



LABORATORY WORK 3
for the course
SECURITY OF MOBILE TECHNOLOGY

Name : Smagulov Iliyas

ID : 30481

Group : SIS-2111

Содержание

1. Введение в лабораторную работу
2. Сопутствующая работа
3. Алгоритмы глубокого обучения
4. Реализация
5. Итог проведенной работы
6. Литература

1. Введение в лабораторную работу

В современном мире технологии стали неизбежной частью человеческой жизни. Количество различных типов терминальных устройств растет с каждым годом. На большом количестве терминальных устройств IoT отсутствуют средства защиты, такие как брандмауэры, поэтому эти устройства чрезвычайно уязвимы для управления вредоносным кодом. Более того, в новой ситуации традиционными методами защиты от сетевых атак с высокой степенью скрытности сложно идентифицировать их и защититься от них. Распределенные атаки типа "Отказ в обслуживании" (DDoS) проводятся на другие важные устройства IoT путем захвата контроля над вредоносным устройством IoT. В феврале 2022 года Украина подверглась массовой DDoS-атаке, которая привела к закрытию нескольких военных веб-сайтов, включая сайты Министерства обороны и Вооруженных сил Украины. Частота инцидентов с безопасностью IoT дала понять, что существует необходимость в усилении защиты power IoT security.

DDoS-атаки создают огромные объемы атакующего трафика, формируя массивные ботнеты, которые загружают серверы и сетевые соединения, таким образом, потребляя ресурсы серверов таким образом, что они не могут соответствующим образом реагировать на запросы об обслуживании согласно полученным данным, большинство методов обнаружения DDoS-атак обычно выполняют обнаружение атак на серверах или облачных центрах. Однако серверная часть должна не только обнаруживать DDoS-атаки, но и обрабатывать запросы от всех сторон, поэтому возникают такие ситуации, как отсутствие контекстов сетевого подключения. Это позволяет серверным методам обнаружения DDoS-атак обрабатывать и анализировать только неполные пакеты сетевого трафика, что, в свою очередь, приводит к тому, что аномальный трафик при DDoS-атаке не сразу можно распознать. Чтобы решить вышеуказанную проблему, необходимо перенаправить весь сетевой трафик в одном и том же цикле связи в свободный сегмент сети для необходимой очистки трафика, но этот процесс увеличит нагрузку на уже заблокированную сеть, что приведет к невозможности реагировать в режиме реального времени на запросы пользователей, которые влияют на работу пользователя. Более того, большинство современных методов обнаружения используют алгоритмы машинного обучения, однако в алгоритмах машинного обучения слишком большое значение придается выбору признаков и

обучению параметрам. Методы глубокого обучения могут эффективно решать проблему классификации огромных объемов данных в реальных средах веб-приложений. Использование методов глубокого обучения значительно улучшило применение традиционных методов машинного обучения для обнаружения атак. Компьютерная часть сети осложняется DDoS-атаками на серверы, в основном с персональных компьютеров, которые являются промежуточными отправными точками. Однако для устройств power IoT существуют различные типы устройств power IoT с особыми характеристиками трафика. Чтобы устранить вышеуказанные ограничения, предлагается метод обнаружения DDoS-атак в power IoT, основанный на пограничных вычислениях и двунаправленной долговременной памяти (BiLSTM). В модели используется концепция пограничных вычислений для разработки распределенного метода обнаружения, основанного на нейронных сетях BiLSTM. В частности, сетевые сервисы, генерируемые управляемыми ими устройствами Интернета вещей, обнаруживаются пограничными узлами с использованием модели обнаружения DDoS-атак. Мы характеризуем характеристики сетевого потока, определяя статистические характеристики IP-пакетов, также разрабатываем модель прогнозирования сетевого трафика на основе BiLSTM, которая способна изучать и определять взаимосвязь между пакетами одного и того же потока данных в целом.

2. Сопутствующая работа

В прошлом было предпринято множество исследований, направленных на снижение угрозы DDoS-атак. Некоторые из этих исследований и попыток являются подлинными, в то время как другие устарели. Эти характеристики соответствуют природе современных DDoS-ботнетов. Существует множество различных типов вредоносных программ для DDoS-атак, некоторые из которых регулярно развиваются, что делает их практически невозможными для обнаружения. Некоторые вредоносные программы для DDoS-атак могут переходить в спящий режим на взломанных устройствах, не создавая проблем, создавая впечатление, что ничего необычного не происходит, пока хакер не запустит атаку. Благодаря глубокому обучению были разработаны новые методы и стратегии борьбы с DDoS-атаками и другими проблемами..

Исследователями была предложена программно-определяемая сетевая классификация с использованием трех моделей, а именно,

закрытого RNN “в качестве основной модели”, и проведено сравнение с VanillaRNN, методом опорных векторов (SVM) и глубокой нейронной сетью (DNN) для набора данных NSL-KDD, и сравнил предложенную модель с DNN, SVM и наивными байесовскими деревьями (NB), используя набор данных для оценки обнаружения вторжений (CICIDS2017). Это исследование вводит смягчающий фактор или фильтр; фильтр должен принимать трафик до того, как он достигнет программно-определяемого сетевого коммутатора (SDN), в качестве SDN-контроллера, состоящего из трех наиболее важных частей: сборщика потока, детектора аномалий и средства устранения аномалий. С помощью интерфейса gated recurrent units-RNN (GRU-RNN) была достигнута точность 0,89 при использовании набора данных NSL-KDD и 0,99 при использовании CICIDS2017.

Исследование реализовали две модели глубокого обучения в наборе данных CICDDoS2019. Это модели DNN и LSTM, которые предоставляют более точные данные за счет использования функции выбора, оставляя для них три метки, отличающие данные от “UDP, SYN и UDP-lag”. Каждая из них содержит набор данных с двумя метками “стандарт” и типом атаки. По-видимому, средняя точность, полученная с помощью обеих моделей, была в некоторой степени сопоставима, поскольку она составила 0,999 при выполнении серии из 32 и 10 периодов для всех экспериментов. В исследовании были проведены эксперименты только с LSTM в наборе данных CICIDS2017. Однако основная цель состоит в том, чтобы определить, какая настройка является наилучшей для получения максимальной точности. Они попробовали разные гиперпараметры, такие как функция активации, функция потерь, оптимизатор и т.д., и различное количество слоев LSTM. В их эксперименте использовалось от одного до шести слоев, сокращенно обозначаемых как L1–L6. Как видно из результатов этого исследования, оптимальная настройка гиперпараметра, при которой достигается наибольшая точность, приведена в таблице 1.

Таблица 1.

Parameter	Name of parameter	Accuracy obtained
LSTM layers	L5	0.9909
Optimizer	RMSprop	0.9908
Activation function	Tanh	0.9908
Loss function	Kullback-Leibler divergence	0.9887

В исследовании представлен алгоритм глубокого обучения, в котором они объединили RNN с CNN в качестве гибридной системы обнаружения. Использовался набор данных realistic cyber defense dataset (CSE-CIC-IDS2018) и еще три алгоритма, а именно логистическая регрессия, экстремальное повышение градиента и Дерево решений, для сравнения с недавно представленной гибридной сверточной рекуррентной системой обнаружения сетевых вторжений на основе нейронных сетей, на основе их производительности. Все модели имеют одинаковую структуру здания с десятикратной перекрестной проверкой. Их комбинация превзошла другие модели, что дало точность 0,97 по сравнению с другими моделями, самой высокой из которых было дерево решений (DT) с точностью 0,88. Также использовался набор алгоритмов глубокого и машинного обучения для обнаружения аномалий в устройствах IoT с использованием набора данных CICIDS2017. В качестве моделей используются глубокие сверточные нейронные сети (DCNN), многослойный персептрон (MLP), LSTM, CNN + LSTM, SVM, NB и случайный лес (RF). В таблице 2 кратко представлены модели глубокого обучения и соответствующие им результаты в соответствующей работе.

Таблица 2.

