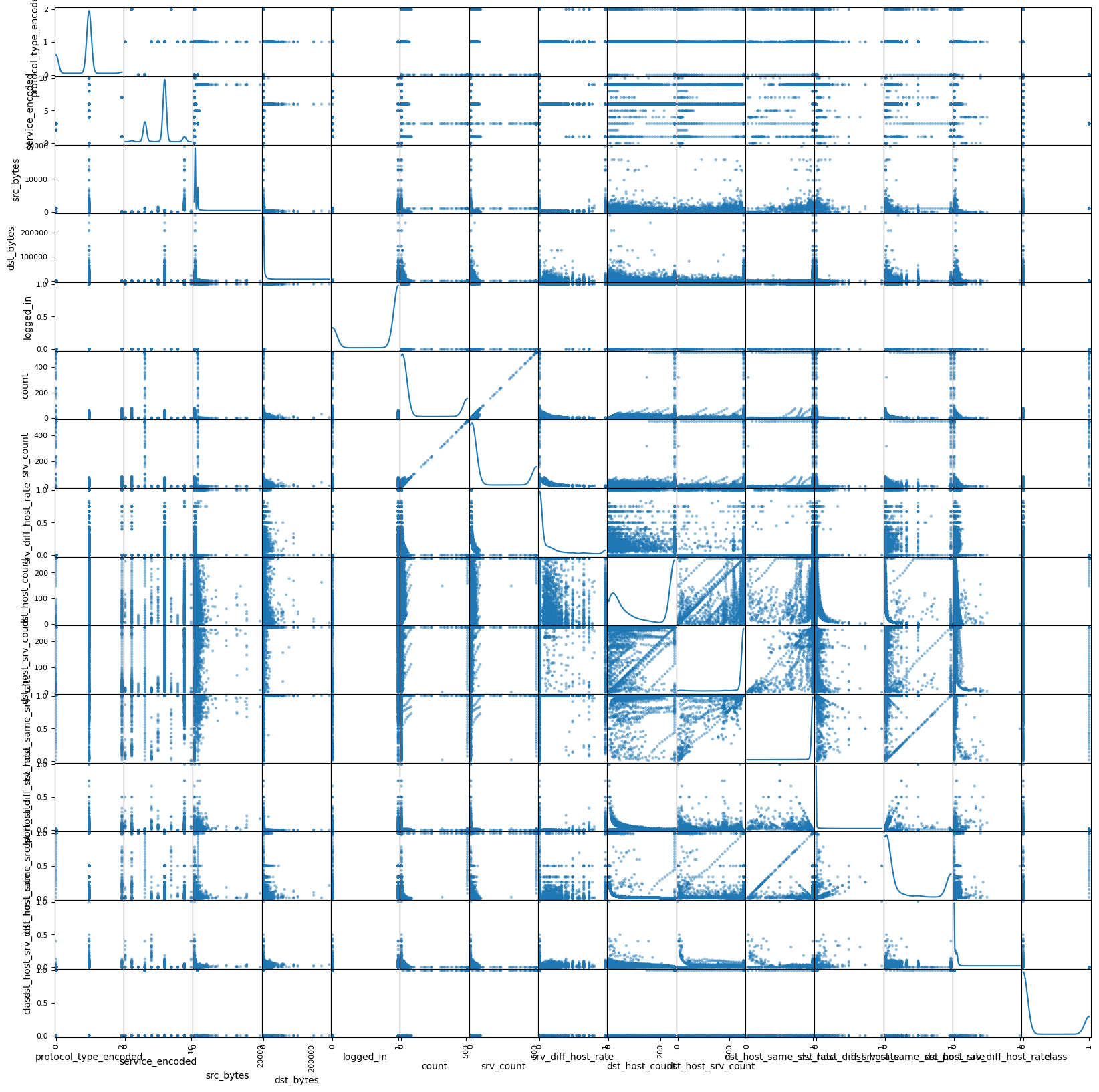


Рисунок 6 – Диаграмма рассеяния с диагональю kde

Была проанализирована корреляционная зависимость между признаками.

