Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки «Анализ данных»

**Система обнаружения атак в сети с помощью методов машинного обучения**

Разработчики проекта:

Заколодкин Егор Юрьевич,

Нижанковский Артём Дмитриевич,

Стрельченко Алексей Александрович

Пермь, 2024

**Оглавление**

[ПАСПОРТ ПРОЕКТА 3](#_Toc185185412)

[СОДЕРЖАНИЕ ПРОЕКТА 4](#_Toc185185413)

[Анализ проблемы исследования 4](#_Toc185185414)

[Исходные данные 5](#_Toc185185415)

[Реализация проекта 8](#_Toc185185416)

[Бинарная классификация 8](#_Toc185185417)

[Этап 1. Подготовка и анализ данных 8](#_Toc185185418)

[Этап 2. Моделирование 26](#_Toc185185419)

[Мультиполиномиальная классификация 46](#_Toc185185420)

[Этап 1. Подготовка и анализ данных 46](#_Toc185185421)

[Этап 2. Моделирование 59](#_Toc185185422)

[Заключение 76](#_Toc185185423)

# **ПАСПОРТ ПРОЕКТА**

**Название проекта:**Система обнаружения атак в сети с помощью методов машинного обучения.

**Сведения об авторах:**Заколодкин Егор Юрьевич, Нижанковский Артём Дмитриевич, Стрельченко Алексей Александрович.

**Цель:**изучить и проанализировать методы обнаружения сетевых атак в системах обнаружения вторжений (IDS) с использованием машинного обучения для повышения их практической применимости в коммерческих продуктах.

**Задачи:**

1. Разработать модель биномиальной классификации для определения, является ли сетевая активность нормальной или представляет собой атаку.
2. Разработать модель мультиномиальной классификации для классификации сетевой активности на несколько категорий: нормальная (не атака), DOS, PROBE, R2L, U2R.

**Краткое описание проекта:**

Проект направлен на разработку системы обнаружения вторжений (IDS), способной идентифицировать аномалии и атаки в компьютерной сети.

Для реализации проекта будет использоваться общеизвестный набор данных KDDCUP'99, который содержит ключевую информацию о сетевых соединениях.

**Конкретные ожидаемые результаты:**

Анализ нескольких моделей бинарной и мультиномиальной классификации и выбор лучших из них для дальнейшего использования в системах обнаружения атак.

# **СОДЕРЖАНИЕ ПРОЕКТА**

## **Анализ проблемы исследования**

С огромным ростом использования компьютерных сетей и увеличением числа приложений, работающих на их основе, безопасность сети приобретает все большее значение. Все компьютерные системы страдают от уязвимостей в системе безопасности, устранение которых производителями является технически сложным и экономически дорогостоящим делом. Это могут быть ошибки в программном обеспечении, недостатки в конфигурациях, а также проблемы с управлением доступом. Производители программного обеспечения часто сталкиваются с трудностями в быстрой идентификации и исправлении этих уязвимостей, что делает системы уязвимыми для атак.

Поэтому роль систем обнаружения вторжений (IDS), как устройств специального назначения для обнаружения аномалий и атак в сети, становится все более важной. Они предназначены для мониторинга деятельности в сети и обнаружения аномалий или подозрительных действий, которые могут свидетельствовать о попытках вторжения.

## **Исходные данные**

Этот набор данных использовался для Третьего Международного конкурса инструментов поиска знаний и интеллектуального анализа данных, который проводился совместно с Пятой Международной конференцией по поиску знаний и интеллектуальному анализу данных KDD-99. Конкурсная задача состояла в создании сетевого детектора вторжений - прогностической модели, способной отличать «плохие» соединения, называемые вторжениями или атаками, от «хороших» обычных соединений.

Список колонок анализируемого набора данных:

1. **duration** – продолжительность подключения по времени.
2. **protocol\_type** – протокол транспортного уровня, используемый при подключении.
3. **Service** – протокол прикладного уровня, используемый при подключении.
4. **Flag** – статус подключения: нормальный или ошибка
5. **Src\_bytes** – количество байт данных, переданных от источника к получателю в одном соединении.
6. **Dst\_bytes** – количество байт данных, переданных от получателя к источнику в одном соединении.
7. **Land** – если IP-адреса источника и назначения и номера портов равны, то эта переменная принимает значение 1, иначе 0
8. **Wrong\_fragment** – общее количество неверных фрагментов в этом соединении.
9. **Urgent** – количество срочных пакетов в этом соединении. Срочные пакеты – это пакеты, которые имеют специальный бит, активируемый в заголовке, указывающий на то, что эти данные нуждаются в приоритетной обработке.
10. **Hot** – количество "горячих" индикаторов в содержимом. Эти индикаторы указывают на количество действий или событий, которые могут вызывать повышенное внимание или сигнализировать о потенциальных угрозах.
11. **Num\_failed\_logins** – количество неудачных попыток входа в систему.
12. **Logged\_in Login Status** – «1» при успешном входе в систему; «0» в противном случае.
13. **Num\_compromised** – количество «скомпрометированных» условий.
14. **Root\_shell** – «1», если получена командная оболочка; «0» в противном случае.
15. **Su\_attempted** – «1» при попытке или использовании команды «root»; «0» в противном случае.
16. **Num\_root** – количество обращений «root» или количество операций, выполняемых от имени «root» в соединении.
17. **Num\_file\_creations** – количество операций создания файлов в соединении.
18. **Num\_shells** – количество запросов оболочки.
19. **Num\_access\_files** – количество операций с файлами контроля доступа.
20. **Num\_outbound\_cmds** – количество исходящих команд в сеансе ftp.
21. **Is\_hot\_login** – «1», если логин принадлежит к "горячему" списку, т.е. root или admin; в противном случае «0».
22. **Is\_guest\_login** – «1», если логин является «гостевым»; «0» в противном случае.
23. **Count** – количество подключений к тому же узлу назначения, что и текущее подключение, за последние две секунды.
24. **Srv\_count** – количество подключений к той же службе (номер порта), что и текущее подключение, за последние две секунды.
25. **Serror\_rate** – процент соединений, у которых значения флага (признак №4) является s0, s1, s2 или s3, среди соединений, объединенных в число (признак №23).
26. **Srv\_serror\_rate** – процент соединений, у которых значения флага (признак №4) является s0, s1, s2 или s3, среди соединений, объединенных в число (признак №24).
27. **Rerror\_rate** – процент соединений, у которых значения флага (признак №4) является REJ, среди соединений, объединенных в число (признак №23).
28. **Srv\_rerror\_rate** – процент соединений, у которых значения флага (признак №4) является REJ, среди соединений, объединенных в число (признак №24).
29. **Same\_srv\_rate** – процент подключений, которые были связаны с одним и тем же сервисом, среди подключений, объединенных в число (признак №23).
30. **Diff\_srv\_rate** – процент подключений к различным сервисам среди подключений, объединенных в число (признак №23).
31. **Srv\_diff\_host\_rate** – процент подключений, которые были установлены к различным компьютерам назначения, среди всех подключений, объединённых в число (признак №24).
32. **Dst\_host\_count** – количество подключений, имеющих один и тот же IP-адрес узла назначения.
33. **Dst\_host\_srv\_count** – количество подключений с одинаковым номером порта.
34. **Dst\_host\_same srv\_rate**– процент подключений к одному и тому же сервису среди подключений, агрегированных в dst\_host\_count (признак №32).
35. **Dst\_host\_diff srv\_rate** – процент подключений к различным сервисам среди подключений, агрегированных в dst\_host\_count (признак №32).
36. **Dst\_host\_same src\_port\_rate**– процент подключений, которые были подключены к одному и тому же исходному порту, среди подключений, агрегированных в dst\_host\_srv\_count (признак №33).
37. **Dst\_host\_srv diff\_host\_rate** – процент подключений, которые были установлены к различным компьютерам назначения, среди подключений, агрегированных в dst\_host\_srv\_count (признак №33).
38. **Dst\_host\_serro r\_rate** – процент соединений, у которых значения флага (признак №4) является s0, s1, s2 или s3, среди соединений, объединенных в число (признак №32).
39. **Dst\_host\_srv\_s error\_rate** – процент соединений, у которых значения флага (признак №4) является s0, s1, s2 или s3, среди соединений, объединенных в число (признак №33).
40. **Dst\_host\_rerro r\_rate** – процент соединений, у которых значения флага (признак №4) является REJ, среди соединений, объединенных в число (признак №32).
41. **Dst\_host\_srv\_r error\_rate** – процент соединений, у которых значения флага (признак №4) является REJ, среди соединений, объединенных в число (признак №33).

Необходимо проанализировать данные по сетевым соединениям и определить, возможно ли отличить нормальное соединение от сетевой атаки на основе имеющихся признаков различными методами машинного обучения.

## **Реализация проекта**

### Бинарная классификация

#### **Этап 1. Подготовка и анализ данных**

Загрузим данные в датафрейм и подключим необходимые библиотеки:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from collections import Counter

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from imblearn.over\_sampling import SMOTE, ADASYN

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

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score, classification\_report, precision\_recall\_curve

from sklearn.metrics import ConfusionMatrixDisplay

import warnings

warnings.filterwarnings("ignore")

Изначальные данные хранятся в двух датасетах: Train.txt и Test.txt. У нас нет информации, каким образом автор разделил выборку на эти два датасета. Поэтому мы сначала объединили их, провели анализ, а при обучении сами разделили единый датасет на обучающую и тестовую выборки.

**Загрузим данные:**

df = pd.read\_csv("Train.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

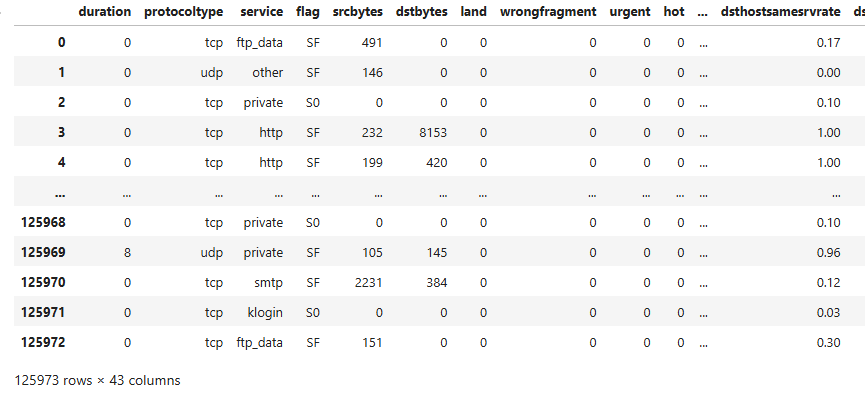


Рисунок 1. Исходный датафрейм (Train.txt)

df\_test = pd.read\_csv("Test.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

Объединяем датасеты:

combined\_df = pd.concat([df, df\_test], ignore\_index=True)

**Выводим статистическую информацию:**

Вычислим описательные статистики по колонкам (среднее, моду, медиану, стандартное отклонение, квартили).

***Среднее арифметическое*** *равно сумме значений всех вариант выборки, деленной на объем выборки:*

.

Здесь *п* − объем выборки, а *xi* − варианты выборки.

***Модой*** называется значение признака, встречающееся в выборке наиболее часто. Условимся использовать для обозначения моды символы *Mo*.

В случае несгруппированных данных для нахождения медианы необходимо ранжировать выборку, т. е. расположить данные в порядке их возрастания или убывания. Медианой будет являться значение признака, находящееся в середине ранжированного ряда. Медиана находится по формуле

Выборочная дисперсия находится по формуле *.*

Используется также другая формула для вычисления дисперсии: , где *.*

Дисперсия имеет размерность квадрата размерности случайной величины, что затрудняет ее интерпретацию и делает не очень наглядной. Для более наглядного описания рассеяния удобнее пользоваться характеристикой, размерность которой совпадает с размерностью исследуемого признака. С этой целью вводится понятие ***стандартного отклонения*** (или ***среднего квадратичного отклонения***).