Ref.	Dataset	Models	Accuracy	Precision	Recall	F1-score
[14]	NSL-KDD	VanilaRNN	0.4439	0.9100	0.9000	0.9000
		SVM	0.6567			
		DNN	0.7590			
		GRU-RNN	0.8900			
	CICIDS2017	SVM	0.6952	—	—	—
		DNN	0.7575	0.9700	0.9700	0.9700
		NB trees	0.8202	—	—	—
[15]	CICDDoS2019	Syn	GRU-RNN	0.9900	0.9900	0.9900
			DNN	0.9995	0.9998	0.9997
			Upd	0.9995	0.9995	1.000
			Upd-lag	0.9993	0.9994	0.9997
		LSTM	Syn	0.9997	0.9997	0.9998
			Upd	0.9996	0.9996	1.0000
			Upd-lag	0.9990	0.9989	1.0000
[16]	CICIDS2017	LSTM	0.9954	—	0.9340	0.9475
[17]	CSE-CIC-DS2018	HCRNN	0.9700	0.9633	0.9712	0.9760
		Linear regression	0.8000	0.7810	0.8010	0.7910
		DT	0.8800	0.8733	0.8850	0.8790
		XGBoost	0.8900	0.8450	0.8340	0.8390
[18]	CICIDS2017	DCNN	0.9514	0.9814	0.9017	—
		MLP	0.8643	0.8847	0.8625	—
		LSTM	0.9624	0.9844	0.8989	—
		CNN + LSTM	0.9716	0.9741	0.9910	—
		SVM	0.9550	0.9772	0.9912	—
		NB	0.9519	0.9256	0.9284	—
		RF	0.9464	0.9018	0.9089	—

Недавние попытки разработать интеллектуальные модели для защиты сетей Интернета вещей от DDoS-атак увенчались заметным успехом. Однако разработка модели, способной защитить сеть от различных типов DDoS-атак, не затрагивая законный трафик, представляет собой основной пробел в исследованиях. Кроме того,

тестирование этих моделей в реальных условиях для таких типов является сложной задачей. Как показано в таблице 2, для тестирования моделей защиты от DDoS-атак подготовлены различные наборы данных, такие как NSL-KDD, CICIDS2017, CSE-CIC-DS2018 и CICDDoS2019. Предлагаемые модели защиты используют машинное обучение и алгоритмы глубокого обучения. Результаты показывают, что, в основном, методы, основанные на глубоком обучении, обеспечивают лучшую производительность, чем методы, основанные на машинном обучении, особенно для хорошо известных атак, например, LSTM достигает точности 99,96% в наборе данных CICDDoS2019, а CNN + LSTM достигает точности 97,16% в наборе данных CICIDS2017. Тем не менее, новые типы атак представляют собой серьезную проблему для обоих подходов.

3. Алгоритмы глубокого обучения

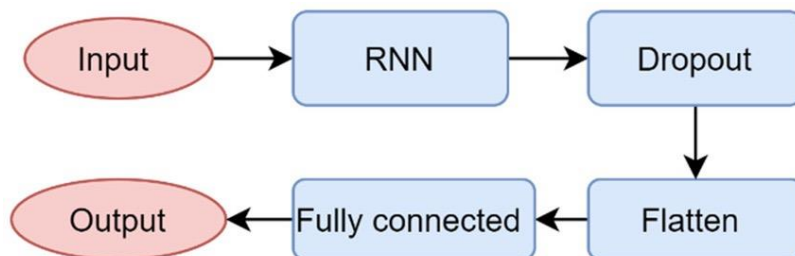
Структура каждой модели практически одинакова, при необходимости в нее вносятся некоторые незначительные изменения. В таблице 3 приведены общие параметры.

Таблица 3.

Batch size	Input shape	Learning rate	Epsilon	Optimizer	Epochs	Loss function	Verbose	Activation
1,024	1, 76	0.001	1×10^{-7}	RMSprop	100	Sparse categorical cross-entropy	1	Sigmoid, Relu

RNN: RNN - это нейронная сеть, в которой выходные данные предыдущих этапов используются в качестве входных данных. В типичной нейронной сети все входы и выходы независимы. Тем не менее, предварительные слова требуются для таких задач, как предсказание следующего выражения фразы, что приводит к повторению последних комментариев. Затем проблема была решена с помощью RNN через скрытый слой [20]. Базовая структура RNN, использованная в данном исследовании, показана на рисунке 1.

Рисунок 1.



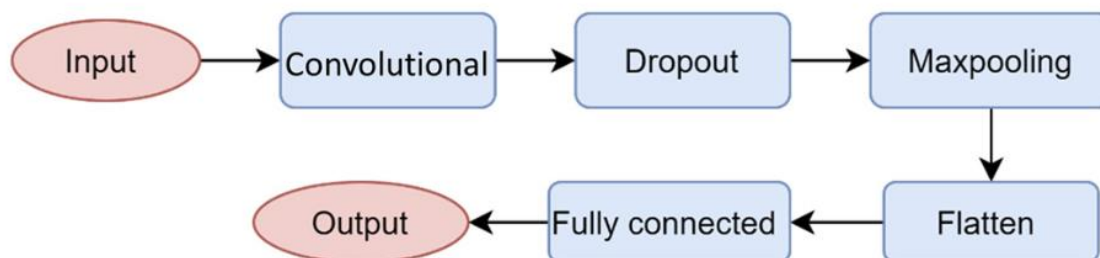
Скрытое состояние, в котором хранятся данные о конкретной последовательности, является основным и существенным компонентом RNN. В этом исследовании мы использовали простую RNN. Таким образом, даже эта очень простая конструкция надежна с символической точки зрения. Однако на практике репрезентативность - не единственная проблема. Важно подчеркнуть, что эти впечатляющие результаты по репрезентации никоим образом не означают, что мы можем извлечь такие изображения из данных за приемлемый период времени. Результаты, полученные с помощью модели, могут в значительной степени подтвердить это утверждение. RNN использует скрытые слои и наилучшим образом представляется следующим уравнением:

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h), t=1, 2, 3, \dots, T,$$

где h_t - скрытое состояние, t - временной шаг, σ - функция активации скрытого слоя, x - входные данные, W - вес, а b - смещение.

CNN: CNN основаны на нейронных сетях, которые построены на основе нейронных сетей. Структура слоев CNN: сверточный, объединяющий и полностью связанный. Этот слой представляет собой просто нейронную сеть, о которой говорилось ранее. CNN также включает в себя два важных компонента: функцию активации и уровень отсева. Алгоритм CNN выполняет два основных процесса на уровнях свертки и максимального объединения: свертку и выборку. На рисунке 2 показана базовая архитектура CNN.

Рисунок 2.



Каждый нейрон получает информацию от ядер $x_{i,j}$, входящих в состав предыдущего слоя, известного как локальное рецептивное поле .

$$x_{i,j}^{(l)} = \sigma \left(b + \sum_{r=0}^n \sum_{c=0}^n w_r, c^{x_{i+r,j+c}^{(j-1)}} \right),$$

где $x_{i,j}$ - текущий выходной сигнал, σ - функция активации скрытого слоя, c - входной сигнал, w - вес, а b - смещение.

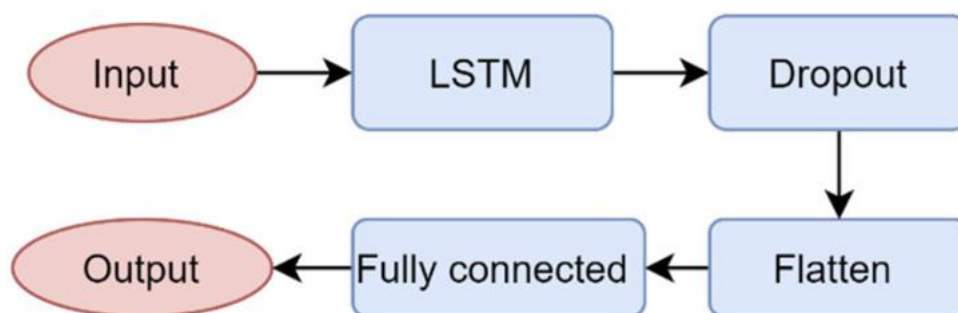
LSTM: Хохрайтер и Шмидхубер были первыми, кто предложил LSTM как особый тип RNN. Стандартная архитектура LSTM имеет входной уровень, повторяющийся уровень LSTM и выходной уровень. Входной уровень связан со слоем LSTM. Формула представляет функцию принятия решений первого уровня LSTM-RNN f_t . Решение связано с использованием или удалением предоставленных данных, h_{t-1} , из обработанного вывода предыдущей ячейки следующим образом:

$$f_t = \sigma(w_f \times [h_{t-1}, x_t] + b_f),$$

где h_{t-1} - данные, полученные в результате предыдущего процесса, t - шаг по времени, σ - сигмовидная функция скрытого слоя, x - входные данные, w - вес, а b - смещение.

Повторяющиеся соединения LSTM напрямую подключаются от блоков вывода ячейки к блокам ввода ячейки, входным элементам, выходным элементам и вентилям. Блоки вывода ячейки также подключены к сетевому уровню вывода. По следующей формуле можно рассчитать общее количество N параметров в типичной сети LSTM с одной ячейкой в каждом блоке памяти (без учета искажений). На рисунке 6 показаны выполненные слои модели LSTM.

Рисунок 3.



Двунаправленный LSTM или BiLSTM - это термин, используемый для модели последовательности, которая содержит два уровня LSTM, один из которых предназначен для обработки входных данных в прямом направлении, а другой - для обработки в обратном направлении. Обычно он используется в задачах, связанных с нейролингвистическое программированием. Суть этого подхода заключается в том, что, обрабатывая данные в обоих направлениях, модель может лучше понять взаимосвязь между последовательностями (например, зная следующее и предшествующее слова в предложении).

Чтобы лучше понять это, давайте рассмотрим пример. Первое утверждение - "Server can you bring me this dish", а второе - "He crashed the server". В обоих этих утверждениях слово "server" имеет разные значения, и это соотношение зависит от следующего и предыдущего слов в утверждении. Двунаправленный LSTM помогает машине лучше понимать эту взаимосвязь, чем однонаправленный LSTM. Эта способность BiLSTM делает его подходящей архитектурой для таких задач, как анализ тональности, классификация текста и машинный перевод.