Листинг 12 – Построение корреляционной зависимости между признаками

correlation = sub\_init\_labled\_data.corr()

correlation

Атрибуты имеющие наибольшую положительную корреляцию к атрибуту class: count, srv\_count, dst\_host\_same\_src\_port\_rate, dst\_host\_count

Атрибуты имеющие наибольшую отрицательную корреляцию к атрибуту class: logged\_in

Был сформирован преобразованный исходный набор. Для чётности был добавлен атрибут *temp0*, который хранит в себе исключительно нули. Данный атрибут необходим для чётности при работе со свёрткой.

Листинг 13 – Формирование преобразованного исходного набора

labled\_result\_data = pd.DataFrame({

'protocol\_type\_encoded': labled\_data['protocol\_type\_encoded'],

'service\_encoded': labled\_data['service\_encoded'],

'logged\_in': labled\_data["logged\_in"],

'count': labled\_data["count"],

'srv\_count': labled\_data["srv\_count"],

'dst\_host\_count': labled\_data['dst\_host\_count'],

'dst\_host\_same\_src\_port\_rate': labled\_data['dst\_host\_same\_src\_port\_rate'],

'temp0': 0,

'class': labled\_data['class']

})

Формат данных преобразованного исходного набора:

<class 'pandas.core.frame.DataFrame'>

Index: 9994 entries, 0 to 9999

Data columns (total 9 columns):

# Column Non-Null Count Dtype

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

0 protocol\_type\_encoded 9994 non-null int32

1 service\_encoded 9994 non-null int32

2 logged\_in 9994 non-null int64

3 count 9994 non-null int64

4 srv\_count 9994 non-null int64

5 dst\_host\_count 9994 non-null int64

6 dst\_host\_same\_src\_port\_rate 9994 non-null float64

7 temp0 9994 non-null int64

8 class 9994 non-null int32

dtypes: float64(1), int32(3), int64(5)

memory usage: 663.7 KB

Новый набор данных был проверен на сбалансированность классов.

Листинг 13 – Проверка нового набора на сбалансированность классов

class\_value\_count = labled\_result\_data['class'].value\_counts().reset\_index()

print(sns.barplot(x = 'class', y = 'count', data = class\_value\_count))

print(class\_value\_count)

print('normal = 0\nsmurf = 1')

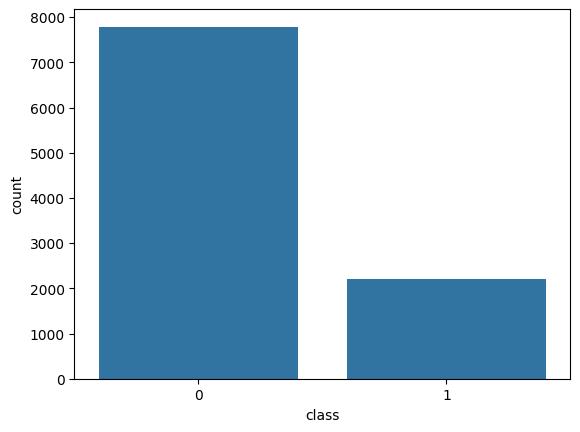


Рисунок 7 – Количество каждого класса

Набор не сбалансирован.

# 2 Построение предиктивной модели

Были реализованы методы с помощью, которых строит предиктивная модель. Данный метод в качестве входных параметров принимает датасет для обучения, количество фильтров для первого сверточного слоя, количество фильтров для второго сверточного слоя, размер свёртки, значения слоя droupout, количество нейронов в полносвязном слое и номер алгоритма. Ансамбль состоит из 5 моделей.

Листинг 14 – Метод построения ансамбля

import random

import tensorflow as tf

from keras import layers, models

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

import pandas as pd

from sklearn.preprocessing import StandardScaler

from metrics import balanced\_accuracy\_m, f1\_m, recall\_m, precision\_m

from roc\_score import get\_roc\_auc\_score

import numpy as np

def build\_ensemble(dataset, filters\_first\_layer, filters\_second\_layer, kernel\_size, dropout, dense\_units, algorithm\_enum,

is\_get\_final\_models=False, is\_plotting=False):

print('Start building ensemble...')