***Стандартным отклонением*** называется положительный квадратный корень из дисперсии:

.

Стандартное отклонение имеет те же единицы измерения, что и результаты измерения исследуемого признака, и, таким образом, оно характеризует степень отклонения признака от среднего арифметического. Иными словами, оно показывает, как расположена основная часть вариант относительно среднего арифметического.

combined\_df.describe() #вывод статистической информации

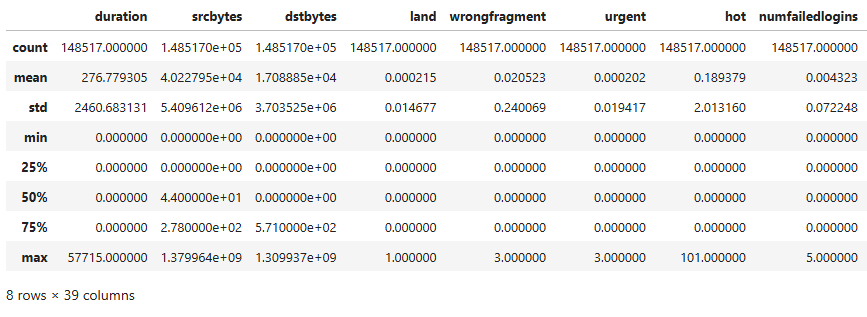


Рисунок 2. Общая информация

Как мы видим, «land», «urgent», «numfailedlogins», «numoutboundcmds», «rootshell», «suattempted», «numshells», «numaccessfiles», «ishostlogin» имеют в основном нулевые значения**, поэтому мы можем удалить эти столбцы:**

combined\_df.drop(['land','urgent','numfailedlogins','numoutboundcmds', 'rootshell', 'suattempted', 'numshells', 'numaccessfiles', 'ishostlogin'],axis=1,inplace=True)

**Проверим, есть ли пустые значения в датасете:**

combined\_df.isnull().values.any() #Проверка отсутствующих значений NaN



Рисунок 3. Наличие пустых значений в датасете

**Выведем названия нечисловых (категориальных) признаков:**

combined\_df.select\_dtypes(include=['category', 'object']).columns #вывод категориальных переменных



Рисунок 5. Список категориальных признаков

**Для бинарной классификации заменяем в столбце «Attack» все виды атак на одно значение «attack»:**

combined\_df['attack'].loc[combined\_df['attack']!='normal']='attack'

**Проверяем, являются ли классы «normal» и «attack» сбалансированными. Для этого выводим количество значений в каждом классе:**

count = combined\_df['attack'].value\_counts()

# Вывод количества значений

print(count)

# Построение гистограммы

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"')

plt.xlabel('Тип атаки')

plt.ylabel('Количество')

plt.xticks(rotation=0)

plt.show()

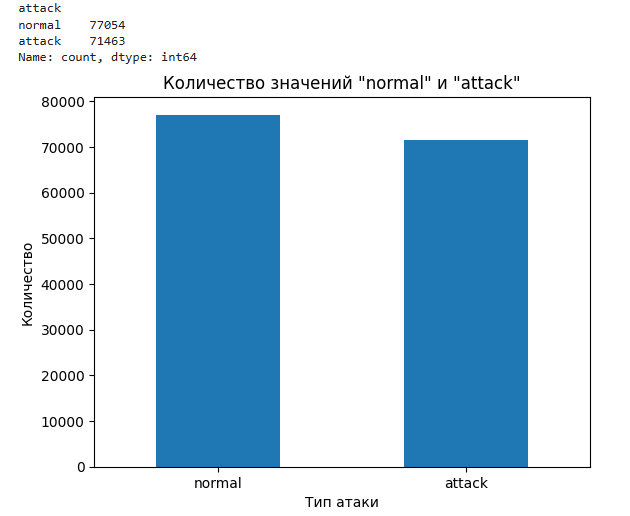


Рисунок 6. Гистограмма распределения классов

Как видно, классы являются сбалансированными.

**Выведем эти же значения в процентном соотношении:**

percentage\_counts = combined\_df['attack'].value\_counts(normalize=True) \* 100

print("Нормальный трафик:", percentage\_counts.iloc[0], '%')

print("Трафик злоумышленника:", percentage\_counts.iloc[1], '%')



Рисунок 7. Доля разных видов трафика

**Заменим категориальные данные количественными с помощью LabelEncoder.**

**LabelEncoder** — это инструмент из библиотеки scikit-learn, который используется для преобразования категориальных признаков в числовые значения. Он полезен, когда есть переменные, которые представляют собой категории, и их нужно подготовить для использования в алгоритмах машинного обучения.

Вот как работает LabelEncoder:

- Идентификация уникальных категорий: LabelEncoder сканирует данные и находит все уникальные категории в категориальном признаке.

- Присвоение целых чисел: Каждой уникальной категории присваивается целое число. Например, если есть категории "красный", "зеленый" и "синий", LabelEncoder может присвоить "красный" = 0, "зеленый" = 1, "синий" = 2.

le=LabelEncoder()

# Преобразуем столбец 'protocoltype' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['protocoltype'] = le.fit\_transform(combined\_df['protocoltype'])

# Преобразуем столбец 'service' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['service'] = le.fit\_transform(combined\_df['service'])

# Преобразуем столбец 'flag' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['flag'] = le.fit\_transform(combined\_df['flag'])

# Преобразуем столбец 'attack' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['attack'] = le.fit\_transform(combined\_df['attack'])

**Проведём корреляционный анализ. Для этого построим матрицу корреляции Пирсона.**

**Парный коэффициент** корреляции характеризует взаимосвязь двух переменных на фоне действия остальных показателей и является самым распространенным показателем тесноты связи при статистическом анализе данных.

Парный коэффициент корреляции между количественными случайными переменными и носит название *выборочного коэффициента корреляции* Пирсона (*sample correlation coefficient*) (или просто коэффициента корреляции) и находится по формуле

где и  — *выборочные дисперсии (sample variances*) переменных  и , а — *выборочная ковариация* или выборочный ковариационный момент, и соответствующие *средние (means)* определяются по формулам



Коэффициент корреляции обладает следующими свойствами:

1. Принимает значения от –1 до +1.
2. Если , то связь между переменными  и  считается сильной. Если , то связь слабая.
3. Если , то корреляционное поле наблюдений представляет собой совокупность точек, которые можно расположить на одной прямой. Знак «+» свидетельствует о прямой линейной зависимости между переменными и , а знак «—» − об обратной линейной зависимости.
4. При  линейная корреляционная связь отсутствует.

Метод ранговой корреляции Спирмена позволяет определить тесноту (силу) и направление корреляционной связи между двумя признаками (как количественными, так и качественными). Коэффициент ранговой корреляции имеет границы изменения от –1 до +1. Полное совпадение рангов означает максимально тесную прямую связь, полная противоположность рангов – максимально тесную обратную связь. Формула расчета ***коэффициента корреляции рангов Ч. Спирмена:***

где  – ранг  в выборке .

# Создаем фигуру и ось для отображения матрицы корреляций

f, ax = plt.subplots(1, 1, figsize=(35, 24))

# Вычисляем корреляцию между признаками нового сбалансированного DataFrame

correlation\_matrix = combined\_df.corr(method='spearman'

)

# Построим тепловую карту для визуализации матрицы корреляций

# Используем палитру 'coolwarm' для улучшения восприятия

sns.heatmap(correlation\_matrix, annot=True, fmt=".4f", cmap='coolwarm', ax=ax)

# Устанавливаем заголовок к графику

ax.set\_title('Матрица корреляций сбалансированной подвыборки', fontsize=14)

# Отображаем график

plt.show()

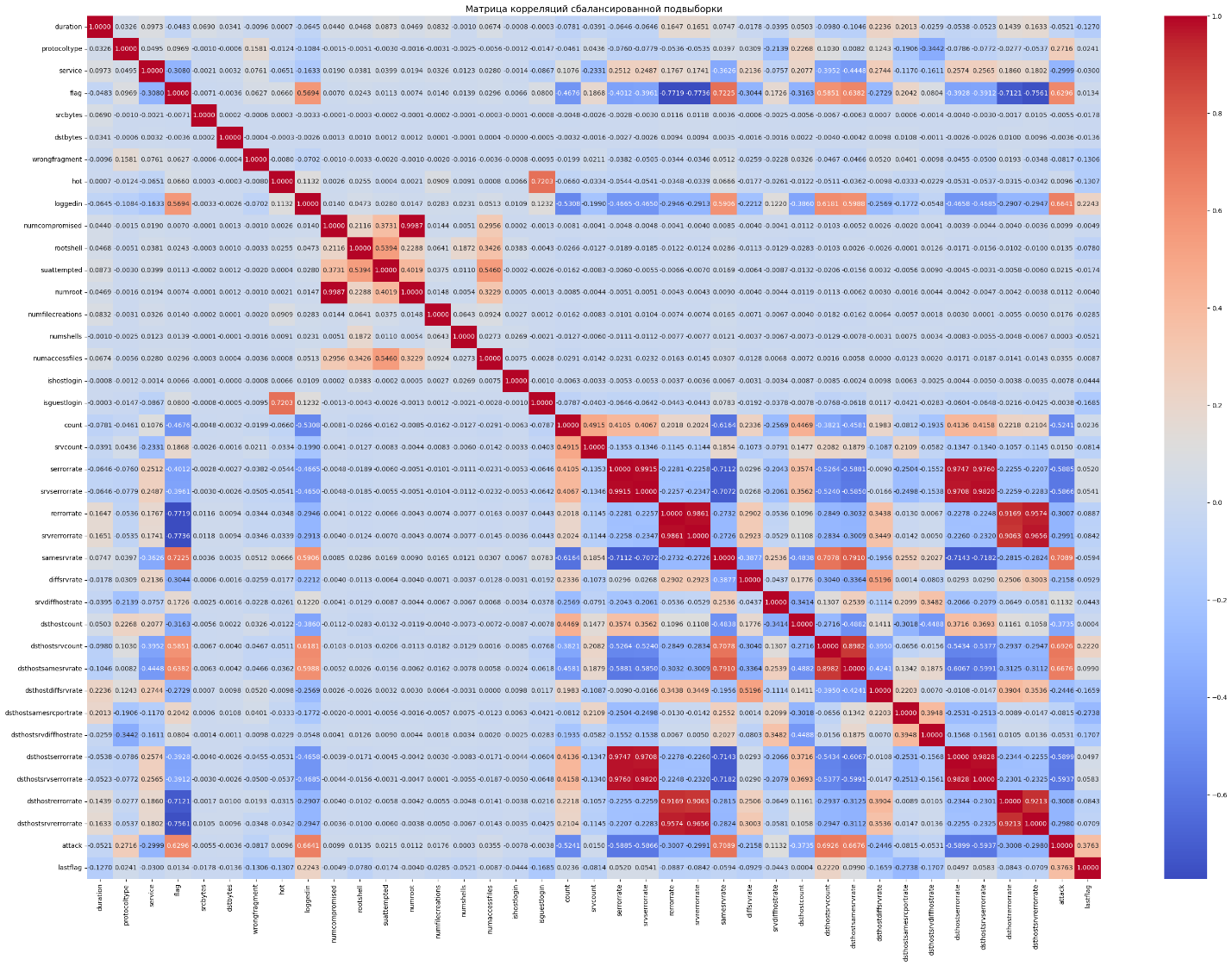


Рисунок 8. Матрица корреляции

По матрице корреляции видно, что некоторые признаки сильно коррелируют между собой (по сути, обозначают одно и то же) (коэффициент корреляции больше 0,9):