Архитектура двунаправленного LSTM состоит из двух однонаправленных Lstm, которые обрабатывают последовательность как в прямом, так и в

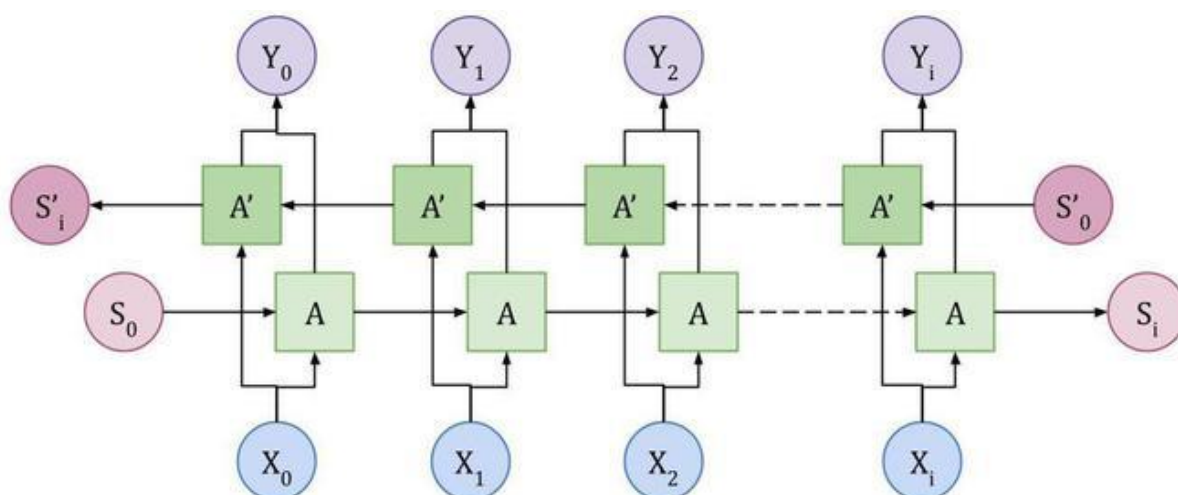
обратном направлении. Эту архитектуру можно интерпретировать как наличие двух отдельных сетей LSTM, одна из которых получает последовательность токенов в том виде, в каком она есть, а другая - в обратном порядке. Обе эти сети LSTM возвращают вектор вероятности в качестве выходных данных, а конечный результат представляет собой комбинацию обеих этих вероятностей. Это может быть представлено в виде:

$$p_t = p_t^f + p_t^b$$

где,

- p_t : Конечный вектор вероятности сети.
- p_t^f : Вектор вероятности из прямой сети LSTM.
- p_t^b : Вектор вероятности из обратной сети LSTM.

Рисунок 4.



На рисунке 4 описана архитектура уровня BiLSTM, где

X_i - входной токен,

Y_i - выходной токен, а

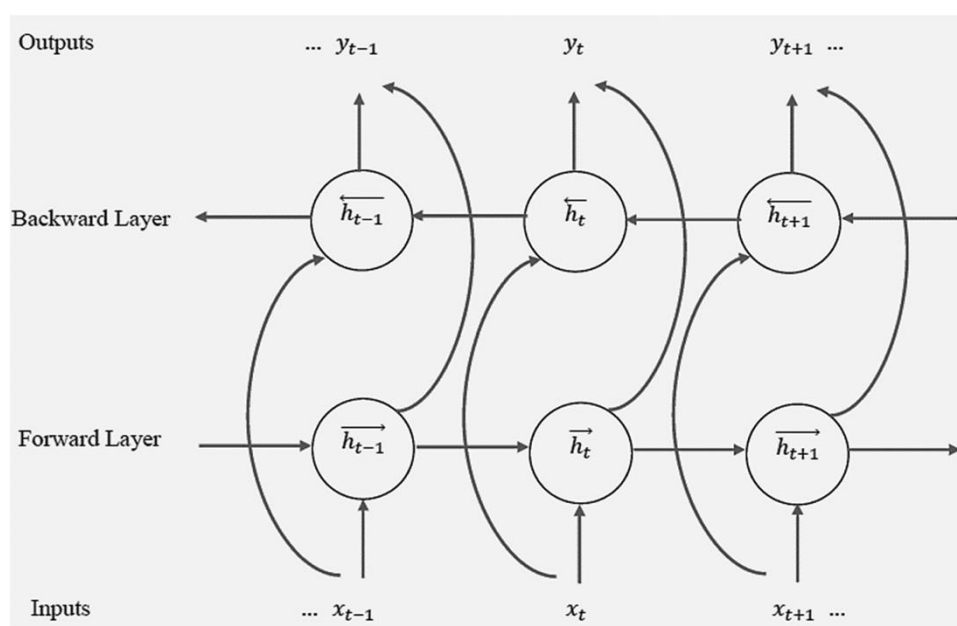
A и A' - узлы LSTM. Конечный результат

Y_i - это комбинация

A и A' LSTM-узлы.

CNN-BiLSTM: В этой модели используется два мощных алгоритма RNN и CNN и объединили их в CNN-BiLSTM. Изначально было предложено убрать ограничения RNN. Идея состоит в том, чтобы разделить нейроны состояний обычной RNN на две части: одну для положительного времени (переднюю) и одну для отрицательного времени (обратную) (обратные состояния). Выходные сигналы внешнего состояния не связаны с входами обратного состояния, и наоборот. В результате получается общая структура BiLSTM, показанная на рисунке 5, которая состоит из трех этапов.

Рисунок 5.

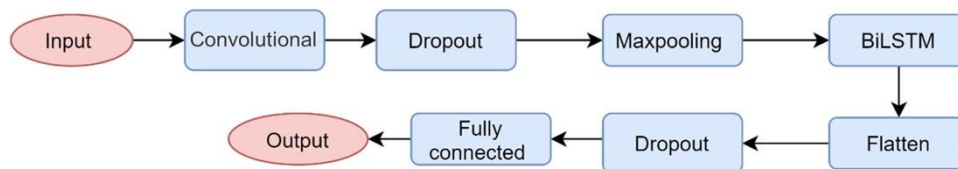


Представленная модель CNN-BiLSTM представлена на рисунке 8.

Сверточный слой создает объекты на основе входных данных. Слой отсева случайным образом отсеивает узлы во время обучения. Этот слой

помогает регулировать процесс обучения, уменьшать ошибки обобщения и устранять причины переобучения. Тогда как по CNN

Рисунок 6.



применяется максимальное объединение слоев для уменьшения пространственного разрешения объектов. Добавляет CNN для извлечения критических шаблонов и создания высококачественных объектов. Затем BiLSTM работает в прямом и обратном направлениях и может обнаруживать обратные шаблоны.

Теперь рассмотрим реализацию представленной модели CNN-BiLSTM с использованием слоев BiLSTM на Python.

4. Реализация

Импорт библиотек и наборов данных

Библиотеки Python позволяют нам легко обрабатывать данные и выполнять типичные и сложные задачи с помощью одной строки кода.

pandas: Библиотека для работы с данными в Python. Предоставляет удобные структуры данных и функции для манипуляции и анализа табличных данных.

numpy: Фундаментальная библиотека для численных вычислений в Python. Предоставляет поддержку для работы с многомерными массивами и матрицами, а также набор математических функций для эффективной работы с этими массивами.

pytorch-tabnet: Модель машинного обучения для работы с табличными данными, основанная на механизме внимания. Реализация этой модели на базе библиотеки PyTorch.

```
import pandas as pd
import numpy as np
!pip install pytorch-tabnet
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

```
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics
```

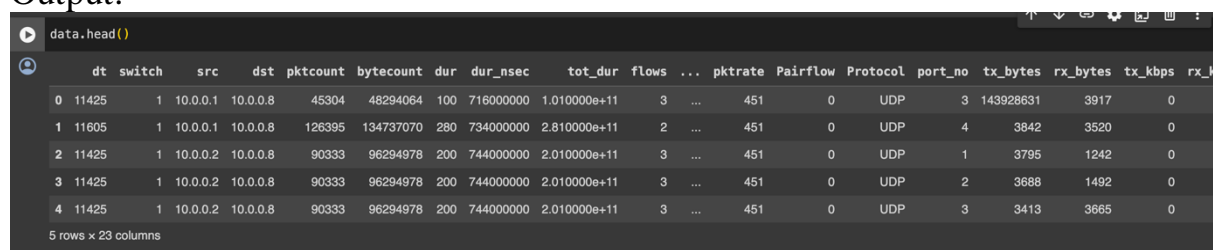
```
data = pd.read_csv('../input/ddos-sdn-dataset/dataset_sdn.csv')
```

Эта функция из библиотеки pandas считывает данные из файла CSV и загружает их в объект DataFrame, который представляет собой двумерную структуру данных с метками.

Анализ данных