test\_size = 0.2

data\_x = dataset.iloc[:, 0:dataset.shape[1] - 1].values

data\_y = dataset.iloc[:, dataset.shape[1] - 1].values

scaler = StandardScaler()

df\_x\_st\_scaled = scaler.fit\_transform(data\_x)

data\_x = df\_x\_st\_scaled

# Создание и обучение ансамбля из 5 моделей

num\_models = 5

models\_list = []

for num\_model in range(num\_models):

seed = random.randint(1, 100)

df\_x\_train, df\_x\_test, df\_y\_train, df\_y\_test = (

train\_test\_split(data\_x, data\_y, test\_size=test\_size, random\_state=seed))

df\_x\_train = tf.cast(df\_x\_train, dtype=tf.float32)

df\_x\_test = tf.cast(df\_x\_test, dtype=tf.float32)

df\_y\_train = tf.cast(df\_y\_train, dtype=tf.float32)

df\_y\_test = tf.cast(df\_y\_test, dtype=tf.float32)

input\_shape = (df\_x\_train.shape[1], 1)

model = build\_model(filters\_first\_layer, filters\_second\_layer, kernel\_size, dropout, dense\_units, algorithm\_enum,

input\_shape)

model.fit(df\_x\_train, df\_y\_train, epochs=5, batch\_size=32, verbose=0)

models\_list.append(model)

if is\_get\_final\_models:

return models\_list

# Оценка ансамбля на тестовых данных

metrics\_data = ensemble\_predict(models\_list, df\_x\_test, df\_y\_test)

ensemble\_accuracy = metrics\_data[1]

ensemble\_f1 = metrics\_data[2]

ensemble\_precision = metrics\_data[3]

ensemble\_recall = metrics\_data[4]

ensemble\_balanced\_accuracy\_score = metrics\_data[5]

roc\_auc\_score = get\_roc\_auc\_score(models\_list, df\_x\_test, df\_y\_test, is\_plotting)

print('Ensemble Accuracy:', ensemble\_accuracy)

print('Ensemble Recall:', ensemble\_recall)

print('Ensemble Precision:', ensemble\_precision)

print('Ensemble F1:', ensemble\_f1)

print('Ensemble ensemble\_balanced\_accuracy\_score:', ensemble\_balanced\_accuracy\_score)

print('Ensemble ROC\_AUC\_SCORE:', roc\_auc\_score)

return ensemble\_balanced\_accuracy\_score

Был реализован метод, с помощью которого строится одна из моделей ансамбля.

Листинг 15 – Метод построения модели

def build\_model(filters\_first\_layer, filters\_second\_layer, kernel\_size, dropout, dense\_units, algorithm\_enum, input\_shape):

model = models.Sequential([

layers.Conv1D(filters\_first\_layer, kernel\_size, padding='same', activation='relu', input\_shape=input\_shape),

layers.MaxPooling1D(2),

layers.Dropout(dropout),

layers.Conv1D(filters\_second\_layer, kernel\_size, padding='same', activation='relu'),

layers.MaxPooling1D(2),

layers.Flatten(),

layers.Dense(dense\_units, activation='relu'),

layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer=get\_algorithm\_enum(algorithm\_enum),

loss='binary\_crossentropy',

metrics=['accuracy', recall\_m, precision\_m, f1\_m, balanced\_accuracy\_m])

return model

Для оценки метрик Recall, Precision, F1 и BalancedAccuracy были реализованы функции получения метрик.

Листинг 16 – Функции получения метрик

from keras import backend as K

def recall\_m(y\_true, y\_pred):

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

recall = true\_positives / (possible\_positives + K.epsilon())

return recall

def precision\_m(y\_true, y\_pred):

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

return precision

def f1\_m(y\_true, y\_pred):

precision = precision\_m(y\_true, y\_pred)

recall = recall\_m(y\_true, y\_pred)

return 2\*((precision\*recall)/(precision+recall+K.epsilon()))

def balanced\_accuracy\_m(y\_true, y\_pred):

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

true\_negatives = K.sum(K.round(K.clip((1 - y\_true) \* (1 - y\_pred), 0, 1)))

possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

possible\_negatives = K.sum(K.round(K.clip(1 - y\_true, 0, 1)))

return (true\_positives / (possible\_positives + K.epsilon()) + true\_negatives / (

possible\_negatives + K.epsilon())) / 2

Был реализован метод получения алгоритма-оптимизатора компиляции модели. В качестве алгоритмов были использованы алгоритмы adam, rmsprop, sgd.