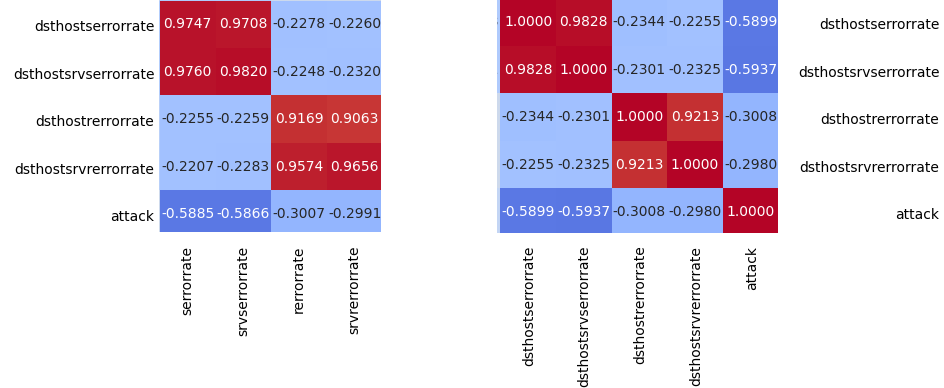


Рисунок 9. Зависимые столбцы



Рисунок 10. Зависимые столбцы

**Из каждой пары удалим один признак:**

combined\_df.drop(['numroot','serrorrate','rerrorrate','dsthostrerrorrate', 'srvserrorrate', 'dsthostserrorrate', 'srvrerrorrate', ],axis=1,inplace=True)

**Ещё раз выводим матрицу корреляции:**

# Создаем фигуру и ось для отображения матрицы корреляций

f, ax = plt.subplots(1, 1, figsize=(35, 24))

# Вычисляем корреляцию между признаками нового сбалансированного DataFrame

correlation\_matrix = combined\_df.corr()

# Построим тепловую карту для визуализации матрицы корреляций

# Используем палитру 'coolwarm' для улучшения восприятия

sns.heatmap(correlation\_matrix, annot=True, fmt=".4f", cmap='coolwarm', ax=ax)

# Устанавливаем заголовок к графику

ax.set\_title('Матрица корреляций сбалансированной подвыборки', fontsize=14)

# Отображаем график

plt.show()

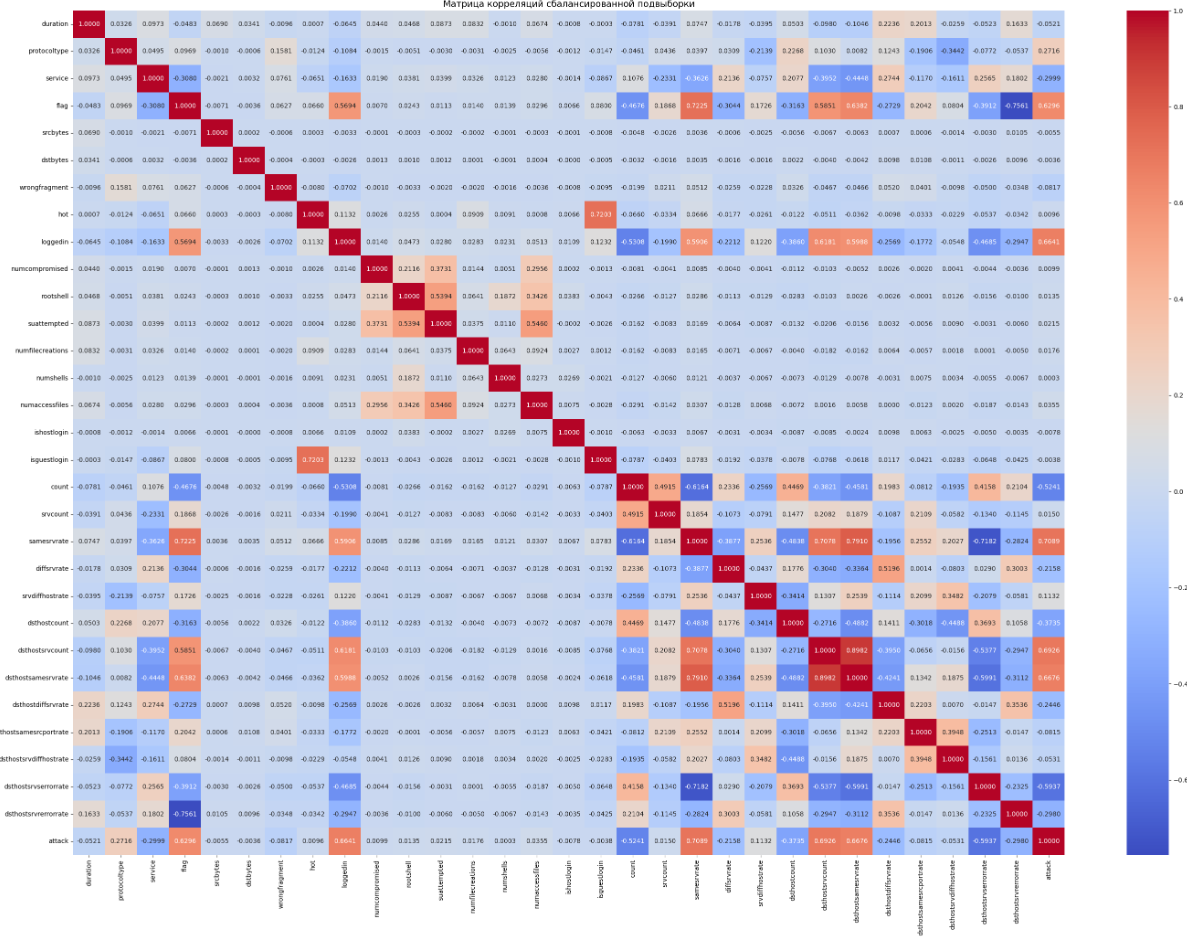


Рисунок 11. Результирующая матрица корреляции

**Выделим отдельно целевую переменную для дальнейшего обучения:**

X=combined\_df.drop(['attack'],axis=1)

y=combined\_df['attack']

**Далее проверим важность каждого признака с помощью модели случайного дерева, перед этим нормализовав признаки.**

**StandardScaler** — это инструмент из библиотеки scikit-learn, который используется для стандартизации признаков в наборе данных. Он масштабирует данные таким образом, чтобы они имели нулевое среднее и единичную дисперсию. Это часто необходимо для алгоритмов машинного обучения, которые чувствительны к масштабу данных (например, для методов, основанных на расстоянии).

Вот как работает StandardScaler:

- Расчет среднего и стандартного отклонения: StandardScaler вычисляет среднее и стандартное отклонение для каждого признака в обучающем наборе данных.

- Применение трансформации: для каждого значения признаков выполняется преобразование по формуле:

standardized\_value = (value - mean) / std\_dev

Где:

- value — исходное значение

- mean — среднее значение этого признака

- std\_dev — стандартное отклонение этого признака

- Возврат стандартизированных данных: StandardScaler возвращает новые данные, в которых среднее значение каждого признака равно 0, а стандартное отклонение равно 1.

# Создаем объект StandardScaler

scaler = StandardScaler()

# Обучаем scaler на данных X

scaler.fit(X)

# Применяем трансформацию к тренировочным и тестовым данным

X\_dop = scaler.transform(X)

# Превращаем numpy массивы обратно в DataFrame

X\_dop = pd.DataFrame(X\_dop, columns=X.columns)

# Преобразование сбалансированного набора данных в массивы значений

X\_ = X\_dop.values # Признаки

y\_ = y.values # Целевая переменная

# Получаем имена признаков из DataFrame

feature\_names = X.columns.tolist() # Список имен столбцов в X

**Как работает метод случайного леса**

Метод случайного леса строится на основе ансамбля деревьев решений. Основная идея заключается в том, чтобы создать множество деревьев решений и объединить их результаты для получения более точного прогноза. Вот основные шаги работы метода случайного леса:

1. **Создание подвыборок данных**: Для каждого дерева создается случайная подвыборка из исходного набора данных с возвращением (bootstrap). Это означает, что некоторые объекты могут быть выбраны несколько раз, а некоторые могут не попасть в подвыборку вовсе. Такой подход позволяет создать разнообразные обучающие выборки и повысить устойчивость модели.
2. **Построение деревьев решений**: Для каждой подвыборки строится дерево решений. При этом на каждом узле дерева выбирается случайное подмножество признаков для разделения. Это помогает снизить корреляцию между деревьями и улучшить общую производительность модели.
3. **Агрегация результатов**: Для задачи классификации используется голосование большинства, а для задачи регрессии — усреднение предсказаний всех деревьев. Таким образом, итоговый прогноз получается более точным и устойчивым к шуму в данных.

# Инициализация модели Random Forest для классификации

model = ensemble.RandomForestClassifier(n\_estimators=100, # Количество деревьев в лесу

criterion="entropy", # Критерий для оценки качества разбиения

random\_state=0) # Задаем seed для воспроизводимости

# Обучение модели на подготовленных данных

model.fit(X\_, y\_)

# Оценка важности признаков

importances = model.feature\_importances\_ # Оценки важности всех признаков модели

**Создадим DataFrame для хранения важностей признаков и выведем графики:**

# Создаем DataFrame для хранения важностей признаков

dtf\_importances = pd.DataFrame({

"IMPORTANCE": importances, # Столбец с важностями

"VARIABLE": feature\_names # Столбец с именами признаков

}).sort\_values("IMPORTANCE", ascending=False) # Сортируем по убыванию важности

# Добавляем столбец с кумулятивной суммой важностей

dtf\_importances['cumsum'] = dtf\_importances['IMPORTANCE'].cumsum(axis=0)

# Устанавливаем имена признаков в качестве индексов

dtf\_importances = dtf\_importances.set\_index("VARIABLE")

## Настройка графика

'''

Формируем надписи для графика

- ticks: местоположения меток по оси x (можно передать пустой список для удаления меток).

- labels: метки для размещения в заданных местах тиков (можно передать только в случае наличия тиков).

- kwargs: текстовые свойства для изменения оформления меток.

Функция Matplotlib xticks() возвращает:

- locs: список местоположений xtick.

- labels: список текстовых объектов xlabel.

plt.grid: Настройка линий сетки.

'''

# Создание графика

fig, ax = plt.subplots(nrows=1, ncols=1, sharex=False, sharey=False, figsize=(10, 5), dpi=100) # Инициализация фигуры и осей

fig.suptitle("Features Importance", fontsize=20) # Заголовок графика

ax.title.set\_text('Variables') # Подпись оси

# Построение горизонтальной столбчатой диаграммы важности признаков

dtf\_importances[["IMPORTANCE"]].sort\_values(by="IMPORTANCE").plot(

kind="barh", # Тип графика - горизонтальная столбчатая диаграмма

legend=False, # Отключаем легенду

ax=ax # Указываем ось для построения

).grid(axis="x") # Настраиваем сетку по оси x

ax.set(ylabel="") # Удаляем метку по оси y, если она не нужна

plt.xticks(rotation=70) # Поворот меток по оси x на 70 градусов для лучшей читаемости

plt.grid(axis='both') # Настройка линий сетки по обеим осям

'''

Отрисовываем график

'''

plt.show() # Показываем график на экране

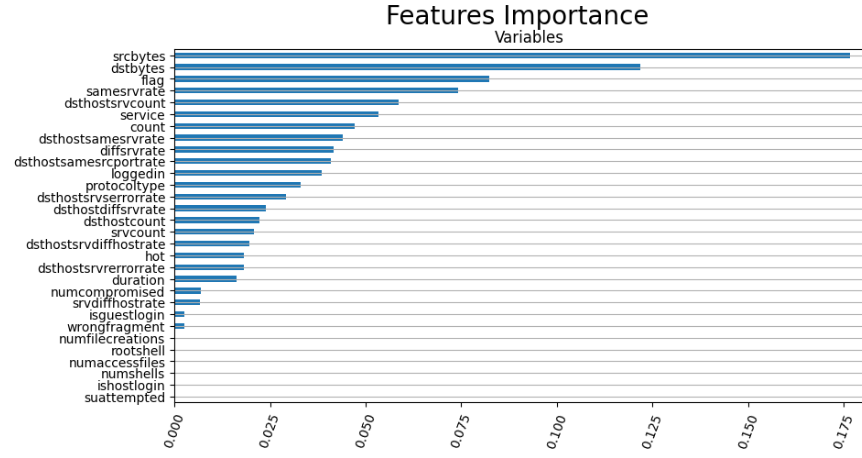
****

Рисунок 12. График важности признаков

**Удалим наименее важные признаки:**

# Указанные столбцы, которые мы хотим сохранить

columns\_to\_keep = ['srcbytes', 'protocoltype', 'count', 'dstbytes', 'service', 'diffsrvrate', 'flag', 'srvcount', 'samesrvrate', 'dsthostsrvcount', 'dsthostdiffsrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate', 'dsthostsrvserrorrate', 'loggedin', 'dsthostsamesrvrate']