```
data.head()
```

Output:



	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3917	0	
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3520	0	
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1242	0	
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	1492	0	
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	3665	0	

5 rows x 23 columns

Столбцы набора данных

dt: Временная метка, указывающая дату и время зарегистрированного события.

switch: Идентификатор сетевого коммутатора, участвующего в передаче данных.

src: IP-адрес источника сетевого трафика.

dst: IP-адрес назначения сетевого трафика.

pktscount: Количество пакетов, переданных в процессе обмена данными.

bytescount: Общее количество байтов, переданных в процессе обмена данными.

dur: Продолжительность обмена данными в секундах.

dur_nsec: Продолжительность обмена данными в наносекундах.

tot_dur: Общая продолжительность обмена данными.

потоки: количество потоков, связанных с обменом данными.

packetins: Количество полученных пакетов.

pktpsflow: Среднее количество пакетов в потоке.

bytespsflow: Среднее количество байт в потоке.

pktrate: Скорость передачи пакетов, указывающая скорость передачи пакетов.

Pairflow: Идентификатор парного потока.

Протокол: сетевой протокол, используемый при передаче данных (например, TCP, UDP).

port_no: номер порта, связанного с передачей данных.

tx_bytes: Общее количество переданных байт.

rx_bytes: Общее количество полученных байт.

tx_kbps: Скорость передачи в килобитах в секунду.

rx_kbps: Скорость приема в килобитах в секунду.

tot_kbps: Общая скорость в килобитах в секунду.

label: метка, указывающая, связано ли сообщение с атакой DDoS (распределенный отказ в обслуживании) или нет.

```
data.shape
```

Этот код позволяет узнать размеры загруженного набора данных. Конкретно метод `shape` возвращает кортеж, содержащий количество строк и столбцов в DataFrame.

Output:

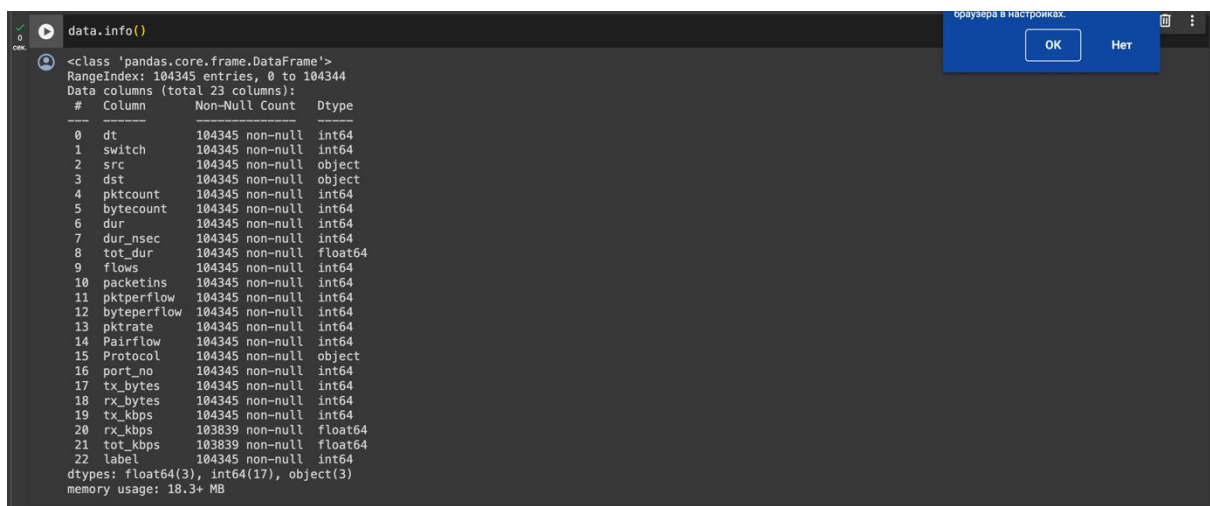
A screenshot of a Jupyter Notebook cell. The input code is `data.shape`. The output is a tuple `(104345, 23)`.

```
data.shape
(104345, 23)
```

```
data.info()
```

Этот код использует метод `info()` для вывода краткой информации о загруженном наборе данных.

Output:

A screenshot of a Jupyter Notebook cell. The input code is `data.info()`. The output shows the DataFrame's structure, including the number of entries (104345), the number of columns (23), and the data types for each column. It also shows the memory usage (18.3+ MB).

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
#   Column              Non-Null Count  Dtype
---  -
0   dt                   104345 non-null int64
1   switch               104345 non-null int64
2   src                  104345 non-null object
3   dst                  104345 non-null object
4   pktcount             104345 non-null int64
5   bytecount            104345 non-null int64
6   dur                  104345 non-null int64
7   dur_nsec             104345 non-null int64
8   tot_dur              104345 non-null float64
9   flows                104345 non-null int64
10  packetins             104345 non-null int64
11  pktperflow            104345 non-null int64
12  byteperflow           104345 non-null int64
13  pktrate               104345 non-null int64
14  Pairflow              104345 non-null int64
15  Protocol              104345 non-null object
16  port_no               104345 non-null int64
17  tx_bytes              104345 non-null int64
18  rx_bytes              104345 non-null int64
19  tx_kbps               104345 non-null int64
20  rx_kbps               103839 non-null float64
21  tot_kbps              103839 non-null float64
22  label                 104345 non-null int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

```
data.label.unique()
```

Этот код выполняет операцию `unique()` на столбце `label` в наборе данных `data`. Метод `unique()` возвращает уникальные значения из указанного столбца.

Output:

A screenshot of a Jupyter Notebook cell. The input code is `data.label.unique()`. The output is an array containing the values 0 and 1.

```
data.label.unique()
array([0, 1])
```



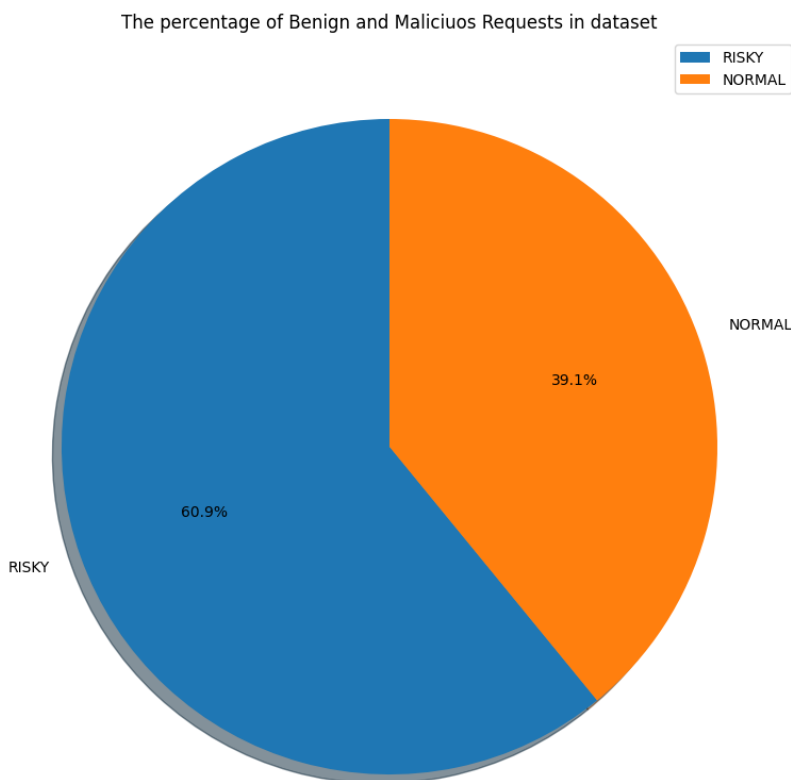
```

labels = ["RISKY", 'NORMAL']
sizes = [dict(data.label.value_counts())[0], dict(data.label.value_counts())[1]]
plt.figure(figsize = (20,10))
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.legend(["RISKY", "NORMAL"])
plt.title('The percentage of Benign and Maliciuos Requests in dataset')
plt.show()

```

Этот код использует библиотеку Matplotlib для создания круговой диаграммы, отображающей процентное соотношение меток классов в наборе данных. Этот код помогает визуализировать баланс классов в наборе данных.

Output:



```
data.describe()
```

Метод `'describe()'` возвращает основные статистические характеристики числовых столбцов в `DataFrame`, такие как количество, среднее значение, стандартное отклонение, минимальное и максимальные значения, а также квантили.

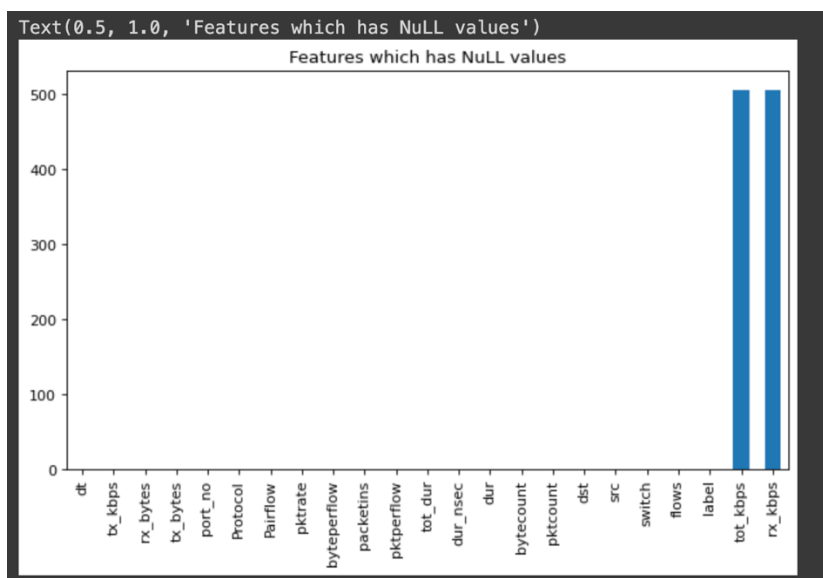
Output:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	pktrate
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000	104345.000000	1.043450e+05	104345.000000	104345.000000
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234	5200.383468	6381.715291	4.716150e+06	212.2106
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036	5257.001450	7404.777808	7.560116e+06	246.8851
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000	4.000000	-130933.000000	-1.464426e+08	-4385.0000
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000	1943.000000	29.000000	2.842000e+03	0.0000
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000	3024.000000	8305.000000	5.521680e+05	276.0000
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000	7462.000000	10017.000000	9.728112e+06	333.0000
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000	25224.000000	19190.000000	1.495387e+07	639.0000

```
# Let's look at the visualisation of Null valued features
figure(figsize=(9, 5), dpi=80)
data[data.columns[data.isna().sum() >= 0]].isna().sum().sort_values().plot.bar()
plt.title("Features which has NuLL values")
```

Этот код создает столбчатую диаграмму, отображающую количество пропущенных значений (NaN) для каждого признака в наборе данных.

Output:



```
#### Let's support which columns NUMERIC and which is OBJECT
```

```

numeric_df = data.select_dtypes(include=['int64', 'float64'])
object_df = data.select_dtypes(include=['object'])
numeric_cols = numeric_df.columns
object_cols = object_df.columns
print('Numeric Columns: ')
print(numeric_cols, '\n')
print('Object Columns: ')
print(object_cols, '\n')
print('Number of Numeric Features: ', len(numeric_cols))
print('Number of Object Features: ', len(object_cols))

```

Этот код помогает разделить столбцы набора данных на числовые и категориальные, а также выводит информацию о количестве столбцов каждого типа.

Output:

```

Numeric Columns:
Index(['dt', 'switch', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur',
      'flows', 'packetins', 'pktperflow', 'byteperflow', 'pktrate',
      'Pairflow', 'port_no', 'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps',
      'tot_kbps', 'label'],
      dtype='object')

Object Columns:
Index(['src', 'dst', 'Protocol'], dtype='object')

Number of Numeric Features:  20
Number of Object Features:   3

```

```
object_df.head()
```

Этот код выводит первые несколько строк из DataFrame 'object_df', который содержит только столбцы типа данных object. Обычно этот тип данных используется для текстовых или категориальных переменных.

Output:

	src	dst	Protocol
0	10.0.0.1	10.0.0.8	UDP
1	10.0.0.1	10.0.0.8	UDP
2	10.0.0.2	10.0.0.8	UDP
3	10.0.0.2	10.0.0.8	UDP
4	10.0.0.2	10.0.0.8	UDP

```
figure(figsize=(12, 7), dpi=80)
plt.barh(list(dict(data.src.value_counts()).keys()), dict(data.src.value_counts()).values(), color='lawngreen')
plt.barh(list(dict(data[data.label == 1].src.value_counts()).keys()), dict(data[data.label == 1].src.value_counts()).values(), color='blue')

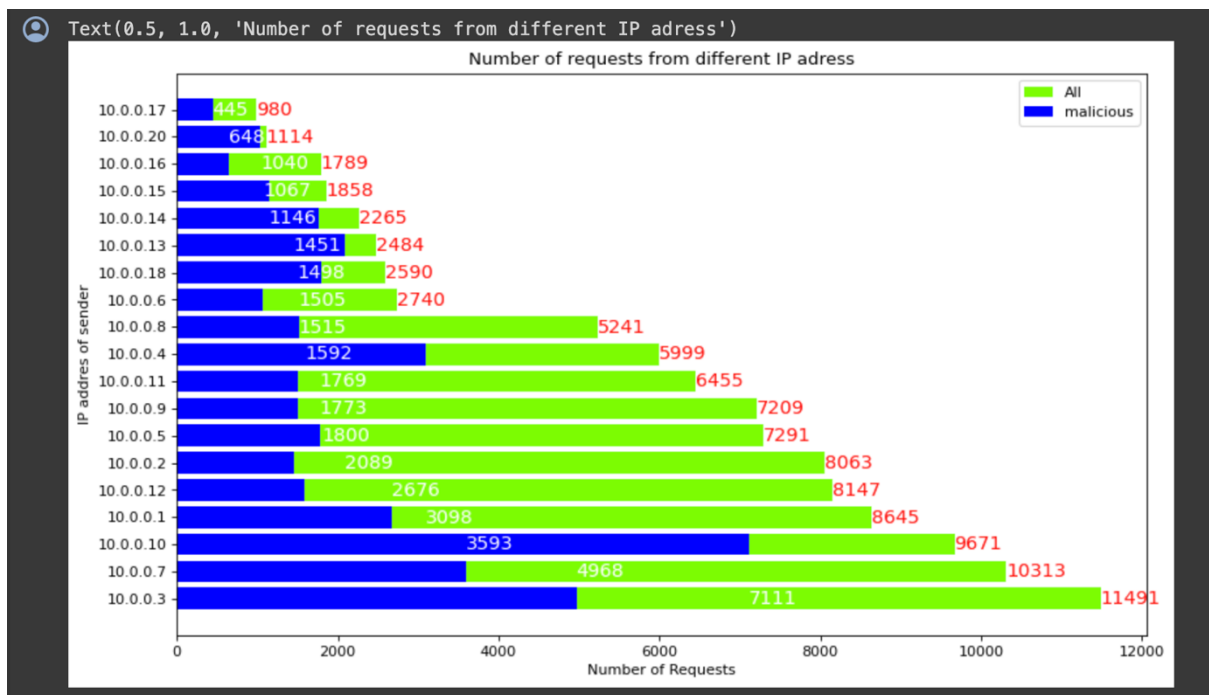
for idx, val in enumerate(dict(data.src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='r', size = 13)

for idx, val in enumerate(dict(data[data.label == 1].src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='w', size = 13)

plt.xlabel('Number of Requests')
plt.ylabel('IP address of sender')
plt.legend(['All', 'malicious'])
plt.title('Number of requests from different IP address')
```

Этот код создает горизонтальную столбчатую диаграмму, отображающую количество запросов от различных IP-адресов отправителей данных. Он также выделяет количество "вредоносных" запросов отдельным цветом.

Output:



```
figure(figsize=(10, 6), dpi=80)
plt.bar(list(dict(data.Protocol.value_counts()).keys()), dict(data.Protocol.value_counts()).values(), color='r')
plt.bar(list(dict(data[data.label == 1].Protocol.value_counts()).keys()), dict(data[data.label == 1].Protocol.value_counts()).values(), color='b')

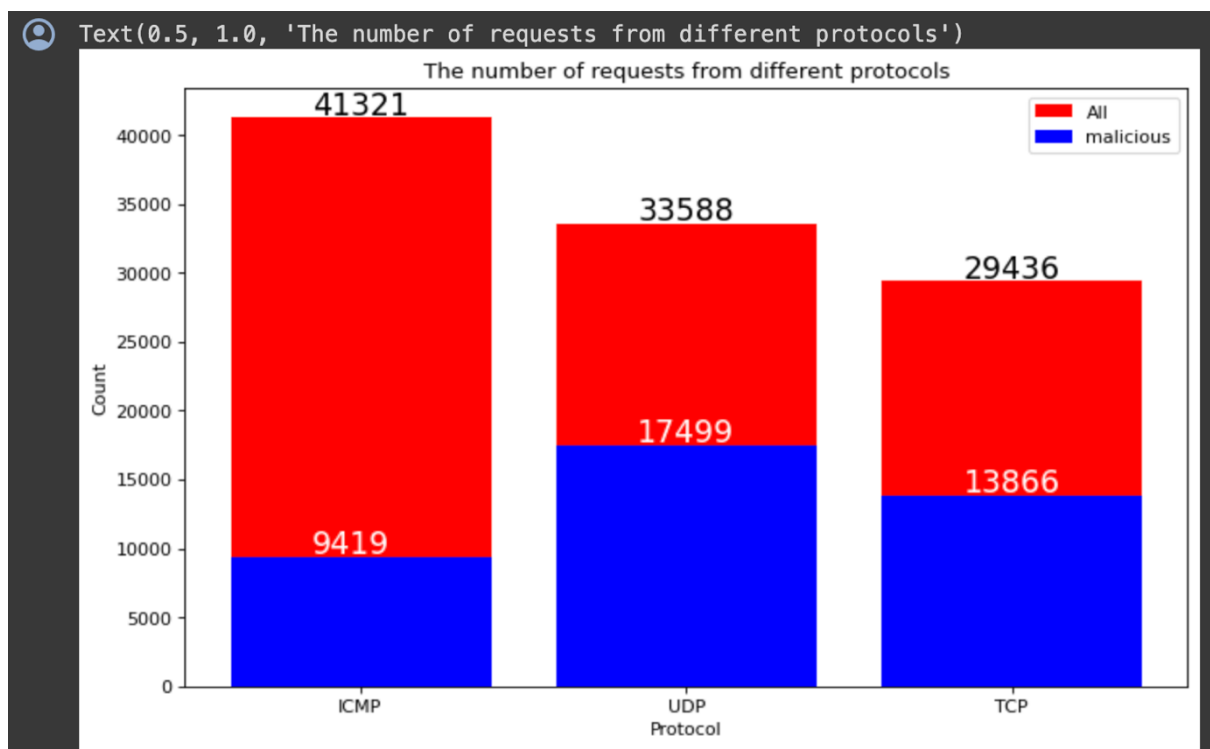
plt.text(x = 0 - 0.15, y = 41321 + 200, s = str(41321), color='black', size=17)
plt.text(x = 1 - 0.15, y = 33588 + 200, s = str(33588), color='black', size=17)
plt.text(x = 2 - 0.15, y = 29436 + 200, s = str(29436), color='black', size=17)