Листинг 17 – Метод получения алгоритма-оптимизатора компиляции модели

def get\_algorithm\_enum(algorithm\_enum) -> str:

if algorithm\_enum <= 1:

return 'adam'

if algorithm\_enum == 2:

return 'rmsprop'

if algorithm\_enum >= 3:

return 'sgd'

Был реализован метод получения среднего значения предсказания ансамбля.

Листинг 18 – Метод получения среднего значения предсказания ансамбля

# Среднее предсказание от всех моделей

def ensemble\_predict(models, data\_x, data\_y):

metrics = [model.evaluate(data\_x, data\_y, verbose=0) for model in models]

return np.mean(metrics, axis=0)

Был реализован метод расчёта ROC\_AUC\_SCORE, а также отрисовки графика.

Листинг 19 – Метод расчёта и отрисовки ROC\_AUC\_SCORE

import copy

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

def get\_roc\_auc\_score(models, data\_x, data\_y, is\_plotting=False):

y\_score = [i[0] for i in np.mean([model.predict(data\_x, verbose=0) for model in models], axis=0)]

y\_true = copy.deepcopy(data\_y.numpy())

fpr, tpr, thresholds = roc\_curve(y\_true, y\_score)

roc\_auc = auc(fpr, tpr)

if is\_plotting:

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")

plt.show()

return roc\_auc

# 3 Поиск гиперпараметров с помощью генетического алгоритма

Для выбранной комбинации набора данных и предиктивной модели был реализован скрипт поиска гиперпараметров с помощью генетического алгоритма. Для реализации генетического алгоритма использовалась библиотека DEAP. Размер популяции равен 5, количество популяций равно 5.

Листинг 20 – Генетический алгоритм

import random

from deap import base, creator, tools

from net import build\_ensemble

import pandas as pd

crossover\_probability = 0.5

mutation\_probability = 0.5

# Определение границ для каждой переменной

bounds = [(1, 100), (1, 9), (0, 1), (1, 3)]

csv\_path = ('C:\\Users\\Gubay\\OneDrive\\Documents\\Archive\_University\\Мага\_3\\ml\_course\_work\\datasets'

'\\kdd\_10000\_labled\_modified.csv')

data = pd.read\_csv(csv\_path)

# Создаем класс FitnessMax для определения максимизируемой функции

creator.create("FitnessMax", base.Fitness, weights=(1.0,))

# Создаем класс Individual для определения структуры хромосомы

creator.create("Individual", list, fitness=creator.FitnessMax)

# Функция для генерации случайного целого числа от low до high

def random\_int(low, high):

return random.randint(low, high)

# Функция для генерации случайного вещественного числа от low до high

def random\_float(low, high):

return random.uniform(low, high)

def control\_mutation(mutant):

if mutant[0] < 0:

mutant[0] = -mutant[0]

if mutant[0] == 0:

mutant[0] = 1

if mutant[1] < 0:

mutant[1] = -mutant[1]

if mutant[1] == 0:

mutant[1] = 1

if mutant[2] < 0:

mutant[2] = -mutant[2]

if mutant[2] == 0:

mutant[2] = 1

if mutant[3] < 0:

mutant[3] = -mutant[3]

if mutant[4] < 0:

mutant[4] = -mutant[4]

if mutant[4] == 0:

mutant[4] = 1

return mutant

# Определяем функцию оценки (fitness function), которая будет вычислять значение максимизируемой функции для каждой

# хромосомы

def evaluate\_g(individual):

# Здесь выполняется обучение модели и вычисление значения максимизируемой функции на основе выбранного набора

# гиперпараметров

fitness\_value = build\_ensemble(data, individual[0], individual[1], individual[2], individual[3], individual[4], individual[5])

return (fitness\_value,)

# Определяем размер популяции и количество поколений

population\_size = 5

generations = 5

# Создаем объект Toolbox для хранения инструментов для работы с ГА

toolbox = base.Toolbox()

# Задаем функцию для инициализации хромосомы

toolbox.register("attr\_filter", random\_int, bounds[0][0], bounds[0][1])

toolbox.register("attr\_kernel", random\_int, bounds[1][0], bounds[1][1])

toolbox.register("attr\_float", random\_float, bounds[2][0], bounds[2][1])

toolbox.register("attr\_algorithm", random\_int, bounds[3][0], bounds[3][1])