# Удаляем все столбцы, кроме указанных

X = X[columns\_to\_keep]

**Делим датасет на обучающую и тестовую выборки:**

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

**Нормализуем обучающую и тестовую выборки (в предыдущий раз нормализовались другие выборки для определения важности признаков):**

# Создаем объект StandardScaler, который будет использоваться для стандартизации данных

scaler = StandardScaler()

# Применяем метод fit к данным X, чтобы вычислить среднее и стандартное отклонение,

# необходимые для стандартизации

scaler.fit(X)

# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,

# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)

X\_transformed = scaler.transform(X\_train)

# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,

# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)

X\_transformed\_test = scaler.transform(X\_test)

#### **Этап 2. Моделирование**

**Обучим модель Random Forest:**

#Случайный лес (Random Forest) # Создаем модель случайного леса

rf\_classifier = RandomForestClassifier() # Обучаем модель rf\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = rf\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

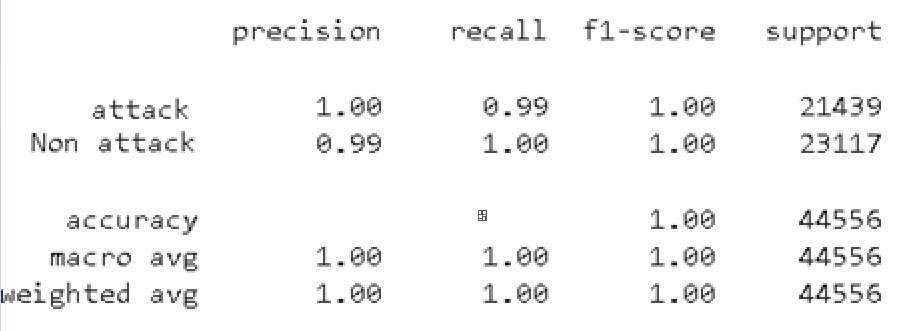
.

Рисунок 13. Отчет об классификации для модели случайного леса

**Precision (Точность):**

- Это доля правильно предсказанных положительных случаев среди всех предсказанных положительных.

- Формула: Precision = True Positives / (True Positives + False Positives).

- Чем выше, тем лучше. Высокая точность означает, что модель уделяет меньше внимания ложным срабатываниям.

**Recall (Полнота):**

- Это доля правильно предсказанных положительных случаев среди всех реальных положительных случаев.

- Формула: Recall = True Positives / (True Positives + False Negatives).

- Высокий показатель полноты показывает, что модель успешно распознает большую часть реальных положительных случаев.

**F1-Score:**

- Это гармоническое среднее точности и полноты, которое учитывает оба аспекта. F1-Score полезен, когда необходимо сбалансировать точность и полноту.

- Формула: F1-Score = 2 \* (Precision \* Recall) / (Precision + Recall).

- F1-Score близок к 1, когда и точность, и полнота высокие.

**Support:**

- Это метрика, отражающая количество объектов, относящихся к определенному классу, в наборе данных

**Accuracy**:

- Это доля правильно классифицированных случаев вообще.

- Формула: Accuracy = (True Positives + True Negatives) / Total Cases.

- Высокая точность показывает, что модель в целом хороша, но не всегда является лучшим показателем, особенно при несбалансированных классах.

**Macro Average (Макро-среднее):**

- Это среднее значение метрик (точности, полноты и F1) по всем классам, без учета их долей.

- Macro Average хорош для понимания модели в контексте каждого класса, независимо от того, сколько объектов в каждом классе.

**Weighted Average (Взвешенное среднее):**

- Это среднее значение метрик, взвешенное по количеству объектов в каждом классе. Оно учитывает классы с большим количеством примеров.

- Это полезно, когда классы несбалансированы, так как дает более реалистичную оценку общей производительности.

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

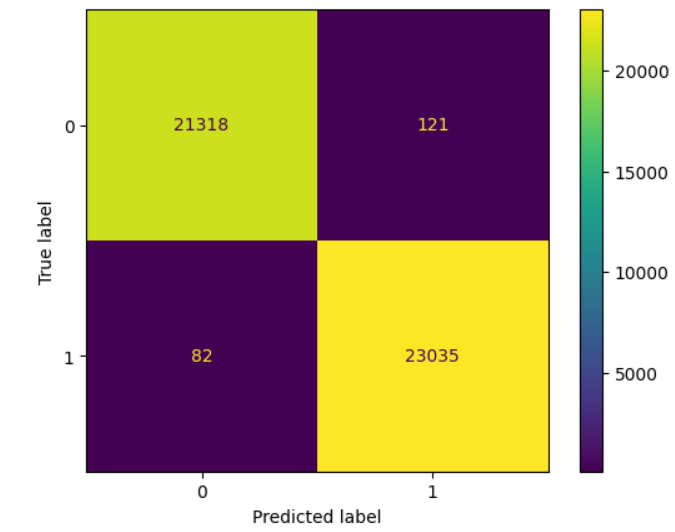


Рисунок 14. Матрица ошибок для модели случайного леса

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 15. Значение accuracy для модели случайного леса

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test = { "Random Forest": f"{round(accuracy, 4) \* 100}%" }

**Обучим модель Gradient Boosting:**

**Градиентный бустинг** — это метод машинного обучения, основанный на ансамблевых методах, который используется для задач регрессии и классификации. Он работает путем последовательного улучшения моделей, добавляя новые слабые модели, которые корректируют ошибки предыдущих. Вот основные шаги, как работает градиентный бустинг:

- Инициализация: Начинается с простого предсказания. Обычно это среднее значение целевой переменной (в случае регрессии) или переданный класс (в случае классификации).

- Расчет ошибок: Для каждой итерации градиентного бустинга вычисляются ошибки (остатки) текущей модели, которые показывают, насколько хорошо или плохо модель предсказывает результаты.

- Обучение новой модели: На основе этих ошибок создается новая слабая модель (например, дерево решений). Она обучается для предсказания оставшихся ошибок, то есть для минимизации разности между предсказаниями и реальными значениями.

- Обновление предсказаний: Новая модель добавляется в ансамбль с некоторым коэффициентом (обычно обозначаемым как обучение rate или шаг). Предсказания обновляются путем суммирования предсказаний всех моделей, включая новую. Это позволяет улучшить качество предсказаний.

- Повторение: Процесс повторяется несколько раз (или до тех пор, пока не достигнуты критерии остановки), где на каждом шаге новая слабая модель добавляется для коррекции ошибок предыдущих моделей.

- Финальное предсказание: В итоге предсказание делается на основе всех слабых моделей, каждая из которых вносит свой вклад в окончательное решение.

#Градиентный бустинг (Gradient Boosting)

from sklearn.ensemble import GradientBoostingClassifier

# Создаем модель градиентного бустинга

gb\_classifier = GradientBoostingClassifier()

# Обучаем модель

gb\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = gb\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

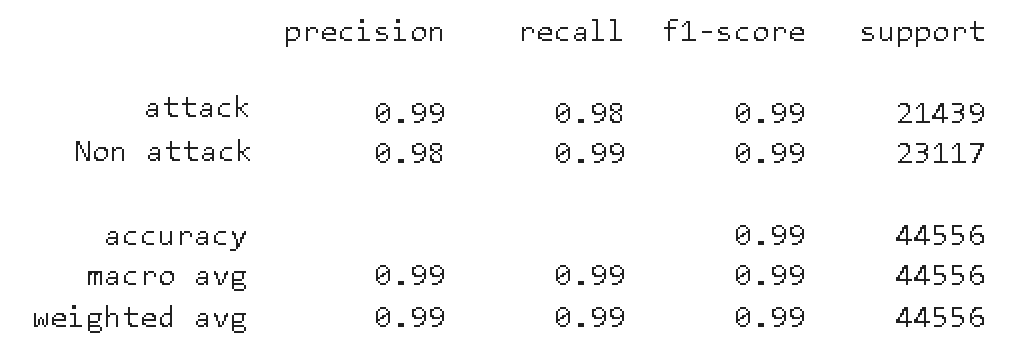


Рисунок 16. Отчет об классификации для модели градиентного бустинга

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

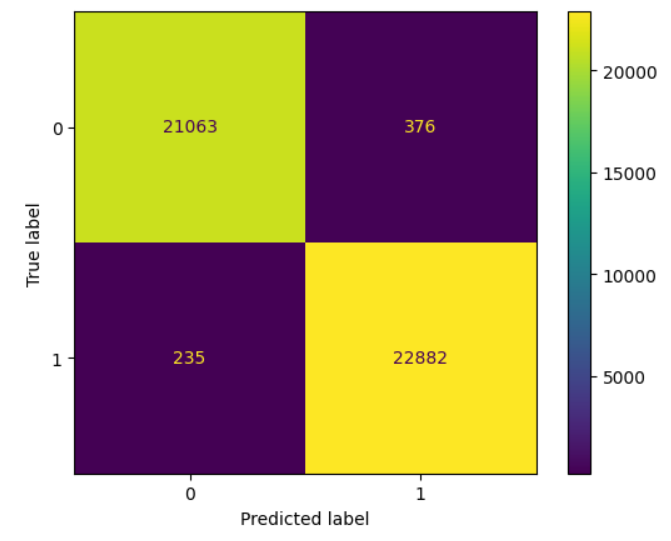


Рисунок 17. Матрица ошибок для модели градиентного бустинга

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 18. Значение accuracy для модели градиентного бустинга

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["Gradient Boosting"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Naive Bayes Classifier:**

**Наивный байесовский классификатор** — это простой и эффективный метод классификации, основанный на теореме Байеса. Он называется "наивным", потому что предполагает, что все признаки (или атрибуты) являются независимыми друг от друга, что на практике часто не соответствует действительности. Вот как он работает:

Обучение модели:

- На этапе обучения классификатор использует тренировочные данные, чтобы оценить вероятности классов и условия для признаков. Для каждого класса (например, категория товара или токсичность текста) рассчитываются:

- Приоритетная вероятность класса (доля экземпляров данного класса в обучающем наборе).

- Условные вероятности признаков для каждого класса (например, вероятность появления слова в документе, принадлежащем к определенному классу).

Расчет вероятности для нового объекта:

- Когда приходит новый объект (например, новый документ), модель вычисляет вероятность принадлежности этого объекта к каждому из классов, используя теорему Байеса:

- P(Class|Features) = P(Features|Class) \* P(Class) / P(Features)

- Здесь P(Class|Features) — это вероятность того, что объект принадлежит классу, учитывая его признаки. P(Features|Class) — это вероятность наблюдать признаки, если объект принадлежит этому классу. P(Class) — это приоритетная вероятность класса, а P(Features) — вероятность наблюдать данные признаки.