plt.text(x = 0 - 0.15, y = 9419 + 200, s = str(9419), color='w', size=17)
plt.text(x = 1 - 0.15, y = 17499 + 200, s = str(17499), color='w', size=17)
plt.text(x = 2 - 0.15, y = 13866 + 200, s = str(13866), color='w', size=17)

plt.xlabel('Protocol')
plt.ylabel('Count')
plt.legend(['All', 'malicious'])
plt.title('The number of requests from different protocols')
```

Этот код создает вертикальную столбчатую диаграмму, отображающую количество запросов, выполненных с использованием различных протоколов. Он также выделяет количество "вредоносных" запросов отдельным цветом.

Output:



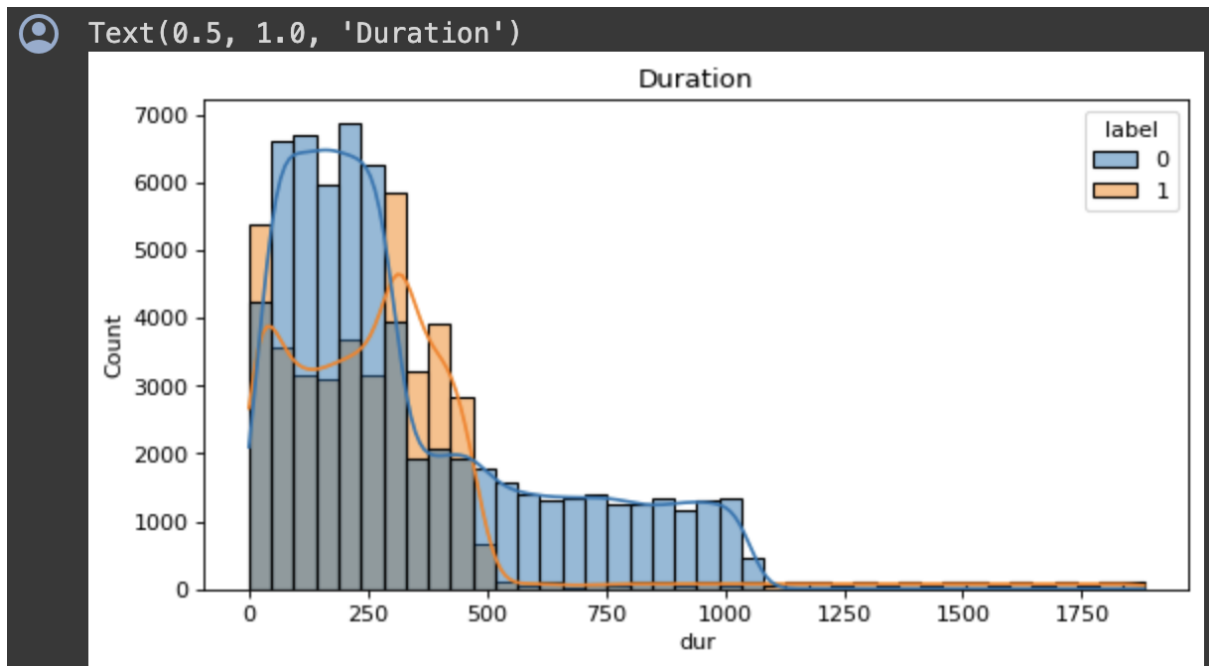
```
df = data.copy()
```

Этот код создает копию DataFrame `data` и присваивает ее переменной `df`. Создание копии позволяет работать с данными в новом DataFrame, не изменяя исходный DataFrame `data`.

```
figure(figsize=(8, 4), dpi=80)
sns.histplot(data=df, x="dur", hue="label", bins=40, kde=True).set_title("Duration")
```

Этот код использует библиотеку Seaborn для построения гистограммы распределения продолжительности запросов (`dur`) в зависимости от метки класса (`label`). Гистограмма разделена на две части с помощью параметра `hue`, одна для "нормальных" запросов, а другая для "вредоносных".

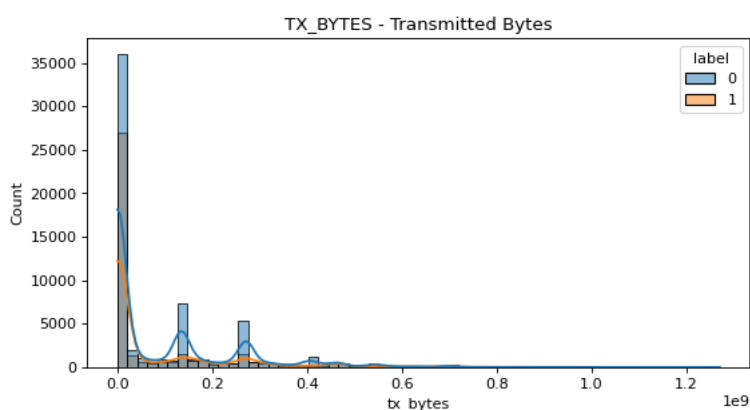
Output:



```
figure(figsize=(8, 4), dpi=80)
sns.histplot(data=df, x="tx_bytes", hue="label", bins=60, kde=True).set_title("TX_BYTES - Transmitted Bytes")
```

Этот код использует библиотеку Seaborn для создания гистограммы распределения, но уже для признака `tx_bytes` (количество переданных байтов). Он также разделяет данные на две части с помощью параметра `hue`, чтобы отобразить распределение для каждой метки класса (`label`).

Output:

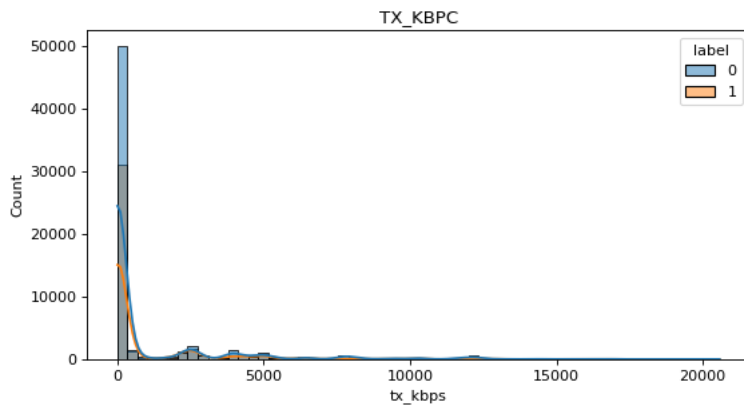


```
figure(figsize=(8, 4), dpi=80)
sns.histplot(data=df, x="tx_kbps", hue="label", bins=60, color='b', kde=True).set_title("TX_KBPS")
```

Этот код также использует библиотеку Seaborn для создания гистограммы распределения, но уже для признака `tx_kbps` (скорость передачи данных).

в килобайтах в секунду). Он также разделяет данные на две части с помощью параметра `hue`, чтобы отобразить распределение для каждой метки класса (`label`).

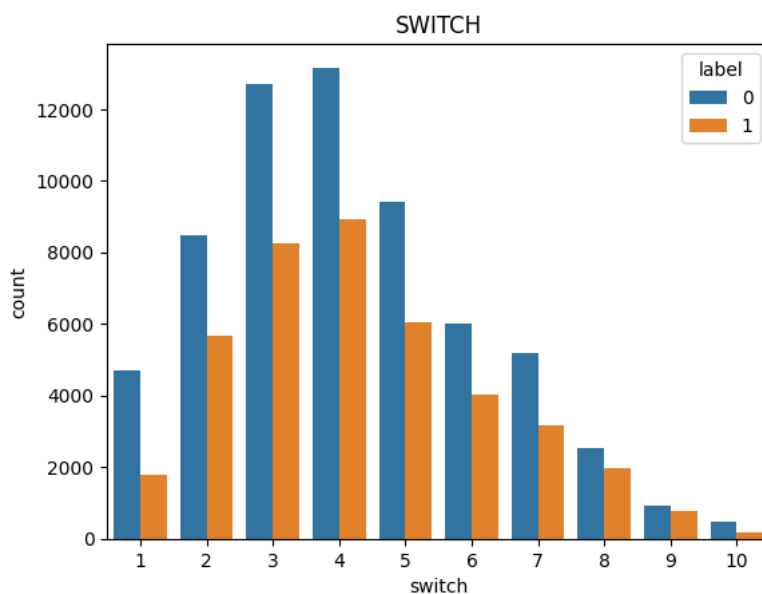
Output:



```
sns.countplot(data=df, x="switch", hue="label").set_title("SWITCH")
```

Этот код использует библиотеку Seaborn для создания столбчатой диаграммы подсчета количества уникальных значений в столбце `switch`, разделяя данные на разные группы с помощью параметра `hue` в зависимости от метки класса (`label`).

Output:



Классические модели ML


```

from pytorch_tabnet.tab_model import TabNetClassifier

class Model:
    global y
    def __init__(self, data):
        self.data = data
        X = preprocessing.StandardScaler().fit(self.data).transform(self.data)
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, random_state=42, test_size=0.3)

    def LogisticRegression(self):
        solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

        start_time = time.time()
        results_lr = []
        accuracy_list = []
        for solver in solvers:
            LR = LogisticRegression(C=0.03, solver=solver).fit(self.X_train, self.y_train)
            predicted_lr = LR.predict(self.X_test)
            accuracy_lr = accuracy_score(self.y_test, predicted_lr)
            #print("Accuracy: %.2f%%" % (accuracy_lr * 100.0))
            #print('#####')
            results_lr.append({'solver' : solver, 'accuracy': str(round(accuracy_lr * 100, 2)) + "%",
                              'Coefficients': {'W' : LR.coef_, 'b': LR.intercept_}})

            accuracy_list.append(accuracy_lr)

        solver_name = solvers[accuracy_list.index(max(accuracy_list))]
        LR = LogisticRegression(C=0.03, solver=solver_name).fit(self.X_train, self.y_train)
        predicted_lr = LR.predict(self.X_test)
        accuracy_lr = accuracy_score(self.y_test, predicted_lr)
        print("Accuracy: %.2f%%" % (accuracy_lr * 100.0), '\n')
        print("#####")
        print('Best solver is : ', solver_name)
        print("#####")
        print(classification_report(predicted_lr, self.y_test), '\n')
        print("#####")
        print("--- %s seconds --- time for LogisticRegression" % (time.time() - start_time))

    def KNearestNeighbor(self):
        start_time = time.time()
        Ks = 12
        accuracy_knn = np.zeros((Ks-1))
        std_acc = np.zeros((Ks-1))
        #print(accuracy_knn)
        for n in range(1, Ks):

```

```

        #Train Model and Predict
        neigh = KNeighborsClassifier(n_neighbors = n).fit(self.X_train, self.y_train)
        yhat=neigh.predict(self.X_test)
        accuracy_knn[n-1] = metrics.accuracy_score(self.y_test, yhat)

    std_acc[n-1]=np.std(yhat==self.y_test)/np.sqrt(yhat.shape[0])

    #print(accuracy_knn, '\n\n') # courserany ozinde tek osy gana jazylyp turdy
    #print(std_acc)
    #accuracy_knn[0] = 0
    plt.figure(figsize=(10,6))
    plt.plot(range(1,Ks), accuracy_knn, 'g')
    plt.fill_between(range(1,Ks), accuracy_knn - 1 * std_acc, accuracy_knn + 1 * std_acc, alpha=0.10)
    plt.fill_between(range(1,Ks), accuracy_knn - 3 * std_acc, accuracy_knn + 3 * std_acc, alpha=0.10, color="green")
    plt.legend(('Accuracy ', '+/- 1xstd', '+/- 3xstd'))
    plt.ylabel('Accuracy ')
    plt.xlabel('Number of Neighbors (K)')
    plt.tight_layout()
    plt.show()

    knnc = KNeighborsClassifier()
    knnc_search = GridSearchCV(knnc, param_grid={'n_neighbors': [3, 5, 10],
                                                'weights': ['uniform', 'distance'],
                                                'metric': ['euclidean', 'manhattan']},
                               n_jobs=-1, cv=3, scoring='accuracy', verbose=2)

    knnc_search.fit(self.X_train, self.y_train)
    #print(knnc_search.best_params_)
    #print(knnc_search.best_score_)
    n_neighbors = knnc_search.best_params_['n_neighbors']
    weights = knnc_search.best_params_['weights']
    metric = knnc_search.best_params_['metric']
    KNN = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric, weights=weights).fit(self.X_train, self.y_train)

    predicted_knn = KNN.predict(self.X_test)
    accuracy_knn = metrics.accuracy_score(self.y_test, predicted_knn)

    print(f"Accuracy of KNN model {round(accuracy_knn,2)*100}%", '\n')
    print("#####")
    print(classification_report(predicted_knn, self.y_test))
    print("#####")

```

```

        print("--- %s seconds ---" % (time.time() - start_time))

    def RandomForest(self):
        start_time = time.time()
        RF = RandomForestClassifier(criterion='gini',
                                   n_estimators=500,
                                   min_samples_split=10,
                                   #min_samples_leaf=1,
                                   max_features='auto',
                                   oob_score=True,
                                   random_state=1,
                                   n_jobs=-1).fit(self.X_train, self.
y_train)

        predicted_rf = RF.predict(self.X_test)
        svm_accuracy = accuracy_score(self.y_test, predicted_rf)
        print(f"Accuracy of RF is : {round(svm_accuracy*100,2)}%", '\n'
)

        print("#####")
        print(classification_report(predicted_rf, self.y_test))
        print("#####")

        print("--- %s seconds ---" % (time.time() - start_time))

    def Tabnet(self):
        start_time = time.time()

        classifier = TabNetClassifier(verbose=0, device_name="cuda")
        classifier.fit(X_train=self.X_train, y_train=self.y_train,
                      eval_set=[(self.X_test, self.y_test)],
                      patience=10, max_epochs=100,
                      eval_metric=['balanced_accuracy'])

        predicted = classifier.predict(self.X_test)
        acc = accuracy_score(self.y_test, predicted)
        print(f"Accuracy of TabNet is : {round(acc*100,2)}%", '\n')
        print("#####")
        print(classification_report(predicted, self.y_test))
        print("#####")

        print("--- %s seconds ---" % (time.time() - start_time))

```

Класс `Model` в коде является оберткой для моделей машинного обучения. Каждый метод выполняет обучение модели и выводит оценку ее качества с использованием различных метрик, таких как точность, полнота и F1-мера.

Предсказание:

```
df = data.copy()
df = df.dropna()
```

Этот код создает копию DataFrame `data` и присваивает ее переменной `df`. Затем он использует метод `dropna()` для удаления всех строк, содержащих пропущенные значения (NaN). Таким образом, переменная `df` содержит копию исходного DataFrame без пропущенных значений.

```
X = df.drop(['dt', 'src', 'dst', 'label'], axis=1)
y = df.label
```

Этот код создает матрицу признаков `X` и вектор меток `y` на основе DataFrame `df`.

- `X` содержит все признаки из DataFrame `df`, кроме `dt`, `src`, `dst`, так как эти признаки, вероятно, не несут полезной информации для модели. Они исключены с помощью метода `drop()` с указанием оси 1 (столбцы).

- `y` содержит только столбец меток `label`.

```
X = pd.get_dummies(X)
```

Этот код использует метод `get_dummies()` библиотеки Pandas для преобразования категориальных признаков в фиктивные переменные (one-hot encoding).