# Задаем функцию для создания индивидуума (хромосомы)

toolbox.register("individual", tools.initCycle, creator.Individual,

(toolbox.attr\_filter, toolbox.attr\_filter, toolbox.attr\_kernel, toolbox.attr\_float, toolbox.attr\_filter,

toolbox.attr\_algorithm),

n=1)

# Задаем функцию для создания популяции

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Задаем функцию для выбора родителей

toolbox.register("select", tools.selTournament, tournsize=3)

# Задаем функцию для скрещивания

toolbox.register("mate", tools.cxTwoPoint)

# Задаем функцию для мутации

toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

# Задаем функцию для оценки популяции

toolbox.register("evaluate", evaluate\_g)

# Создаем начальную популяцию

population = toolbox.population(n=population\_size)

# Выполняем генетический алгоритм

for gen in range(generations):

print('start gen#' + str(gen))

# Оцениваем популяцию

fitnesses = list(map(toolbox.evaluate, population))

# Присваиваем каждому индивидууму его значение fitness

for ind, fit in zip(population, fitnesses):

ind.fitness.values = fit

# Выбираем следующее поколение родителей

offspring = toolbox.select(population, len(population))

# Клонируем выбранных родителей

offspring = list(map(toolbox.clone, offspring))

# Применяем скрещивание и мутацию на потомстве

for child1, child2 in zip(offspring[::2], offspring[1::2]):

if random.random() < crossover\_probability:

toolbox.mate(child1, child2)

del child1.fitness.values

del child2.fitness.values

for mutant in offspring:

if random.random() < mutation\_probability:

toolbox.mutate(mutant)

mutant = control\_mutation(mutant)

del mutant.fitness.values

# Вычисляем фитнес-значения для потомства

invalid\_ind = [ind for ind in offspring if not ind.fitness.valid]

fitnesses = map(toolbox.evaluate, invalid\_ind)

for ind, fit in zip(invalid\_ind, fitnesses):

ind.fitness.values = fit

# Заменяем текущую популяцию потомством

population[:] = offspring

# Получаем лучшее решение

best\_solution = tools.selBest(population, 1)[0]

print(best\_solution)

Генетический алгоритм выдал наилучший набор гиперпараметров: filters\_first\_layer = 23, filters\_second\_layer = 83, kernel\_size = 4, dropout = 0.5265110647958439, dense\_units = 57, algorithm\_enum = adam.

Метрики:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0

# 4 Поиск гиперпараметров с помощью метода GreedSearch

Был реализован метод GreedSearch для поиска гиперпараметров.

Листинг 21 – Метод GreedSearch для поиска гиперпараметров

import pandas as pd

from net import build\_ensemble

csv\_path = ('C:\\Users\\Gubay\\OneDrive\\Documents\\Archive\_University\\Мага\_3\\ml\_course\_work\\datasets'

'\\kdd\_10000\_labled\_modified.csv')

data = pd.read\_csv(csv\_path)

# Определяем набор гиперпараметров для поиска по сетке

param\_grid = {

'filters\_first\_layer': [56, 76],

'filters\_second\_layer': [63, 93],

'kernel\_size': [1, 3],

'dropout': [0.3, 0.6],

'relu\_count': [64, 128],

'algorithm\_enum': [1, 3]

}

# Проводим поиск по сетке

best\_accuracy = 0

best\_params = None

for alg\_enum in param\_grid['algorithm\_enum']:

for filters1 in param\_grid['filters\_first\_layer']:

for filters2 in param\_grid['filters\_second\_layer']:

for kernel in param\_grid['kernel\_size']:

for drop in param\_grid['dropout']:

for r\_count in param\_grid['relu\_count']:

print(f"\nTrying: {filters1}, {filters2}, {kernel}, {drop}, {r\_count}, {alg\_enum}")

# Запуск обучения модели для каждой комбинации гиперпараметров

accuracy = build\_ensemble(data, filters1, filters2, kernel, drop, r\_count, alg\_enum)

# Сравнение результатов и сохранение наилучшей комбинации

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_params = {'filters\_first\_layer': filters1,

'filters\_second\_layer': filters2,

'kernel\_size': kernel,

'dropout': drop,

'relu\_count': r\_count,

'algorithm\_enum': alg\_enum}

print("Best BalancedAccuracy:", best\_accuracy)

print("Best Parameters:", best\_params)

GreedSearch выдал наилучший набор гиперпараметров: filters\_first\_layer = 56, filters\_second\_layer = 63, kernel\_size = 1, dropout = 0.3, dense\_units = 64, algorithm\_enum = adam.