Выбор класса:

- Классификатор выбирает класс с наивысшей вероятностью для данного объекта, что и становится его предсказанием

#Наивный байесовский классификатор (Naive Bayes Classifier) from sklearn.naive\_bayes import GaussianNB

# Создаем модель наивного байесовского классификатора nb\_classifier = GaussianNB()

# Обучаем модель

nb\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = nb\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

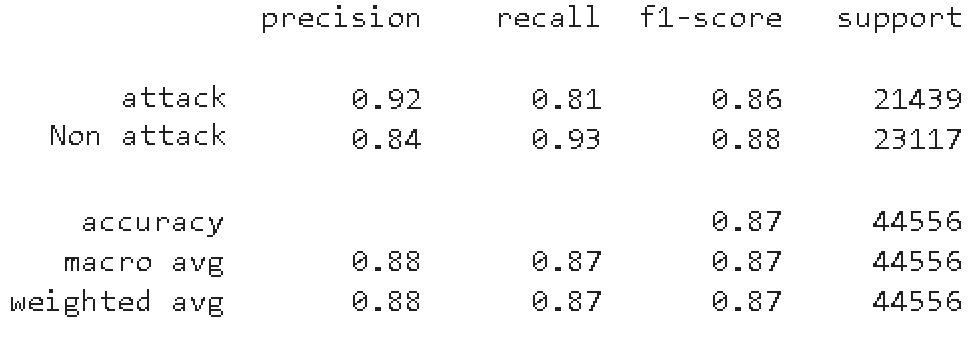


Рисунок 19. Отчет об классификации для модели байесовского классификатора

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

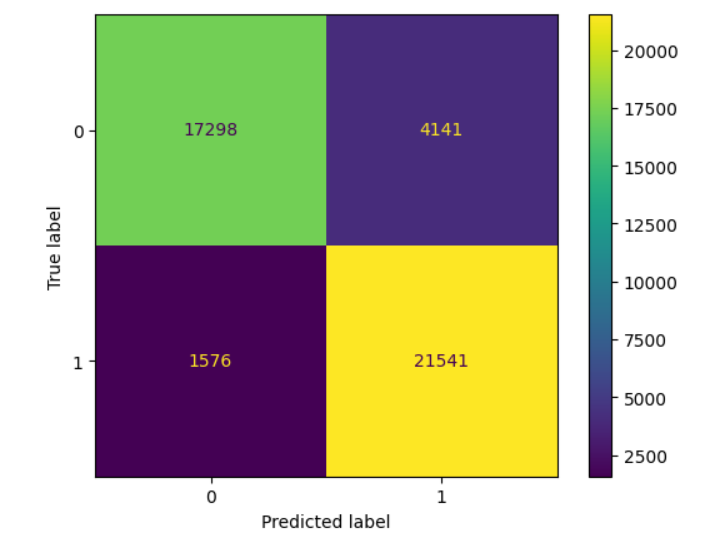


Рисунок 20. Матрица ошибок для модели байесовского классификатора

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 21. Значение accuracy для модели байесовского классификатора

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["Naive Bayes"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Regularized Logistic Regression:**

**Логистическая регрессия** — это метод статистического анализа, используемый для классификации, который предсказывает вероятность принадлежности объекта к определенному классу. Когда мы говорим о логистической регрессии с регуляризацией, мы добавляем дополнительные шаги для предотвращения переобучения.

-Основной принцип логистической регрессии:

Логистическая регрессия моделирует вероятность того, что объект принадлежит к определённому классу, используя логистическую функцию. Эта функция принимает входные данные и преобразует их в значение от 0 до 1, которое интерпретируется как вероятность. Например, если предсказанная вероятность больше 0.5, объект может быть отнесён к положительному классу.

- Построение модели:

При обучении модели логистической регрессии используются данные, чтобы определить коэффициенты (веса) для каждого признака, минимизируя функцию потерь, чаще всего это кросс-энтропия. Однако простая логистическая регрессия может легко переобучаться, особенно на сложных наборах данных с множеством признаков.

- Регуляризация:

Регуляризация включает добавление штрафа за сложность модели, что помогает предотвратить переобучение. Возможно использование различных типов регуляризации:

L1 регуляризация (Lasso): добавляет к функции потерь сумму абсолютных значений коэффициентов. Это может приводить к обнулению некоторых коэффициентов, что позволяет автоматически выбирать важные признаки.

L2 регуляризация (Ridge): добавляет сумму квадратов коэффициентов к функции потерь. Это помогает сгладить значения весов, делая их менее чувствительными к ненадежным данным.

- Оптимизация:

Обучение модели с регуляризацией означает, что мы минимизируем изменённую функцию потерь, которая включает как обычную функцию потерь (кросс-энтропия), так и штраф за регуляризацию. Это приводит к более устойчивой модели.

from sklearn.linear\_model import LogisticRegressionCV

# Создаем модель логистической регрессии

log\_reg\_cv = LogisticRegressionCV()

# Обучаем модель

log\_reg\_cv.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = log\_reg\_cv.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

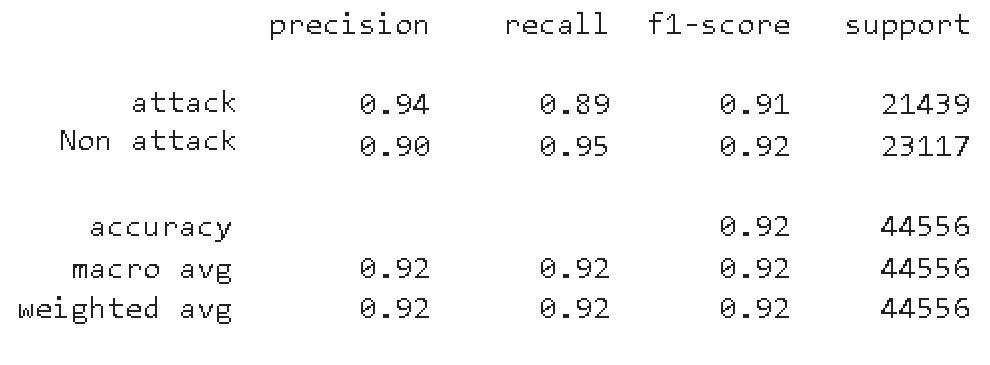


Рисунок 22. Отчет об классификации для модели логистической регрессии с регуляризацией

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

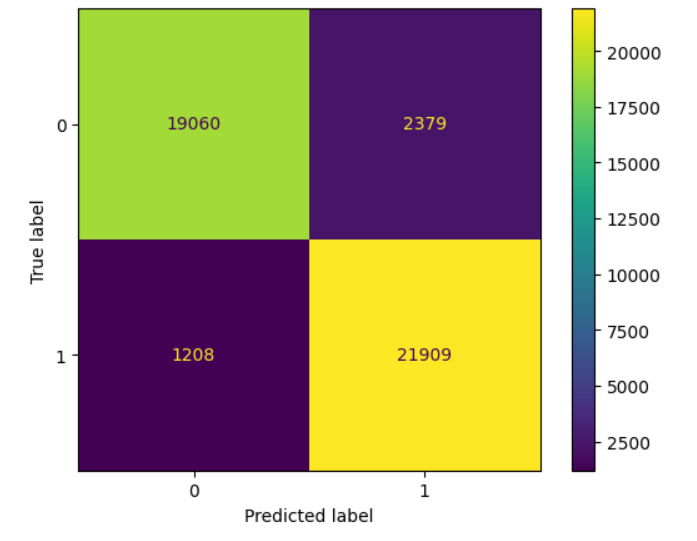


Рисунок 23. Матрица ошибок для модели логистической регрессии с регуляризацией

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 24. Значение accuracy для модели логистической регрессии с регуляризацией

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["LogisticRegressionCV"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Extreme Gradient Boosting (XGBoost):**

**Экстремальный градиентный бустинг (XGBoost)** — это мощный алгоритм машинного обучения, который используется для задач классификации и регрессии. Он основан на концепции градиентного бустинга, но включает в себя несколько оптимизаций и усовершенствований, что делает его одним из лучших алгоритмов для анализа данных.

Одной из ключевых особенностей XGBoost в отличии от обычного градиентного бустинга является использование регуляризации (L1 и L2) по аналогии с линейной регрессией, что помогает предотвратить переобучение модели, сделав её более устойчивой.

Также XGBoost реализует параллельное создание деревьев, что значительно ускоряет процесс обучения по сравнению с традиционными методами бустинга. Это достигается за счет того, что для каждого дерева вычисляются градиенты для всех объектов одновременно.

from xgboost import XGBClassifier

# Создаем модель

XGBoost xgb\_classifier = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

# Обучаем модель

xgb\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = xgb\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

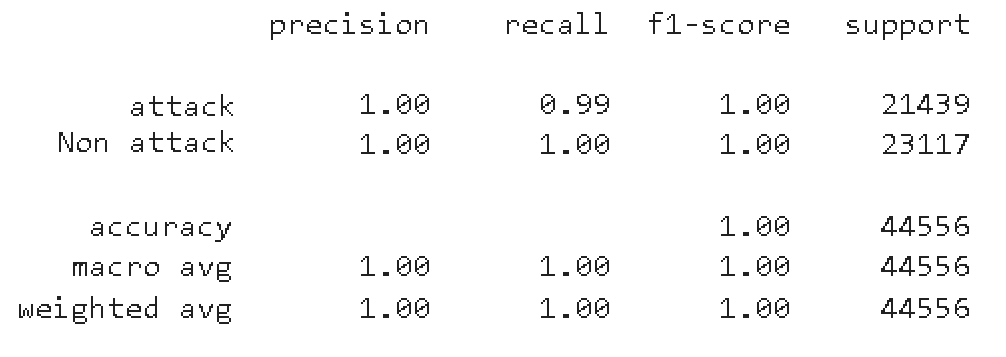


Рисунок 25. Отчет об классификации для модели экстремального градиентного бустинга

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

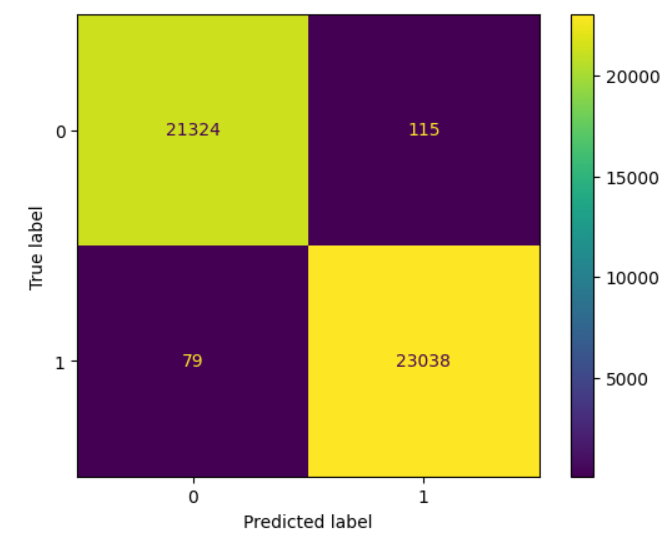


Рисунок 26. Матрица ошибок для модели экстремального градиентного бустинга

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 27. Значение accuracy для модели экстремального градиентного бустинга

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["XGBClassifier"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Decision Tree:**

**Дерево решений** — это простой и интуитивно понятный алгоритм машинного обучения, который используется как для классификации, так и для регрессии. Вот как оно работает:

- Структура дерева:

Дерево решений состоит из узлов, ветвей и листьев. В каждом узле осуществляется проверка условия по одному из признаков, а ветви представляют собой исходы этого условия. Листья дерева содержат конечные результаты (классы или значения).

- Разделение данных:

На каждом узле дерево принимает решение о том, как разделить данные на две или более группы. Это делается на основе определенного критерия, который измеряет, насколько хорошо текущее разделение улучшает прогнозы.

Критерии разделения:

Для задач классификации часто используются такие критерии, как:

Гини (Gini impurity): измеряет степень примеси классов в узле. Чем меньше значение, тем более чистый узел.

Энтропия: оценивает неопределенность узла. Чем выше энтропия, тем меньше чистота узла.