```
X
```

После преобразования категориальных признаков в фиктивные переменные с помощью метода `get_dummies()`, матрица признаков `X` теперь содержит только числовые значения. Она готова к использованию в моделях машинного обучения.

Output:

	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	...	Pairflow	port_no	tx_bytes	rx_bytes	tx_kbps	rx_k
0	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428310	...	0	3	143928631	3917	0	
1	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	14424046	...	0	4	3842	3520	0	
2	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	...	0	1	3795	1242	0	
3	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	...	0	2	3688	1492	0	
4	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	...	0	3	3413	3665	0	
...
104340	3	79	7742	81	842000000	8.184200e+10	5	10	29	2842	...	0	1	15209	12720	1	
104341	3	79	7742	81	842000000	8.184200e+10	5	10	29	2842	...	0	3	15099	14693	1	
104342	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	...	0	2	3409	3731	0	
104343	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	...	0	1	15209	12720	1	
104344	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	...	0	3	15099	14693	1	

103839 rows x 21 columns

```
M = Model(X)
```

Этот код создает объект класса `Model`, передавая матрицу признаков `X` в качестве аргумента конструктору класса.

Традиционный ML Для сравнения

```
M.LogisticRegression()
```

Этот код вызывает метод `LogisticRegression()` объекта `M`, который обучает модель логистической регрессии на данных `X` и метках `y`, а затем выводит оценку качества модели.

Output:

```

Accuracy: 76.64%

#####
Best solver is : liblinear
#####
              precision    recall  f1-score   support

         0       0.84        0.79        0.81        20024
         1       0.66        0.72        0.69        11128

 accuracy          0.77                31152
 macro avg         0.75        0.76        0.75        31152
 weighted avg      0.77        0.77        0.77        31152

#####
--- 7.808847427368164 seconds --- time for LogisticRegression

```

```
M.RandomForest()
```

Этот код вызывает метод `RandomForest()` объекта `M`, который обучает модель случайного леса на данных `X` и метках `y`, а затем выводит оценку качества модели.

Output:

```
 /usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in
warn(
Accuracy of RF is : 99.99%

#####
      precision    recall  f1-score   support

     0       1.00      1.00      1.00    18984
     1       1.00      1.00      1.00    12168

   accuracy          1.00      1.00      1.00    31152
  macro avg          1.00      1.00      1.00    31152
weighted avg          1.00      1.00      1.00    31152

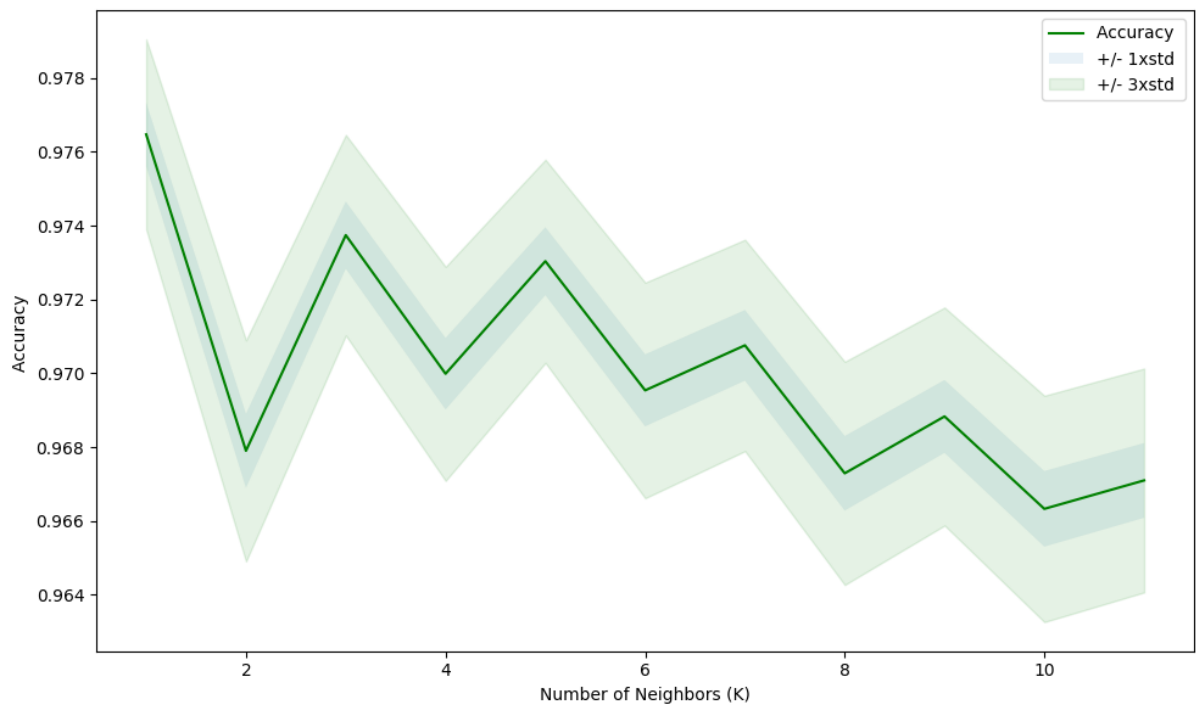
#####
--- 39.91285562515259 seconds ---
```

```
M.KNearetsNeighbor()
```

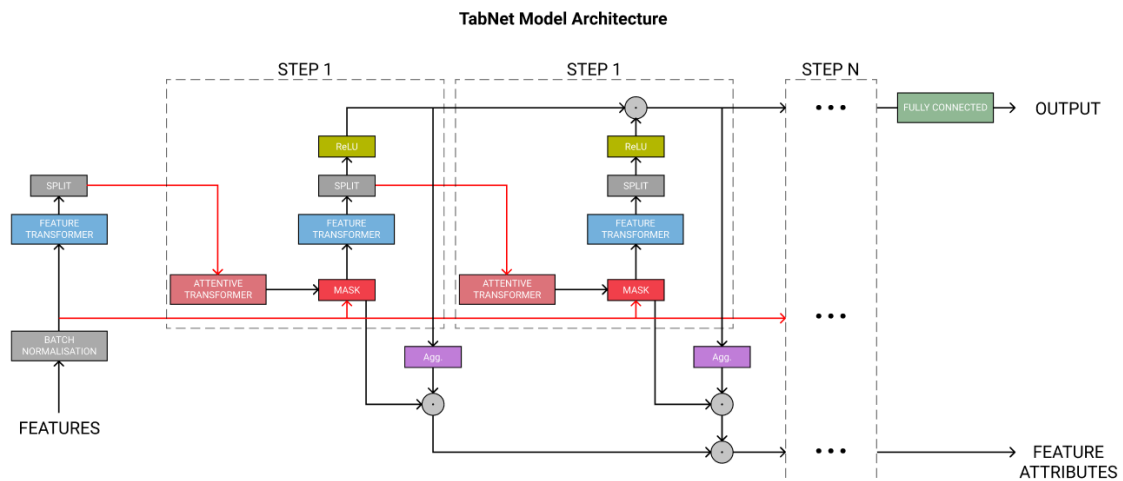
Этот код вызывает метод `KNearetsNeighbor()` объекта `M`, который обучает модель k-ближайших соседей на данных `X` и метках `y`, а затем выводит оценку качества модели.

После обучения он выводит график зависимости точности от числа соседей, а также выводит отчет о классификации, который содержит метрики качества модели, такие как точность, полнота и F1-мера для каждого класса.

Output:



Глубокое обучение с использованием Tabnet для обнаружения атак



M. Tabnet ()

Output:

```
Early stopping occurred at epoch 19 with best_epoch = 9 and best_val_0_balanced_accuracy = 0.97283
/usr/local/lib/python3.10/dist-packages/pytorch_tabnet/callbacks.py:172: UserWarning: Best weights from best epoch are automatically used!
warnings.warn(wrn_msg)
Accuracy of TabNet is : 96.84%

#####
      precision    recall  f1-score   support

     0       0.95      1.00      0.97      18167
     1       0.99      0.93      0.96      12985

 accuracy          0.97          0.97          0.97      31152
 macro avg          0.97          0.96          0.97      31152
 weighted avg          0.97          0.97          0.97      31152

#####
--- 102.08888459205627 seconds ---
```

Этот код вызывает метод `Tabnet()` объекта `M`, который обучает модель TabNet на данных `X` и метках `y`, а затем выводит оценку качества модели.

После обучения он выводит точность модели на тестовых данных, а также выводит отчет о классификации, который содержит метрики качества модели, такие как точность, полнота и F1-мера для каждого класса.

5. Итог проведенной работы

Лабораторная работа по классификации атак DDoS с использованием модели TabNet включал в себя следующие шаги:

1. Подготовка данных:

- Загрузка данных, содержащих информацию о сетевом трафике и метках класса атак.
- Предварительный анализ данных для понимания их структуры и особенностей.
- Предобработка данных, включая удаление пропущенных значений и преобразование категориальных признаков в числовые.

2. Обучение модели TabNet:

- Создание класса `Model`, содержащего методы для обучения и оценки модели TabNet.

- Обучение модели TabNet на предварительно обработанных данных.
- Оценка качества модели на тестовом наборе данных, включая вычисление метрик точности, полноты и F1-меры.

3. Анализ результатов:

- Оценка точности и эффективности модели TabNet в классификации атак DDoS.
- Идентификация наиболее важных признаков, влияющих на классификацию.

4. Выводы и рекомендации:

- Обсуждение результатов и возможных путей улучшения качества модели.
- Оценка применимости модели в реальных условиях и ее потенциального вклада в обнаружение атак DDoS.

В итоге, лабораторная работа позволила оценить эффективность модели TabNet в задаче классификации атак DDoS и предоставил базу для дальнейших исследований и улучшений в этой области.

6. Литература

1. <https://doi.org/10.1515/jisys-2022-0155>
2. <https://www.kaggle.com/code/dbzadnen/ddos-attack-detection-classification-with-tabnet/notebook>
3. <https://www.kaggle.com/datasets/aikenkazin/ddos-sdn-dataset/data>
4. <https://www.geeksforgeeks.org/bidirectional-lstm-in-nlp/amp/>