Метрики:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0

Best BalancedAccuracy: 1.0

# 5 Обучение предиктивной модели с помощью автокодировщика

Были реализованы методы предобучения с помощью автокодировщика и точной настройки. Гиперпараметры для построения базовой нейронной сети были взяты из результата генетического алгоритма.

Листинг 22 – Методы предобучения с помощью автокодировщика и точной настройки

import random

import tensorflow as tf

import pandas as pd

from keras import layers, models

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

from sklearn.preprocessing import StandardScaler

from net import build\_model, get\_algorithm\_enum, ensemble\_predict

from roc\_score import get\_roc\_auc\_score

def build\_autoencoder(cnn\_model, filters\_first\_layer, filters\_second\_layer, kernel\_size, dense\_units, algorithm\_enum, input\_shape):

encoder = models.Sequential(cnn\_model.layers[:-1])

decoder = models.Sequential([

layers.Dense(dense\_units, activation='relu', input\_shape=(dense\_units,)), # Mirror the Dense layer

layers.Dense((input\_shape[0] // 4) \* filters\_second\_layer, activation='relu'), # Mirror the Flatten layer

layers.Reshape((input\_shape[0] // 4, filters\_second\_layer)), # Reshape to mirror the Flatten layer

layers.UpSampling1D(2), # Mirror the MaxPooling1D layer

layers.Conv1DTranspose(filters\_second\_layer, kernel\_size, padding='same', activation='relu'), # Mirror the Conv1D layer

layers.UpSampling1D(2), # Mirror the MaxPooling1D layer

layers.Conv1DTranspose(filters\_first\_layer, kernel\_size, padding='same', activation='relu'), # Mirror the Conv1D layer

layers.Conv1D(1, kernel\_size, padding='same', activation='sigmoid')

])

autoencoder = models.Sequential([

encoder,

decoder

])

autoencoder.compile(optimizer=get\_algorithm\_enum(algorithm\_enum), loss='mae')

return autoencoder

def build\_ensemble(cnn, autoencoder, test\_size):

csv\_path = ('C:\\Users\\Gubay\\OneDrive\\Documents\\Archive\_University\\Мага\_3\\ml\_course\_work\\datasets'

'\\kdd\_10000\_labled\_modified.csv')

dataset = pd.read\_csv(csv\_path)

data\_x = dataset.iloc[:, 0:dataset.shape[1] - 1].values

data\_y = dataset.iloc[:, dataset.shape[1] - 1].values

scaler = StandardScaler()

df\_x\_st\_scaled = scaler.fit\_transform(data\_x)

data\_x = df\_x\_st\_scaled

# Присвоить средние значения весов слою другой модели

for i in range(len(cnn.layers) - 1):

layer\_weights = autoencoder.layers[0].layers[i].get\_weights()

cnn.layers[i].set\_weights(layer\_weights)

num\_cnn = 5

models\_list = []

for \_ in range(num\_cnn):

seed = random.randint(1, 100)

df\_x\_train, df\_x\_test, df\_y\_train, df\_y\_test = (

train\_test\_split(data\_x, data\_y, test\_size=test\_size, random\_state=seed))

df\_x\_train = tf.cast(df\_x\_train, dtype=tf.float32)

df\_x\_test = tf.cast(df\_x\_test, dtype=tf.float32)

df\_y\_train = tf.cast(df\_y\_train, dtype=tf.float32)

df\_y\_test = tf.cast(df\_y\_test, dtype=tf.float32)

cnn.fit(df\_x\_train, df\_y\_train, epochs=5, batch\_size=32, verbose=1)

models\_list.append(cnn)