- Построение дерева:

Обучение дерева начинается с корневого узла, который представляет собой весь набор данных. Затем на каждом шаге выбирается признак и порог, на основе которого происходит разделение. Этот процесс повторяется рекурсивно, пока не будет достигнуто заданное число уровней, или пока данные не будут достаточно хорошо разделены.

- Ограничение глубины:

Важно контролировать глубину дерева, чтобы избежать переобучения, когда модель слишком сильно "подстраивается" под обучающие данные и плохо работает на новых данных.

- Предсказание:

Когда дерево построено, чтобы сделать прогноз для нового образца, модель проходит от корня к листьям, принимая решения на каждом узле о том, какой путь выбрать, пока не достигнет листа, где находится предсказанное значение или класс.

#Дерево решений (Decision Tree)

from sklearn.tree import DecisionTreeClassifier

# Создаем модель дерева решений

dt\_classifier = DecisionTreeClassifier()

# Обучаем модель

dt\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = dt\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

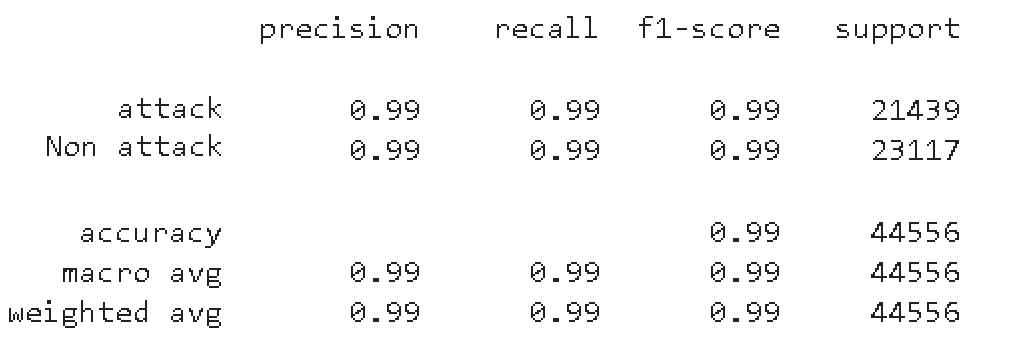


Рисунок 28. Отчет об классификации для модели дерева решений

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

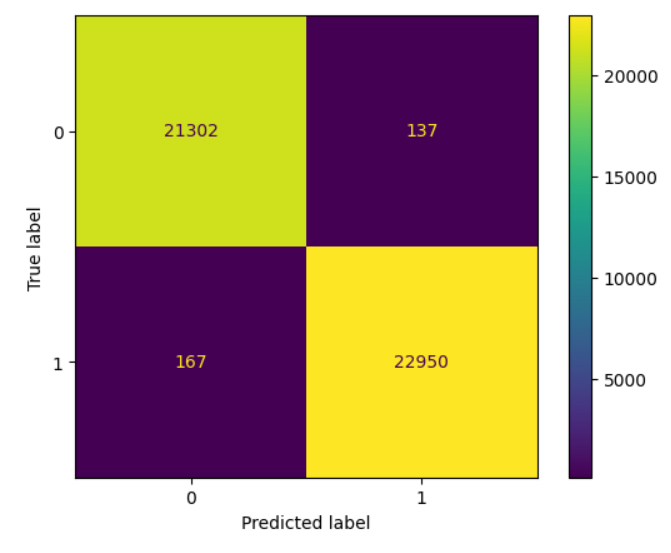


Рисунок 29. Матрица ошибок для модели дерева решений

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 30. Значение accuracy для модели дерева решений

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["DecisionTreeClassifier"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Support Vector Machine:**

Метод опорных векторов (SVM, Support Vector Machine) — это мощный инструмент для решения задач классификации и регрессии. Вот его основные принципы:

- Гиперплоскость:

SVM работает, находя гиперплоскость, которая эффективно разделяет классы в пространстве признаков. Для двумерного пространства это будет линия, а в более высоких измерениях — плоскость.

- Опорные векторы:

Опорные векторы — это данные, которые находятся ближайшими к гиперплоскости. Они определяют положение и ориентацию гиперплоскости. Если взять другие точки, находящиеся далеко от гиперплоскости, они не повлияют на ее положение.

- Максимизация зазора:

SVM стремится максимизировать зазор (расстояние) между гиперплоскостью и ближайшими точками обоих классов (опорными векторами). Это помогает обеспечить лучшую обобщающую способность модели на новых данных.

- Ядровые функции:

В некоторых случаях данные могут быть не линейно разделимыми в исходном пространстве. Для таких ситуаций SVM использует ядровые функции, которые проецируют данные в пространство более высокой размерности, где они могут быть линейно разделимыми. Популярные ядра включают полиномиальные и радиальные (RBF).

- Регуляризация:

Метод включает параметр регуляризации, который контролирует компромисс между максимизацией зазора и минимизацией классификационных ошибок. Это позволяет модели быть более устойчивой к шуму в данных.

- Классификация и прогноз:

Для предсказания класса нового образца SVM определяет, с какой стороны гиперплоскости он находится. Если образец находится с одной стороны, он принадлежит одному классу, если с другой — другому.

#Метод опорных векторов (Support Vector Machine)

from sklearn.svm import SVC

# Создаем модель

SVM svm\_classifier = SVC()

# Обучаем модель

svm\_classifier.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = svm\_classifier.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

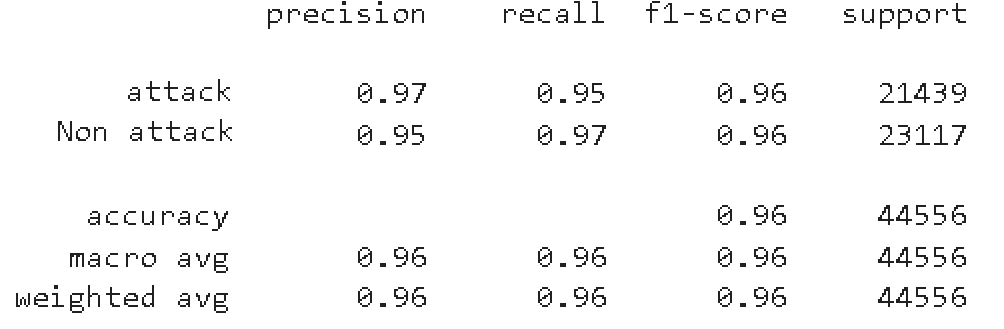


Рисунок 31. Отчет об классификации для модели метода опорных векторов

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

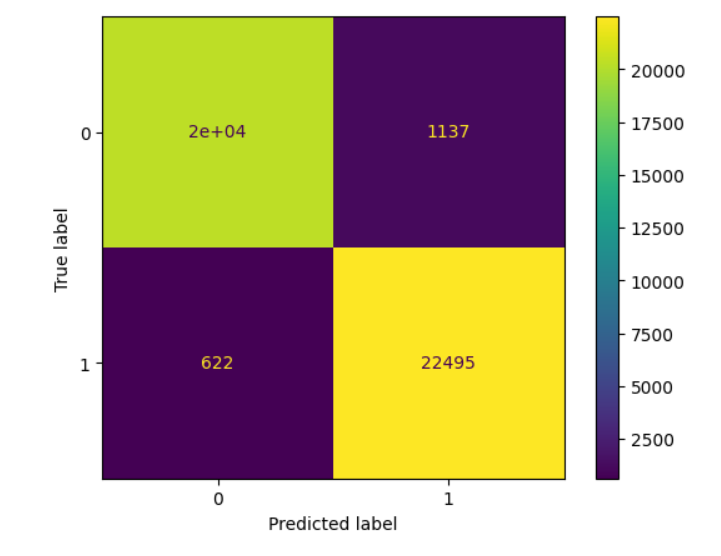


Рисунок 32. Матрица ошибок для модели метода опорных векторов

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 33. Значение accuracy для модели метода опорных векторов

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["SVC"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Logistic Regression:**

#Логистическая регрессия (Logistic Regression)

from sklearn.linear\_model import LogisticRegression

# Создаем модель логистической регрессии

log\_reg = LogisticRegression()

# Обучаем модель

log\_reg.fit(X\_transformed, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = log\_reg.predict(X\_transformed\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Non attack', 'attack']

print(classification\_report(y\_test, prediction, target\_names=labels))

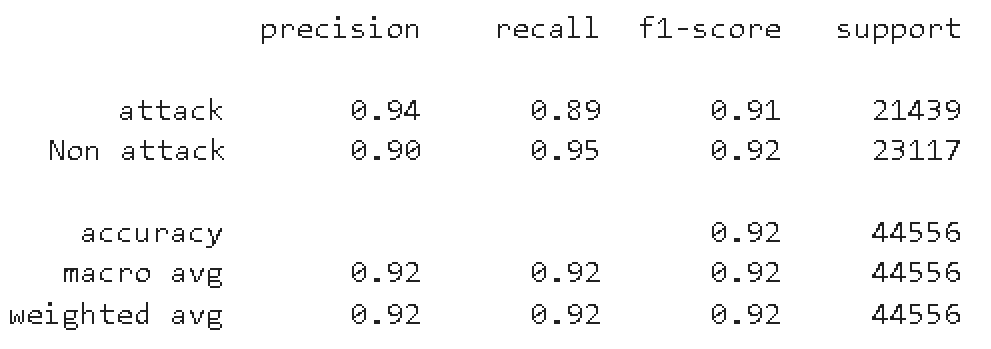


Рисунок 34. Отчет об классификации для модели метода логистической регрессии

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

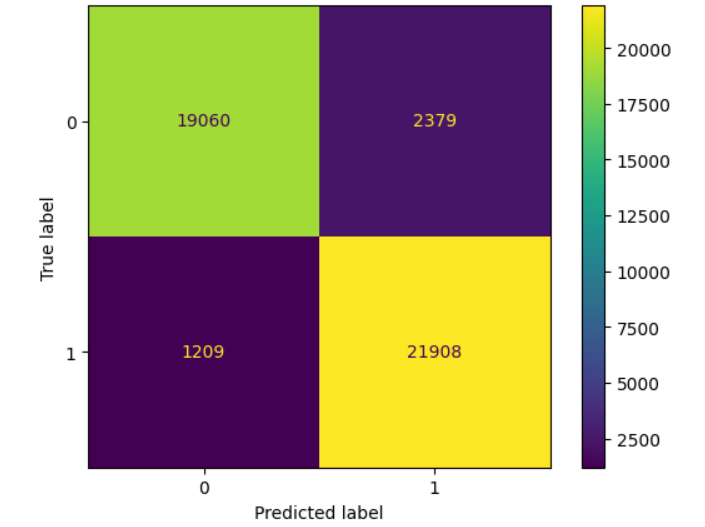


Рисунок 35. Матрица ошибок для модели метода логистической регрессии

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 36. Значение accuracy для модели метода логистической регрессии

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["LogisticRegression"] = f"{round(accuracy, 4) \* 100}%"

**Выведем все значения accuracy:**

for key, value in results\_test.items():

print(f"{key}: {value}")

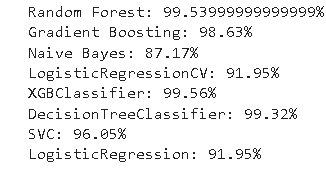


Рисунок 37. Все значения accuracy для бинарной классификации

**Исходя их полученных результатов, можно выделить модель XGBClassifier, которая показала лучшие результаты.**

### Мультиполиномиальная классификация

#### **Этап 1. Подготовка и анализ данных**

**Загрузим датасеты (Train.txt, Test.txt) и объединим их аналогично бинарной классификацией:**

df = pd.read\_csv("Train.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

.

df\_test = pd.read\_csv("Test.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

combined\_df = pd.concat([df, df\_test], ignore\_index=True)

**Выведем и подсчитаем кол-во всех видов атак:**

# Подсчет количества всех типов атак

attack\_counts = combined\_df['attack'].value\_counts()

# Вывод результата

print("Количество атак по типам:") print(attack\_counts)