# Оценка ансамбля на тестовых данных

metrics\_data = ensemble\_predict(models\_list, df\_x\_test, df\_y\_test)

ensemble\_accuracy = metrics\_data[1]

ensemble\_f1 = metrics\_data[2]

ensemble\_precision = metrics\_data[3]

ensemble\_recall = metrics\_data[4]

ensemble\_balanced\_accuracy\_score = metrics\_data[5]

roc\_auc\_score = get\_roc\_auc\_score(models\_list, df\_x\_test, df\_y\_test, True)

print('Ensemble Accuracy:', ensemble\_accuracy)

print('Ensemble Recall:', ensemble\_recall)

print('Ensemble Precision:', ensemble\_precision)

print('Ensemble F1:', ensemble\_f1)

print('Ensemble ensemble\_balanced\_accuracy\_score:', ensemble\_balanced\_accuracy\_score)

print('Ensemble ROC\_AUC\_SCORE:', roc\_auc\_score)

return ensemble\_balanced\_accuracy\_score

def build\_ensemble\_by\_autoencoder(dataset, filters\_first\_layer, filters\_second\_layer, kernel\_size, dropout, dense\_units,

algorithm\_enum):

test\_size = 0.2

data\_x = dataset.iloc[:, 0:dataset.shape[1]].values

input\_shape = (data\_x.shape[1], 1)

# Сохранил имена столбцов и последний столбец

scaler = StandardScaler()

df\_st\_scaled = scaler.fit\_transform(data\_x)

data\_x = df\_st\_scaled

print('start create model')

cnn = build\_model(filters\_first\_layer, filters\_second\_layer, kernel\_size, dropout, dense\_units, algorithm\_enum, input\_shape)

print('end create model\n')

df\_x\_train, df\_x\_test = train\_test\_split(data\_x, test\_size=test\_size)

df\_x\_train = tf.cast(df\_x\_train, dtype=tf.float32)

df\_x\_test = tf.cast(df\_x\_test, dtype=tf.float32)

autoencoder = build\_autoencoder(cnn, filters\_first\_layer, filters\_second\_layer, kernel\_size, dense\_units, algorithm\_enum, input\_shape)

autoencoder.fit(df\_x\_train, df\_x\_train, epochs=25, batch\_size=512, verbose=1, validation\_data=(df\_x\_test, df\_x\_test))

autoencoder.summary()

res = build\_ensemble(cnn, autoencoder, test\_size)

print(res)

Полученная модель автокодировщика:

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

sequential\_1 (Sequential) (None, 57) 17353

sequential\_2 (Sequential) (None, 8, 1) 48325

=================================================================

Total params: 65678 (256.55 KB)

Trainable params: 65678 (256.55 KB)

Non-trainable params: 0 (0.00 Byte)

Результат обучения:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0

# Вывод

Было разработано приложения на языке Python и необходимых библиотек машинного и глубокого обучения на основе классификационной модели ансамбль нейронных сетей типа бэггинг на основе сверточной нейросетевой модели с полносвязным классификатором, которое определяет попытки сетевых вторжений на основе данных сетевого трафика.

Был реализован генетический алгоритм поиска гиперпараметров предиктивной модели. Генетический алгоритм выдал наилучший набор гиперпараметров: filters\_first\_layer = 23, filters\_second\_layer = 83, kernel\_size = 4, dropout = 0.5265110647958439, dense\_units = 57, algorithm\_enum = adam.

Метрики:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0

Также был реализован метод GreedSearch для поиска гиперпараметров предиктивно модели. GreedSearch выдал наилучший набор гиперпараметров: filters\_first\_layer = 56, filters\_second\_layer = 63, kernel\_size = 1, dropout = 0.3, dense\_units = 64, algorithm\_enum = adam.

Метрики:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0

Best BalancedAccuracy: 1.0

Невозможно окончательно сказать какой метод поиска гиперпараметров лучше, так как они оба показали хороший результат.

Были реализованы методы предобучения с помощью автокодировщика и точной настройки. Гиперпараметры для построения базовой нейронной сети были взяты из результата генетического алгоритма. Потери автокодировщика на первой эпохе составляли 0.7304, а на последней 0.6246. Итоговые метрики после пред обучения и точной настройке:

Ensemble Accuracy: 1.0

Ensemble Recall: 1.0

Ensemble Precision: 1.0

Ensemble F1: 1.0

Ensemble ensemble\_balanced\_accuracy\_score: 1.0

Ensemble ROC\_AUC\_SCORE: 1.0