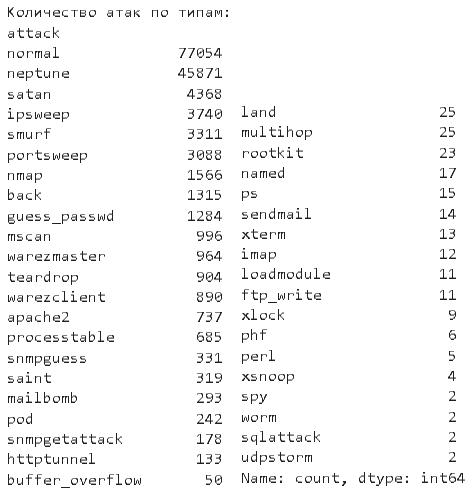


Рисунок 38. Количество атак по типам

Аналогично результату статистического анализа в бинарной классификации, удалим столбцы с преимущественно нулевыми значениями:

combined\_df.drop(['land','urgent','numfailedlogins','numoutboundcmds'],axis=1,inplace=True)

**Разделим все атаки на 4 класса (Ddos, Prob, R2L, U2R).**

1. **DDOS**: Отказ в обслуживании — это категория атак, которая истощает ресурсы жертвы, тем самым делая ее неспособной обрабатывать законные запросы — например, syn flooding. Соответствующие характеристики: «исходные байты» и «процент пакетов с ошибками»

2. **Probing**: Целью слежки и других зондирующих атак является получение информации об удаленной жертве, например, сканирование портов. Соответствующие характеристики: «продолжительность соединения» и «исходные байты»

3. **U2R**: несанкционированный доступ к локальным привилегиям суперпользователя (root) — это тип атаки, при котором злоумышленник использует обычную учетную запись для входа в систему жертвы и пытается получить привилегии root/администратора, эксплуатируя некоторые уязвимости в жертве, например, атаки переполнения буфера. Соответствующие характеристики: «количество созданных файлов» и «количество вызванных приглашений оболочки»,

4. **R2L**: несанкционированный доступ с удаленной машины, злоумышленник проникает на удаленную машину и получает локальный доступ к машине жертвы. Например, соответствующие функции подбора пароля: характеристики уровня сети — «Длительность соединения» и «запрошенная услуга» и характеристики уровня хоста — «количество неудачных попыток входа»

combined\_df['attack'].loc[combined\_df['attack'] ==

'neptune'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'satan'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'ipsweep'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'smurf'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'portsweep'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'nmap'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'back'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'guess\_passwd'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'mscan'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'warezmaster'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'teardrop'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'warezclient'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'apache2'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'processtable'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'snmpguess'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'saint'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'mailbomb'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'pod'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'snmpgetattack'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'httptunnel'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'buffer\_overflow'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'land'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'multihop'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'rootkit'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'named'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'ps'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'sendmail'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'xterm'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'imap'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'loadmodule'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'ftp\_write'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'xlock'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'phf'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'perl'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'xsnoop'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'spy'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'worm'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'sqlattack'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'udpstorm'] = 'Ddos'

**Проверяем, являются ли классы сбалансированными. Для этого выводим количество значений в каждом классе:**

count = combined\_df['attack'].value\_counts()

# Вывод количества значений

print(count)

# Построение гистограммы

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"') plt.xlabel('Тип атаки')

plt.ylabel('Количество') plt.xticks(rotation=0) plt.show()

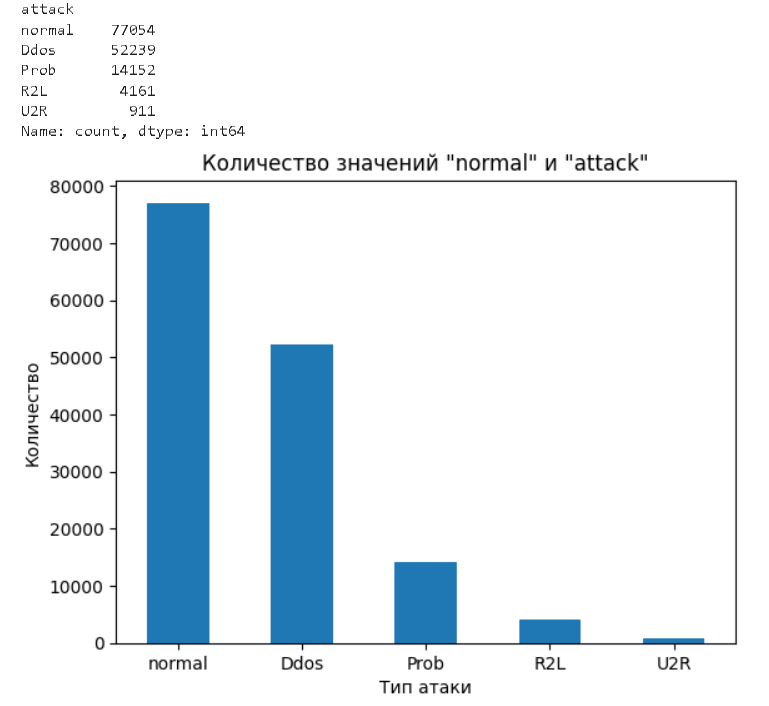


Рисунок 39. Гистограмма распределения классов

Видно, что классы несбалансированны.

**Заменим категориальные данные количественными с помощью LabelEncoder:**

le=LabelEncoder()

# Преобразуем столбец 'protocoltype' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['protocoltype'] = le.fit\_transform(combined\_df['protocoltype'])

# Преобразуем столбец 'service' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['service'] = le.fit\_transform(combined\_df['service'])

# Преобразуем столбец 'flag' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['flag'] = le.fit\_transform(combined\_df['flag'])

# Преобразуем столбец 'attack' в числовые значения с помощью кодировщика LabelEncoder

combined\_df['attack'] = le.fit\_transform(combined\_df['attack'])

**Аналогично корреляционному анализу в бинарной классификации, удалим сильно коррелирующие признаки:**

combined\_df.drop(['numroot','serrorrate','rerrorrate','dsthostrerrorrate', 'srvserrorrate', 'dsthostserrorrate', 'srvrerrorrate', ],axis=1,inplace=True)

**Выделим отдельно целевую переменную для дальнейшего обучения:**

X=combined\_df.drop(['attack'],axis=1)

y=combined\_df['attack']

**Сбалансируем классы с помощью RandomOverSampler**

**RandomOverSampler** – это техника, используемая для борьбы с дисбалансом классов в наборах данных. Когда один класс составляет значительно меньшую долю по сравнению с другими, модели машинного обучения могут обучаться хуже из-за недостатка информации о меньшинстве.

Вот как работает RandomOverSampler:

- Идентификация классов: сначала определяются количество экземпляров в каждом классе в вашем наборе данных. Например, если у вас есть данные о заболеваниях, положительный класс (больные) может состоять из 100 экземпляров, тогда как отрицательный класс (здоровые) — из 900.

- Определение меньшинства: RandomOverSampler определяет, какой класс является меньшинством. В приведенном примере положительный класс будет меньшинством.

- Случайная выборка: затем RandomOverSampler случайным образом дублирует экземпляры из меньшинственного класса. Например, если в нашем примере у нас 100 положительных примеров, RandomOverSampler может случайно выбрать 100 из них и добавить их в набор данных, чтобы сбалансировать с 900 отрицательными.

- Создание нового набора данных: итоговый набор данных теперь содержит равное количество экземпляров из обоих классов – 900 отрицательных и 900 положительных. Это позволяет модели лучше обучаться на меньшинственном классе.

- Обучение модели: Сбалансированный набор данных (с одинаковым числом примеров от обоих классов) используется для обучения модели. Модель имеет больше шансов научиться распознавать характеристики меньшинственного класса.

- Оценка производительности: После обучения модель проверяется на тестовом наборе данных, чтобы оценить ее производительность. Балансировка классов помогает модели более надежно классифицировать положительные примеры.

Таким образом, RandomOverSampler позволяет создать более сбалансированный набор данных, что в конечном итоге улучшает характеристики модели в классификации.

#Применяем метод RandomOverSampler over\_sampling

from imblearn.over\_sampling import RandomOverSampler

''' Этот метод случайным образом выбирает примеры из класса-меньшинства и дублирует их, пока не достигнется баланс.

'''

# Создание экземпляра RandomOverSampler ros = RandomOverSampler()

# Применение увеличения выборки к данным X\_resampled, y\_resampled = ros.fit\_resample(X, y)

**Выводим количество значений в каждом классе:**

balance\_df = pd.concat([X\_resampled, y\_resampled], axis=1) count = balance\_df['attack'].value\_counts()

# Вывод количества значений

print(count)

# Построение гистограммы

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"') plt.xlabel('Тип атаки') plt.ylabel('Количество') plt.xticks(rotation=0)

plt.show()

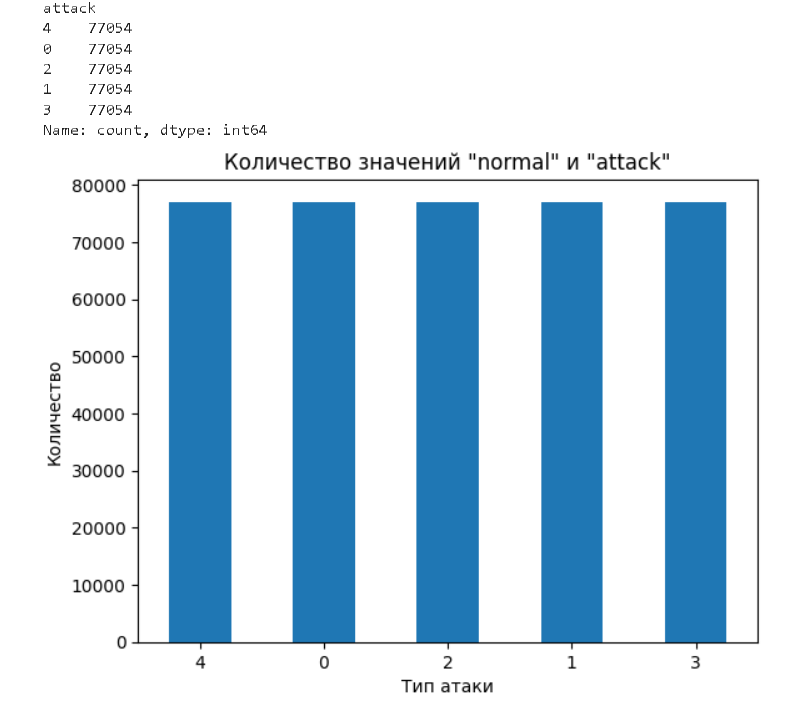


Рисунок 40. Гистограмма распределения классов после балансировки

**Далее проверим важность каждого признака с помощью модели случайного дерева, перед этим нормализовав признаки:**

# Создаем объект StandardScaler

scaler = StandardScaler()

# Обучаем scaler на данных X

scaler.fit(X\_resampled)

# Применяем трансформацию к тренировочным и тестовым данным X\_dop = scaler.transform(X\_resampled)

# Превращаем numpy массивы обратно в DataFrame

X\_dop = pd.DataFrame(X\_dop, columns=X\_resampled.columns)

# Преобразование сбалансированного набора данных в массивы значений

X = X\_dop.values # Признаки

y = y\_resampled.values # Целевая переменная

# Получаем имена признаков из DataFrame

feature\_names = X\_resampled.columns.tolist() # Список имен

столбцов в X\_resampled

# Инициализация модели Random Forest для классификации

model = ensemble.RandomForestClassifier(n\_estimators=100,

criterion="entropy", random\_state=0)

# Обучение модели на подготовленных данных

model.fit(X, y)

# Оценка важности признаков

importances = model.feature\_importances\_ # Оценки важности всех признаков модели

# Создаем DataFrame для хранения важностей признаков

dtf\_importances = pd.DataFrame({

"IMPORTANCE": importances, # Столбец с важностями

"VARIABLE": feature\_names # Столбец с именами признаков

}).sort\_values("IMPORTANCE", ascending=False) # Сортируем по убыванию важности

dtf\_importances['cumsum'] = dtf\_importances['IMPORTANCE'].cumsum(axis=0)

# Устанавливаем имена признаков в качестве индексов

dtf\_importances = dtf\_importances.set\_index("VARIABLE")

## Настройка графика

'''

Формируем надписи для графика

- ticks: местоположения меток по оси x (можно передать пустой список для удаления меток).

- labels: метки для размещения в заданных местах тиков (можно передать только в случае наличия тиков).

- kwargs: текстовые свойства для изменения оформления меток.

Функция Matplotlib xticks() возвращает:

- locs: список местоположений xtick.

- labels: список текстовых объектов xlabel.

plt.grid: Настройка линий сетки.

'''

# Создание графика

fig, ax = plt.subplots(nrows=1, ncols=1, sharex=False, sharey=False, figsize=(10, 5), dpi=100) # Инициализация фигуры и осей

fig.suptitle("Features Importance", fontsize=20) # Заголовок графика

ax.title.set\_text('Variables') # Подпись оси

# Построение горизонтальной столбчатой диаграммы важности признаков

dtf\_importances[["IMPORTANCE"]].sort\_values(by="IMPORTANCE").plot(

kind="barh", # Тип графика - горизонтальная столбчатая диаграмма

legend=False, # Отключаем легенду

ax=ax # Указываем ось для построения

).grid(axis="x") # Настраиваем сетку по оси x

ax.set(ylabel="") # Удаляем метку по оси y, если она не нужна

plt.xticks(rotation=70) # Поворот меток по оси x на 70 градусов для лучшей читаемости

plt.grid(axis='both') # Настройка линий сетки по обеим осям

'''

Отрисовываем график

'''

plt.show() # Показываем график на экране

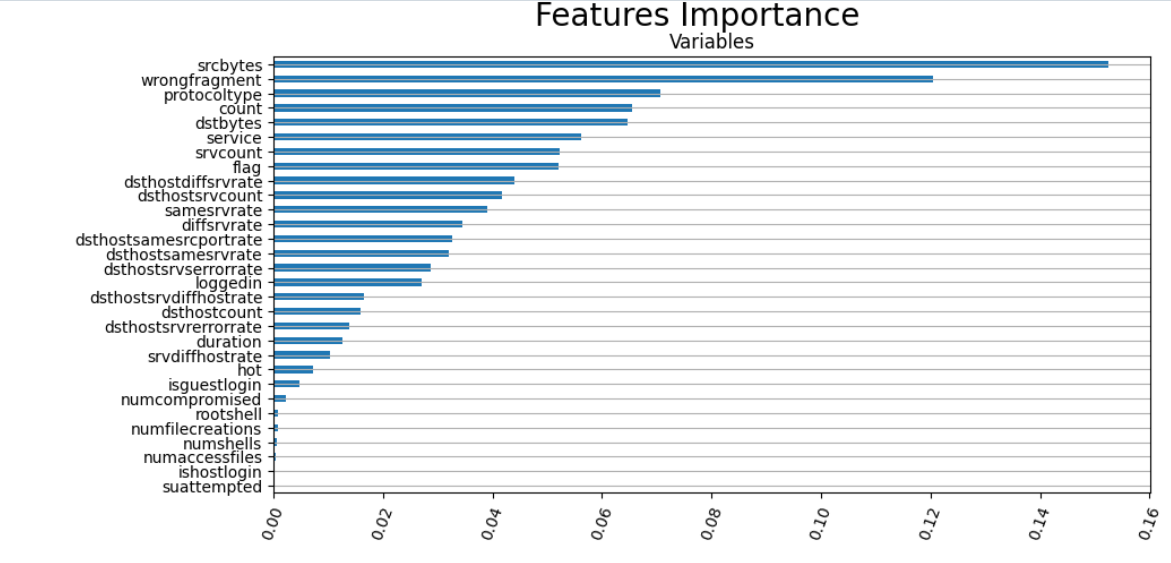


Рисунок 41. График важности признаков

**Удалим наименее важные признаки:**

# Указанные столбцы, которые мы хотим сохранить

columns\_to\_keep = ['srcbytes', 'wrongfragment',

'protocoltype', 'count', 'dstbytes', 'service', 'diffsrvrate', 'flag', 'srvcount', 'samesrvrate', 'dsthostsrvcount', 'dsthostdiffsrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate', 'dsthostsrvserrorrate', 'loggedin', 'dsthostsamesrvrate']

# Удаляем все столбцы, кроме указанных

X\_resampled = X\_resampled[columns\_to\_keep]

**Делим датасет на обучающую и тестовую выборки:**

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

**Нормализуем обучающую и тестовую выборки (в предыдущий раз нормализовались другие выборки для определения важности признаков):**

# Создаем объект StandardScaler, который будет использоваться для стандартизации данных

scaler = StandardScaler()

# Применяем метод fit к данным X, чтобы вычислить среднее и стандартное отклонение,

# необходимые для стандартизации

scaler.fit(X)

# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,

# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)

X\_transformed = scaler.transform(X\_train)

# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,

# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)

X\_transformed\_test = scaler.transform(X\_test)

#### **Этап 2. Моделирование**

**Обучим модель Random Forest:**

*#Случайный лес (Random Forest)*

*# Создаем модель случайного леса*

rf\_classifier = RandomForestClassifier()

*# Обучаем модель*

rf\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = rf\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

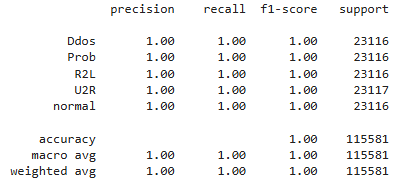
.

Рисунок 42. Отчет об классификации для модели случайного леса

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

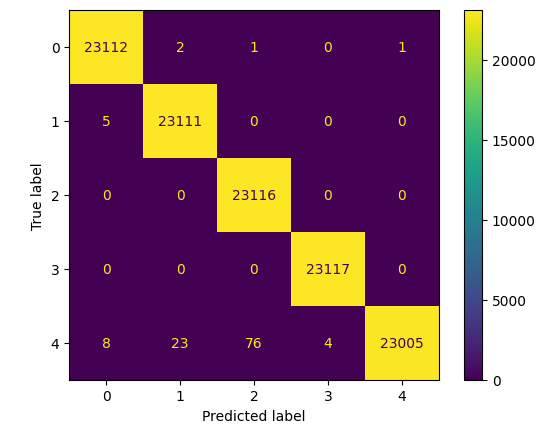
****

Рисунок 43. Матрица ошибок для модели случайного леса

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 44. Значение accuracy для модели случайного леса

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test = { "Random Forest": f"{round(accuracy, 4) \* 100}%" }

**Обучим модель Gradient Boosting:**

#Градиентный бустинг (Gradient Boosting)

from sklearn.ensemble import GradientBoostingClassifier

# Создаем модель градиентного бустинга

gb\_classifier = GradientBoostingClassifier()

# Обучаем модель

gb\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = gb\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

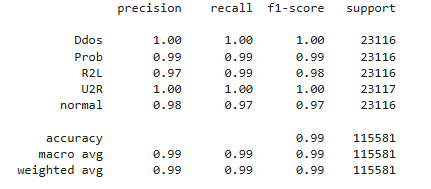
****

Рисунок 45. Отчет об классификации для модели градиентного бустинга

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

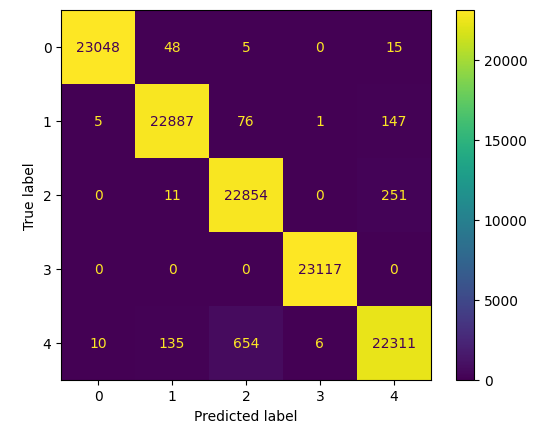
****

Рисунок 46. Матрица ошибок для модели градиентного бустинга

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 47. Значение accuracy для модели градиентного бустинга

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["Gradient Boosting"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Naive Bayes Classifier:**

#Наивный байесовский классификатор (Naive Bayes Classifier) from sklearn.naive\_bayes import GaussianNB

# Создаем модель наивного байесовского классификатора nb\_classifier = GaussianNB()

# Обучаем модель

nb\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = nb\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

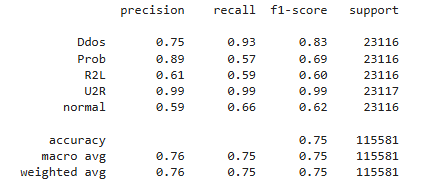
****

Рисунок 48. Отчет об классификации для модели байесовского классификатора

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

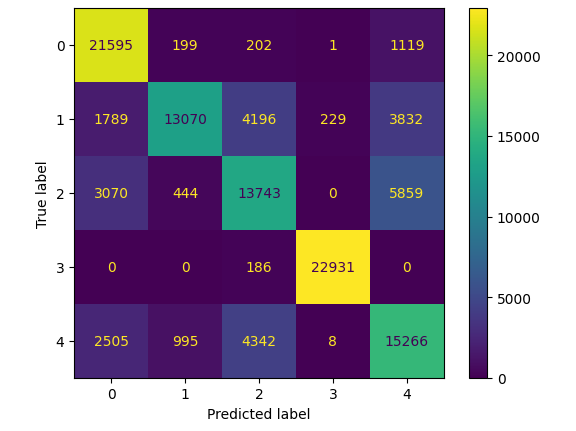
****

Рисунок 49. Матрица ошибок для модели байесовского классификатора

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 50. Значение accuracy для модели байесовского классификатора

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["Naive Bayes"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Regularized Logistic Regression:**

from sklearn.linear\_model import LogisticRegressionCV

# Создаем модель логистической регрессии

log\_reg\_cv = LogisticRegressionCV()

# Обучаем модель

log\_reg\_cv.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = log\_reg\_cv.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

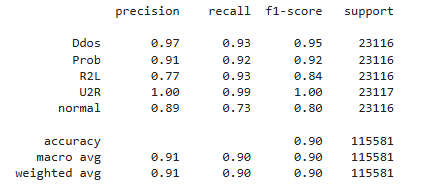
****

Рисунок 51. Отчет об классификации для модели логистической регрессии с регуляризацией

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

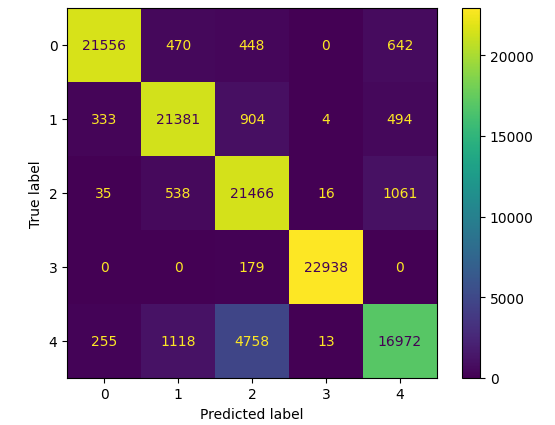
****

Рисунок 52. Матрица ошибок для модели логистической регрессии с регуляризацией

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 53. Значение accuracy для модели логистической регрессии с регуляризацией

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["LogisticRegressionCV"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Extreme Gradient Boosting (XGBoost):**

from xgboost import XGBClassifier

# Создаем модель

XGBoost xgb\_classifier = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

# Обучаем модель

xgb\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = xgb\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

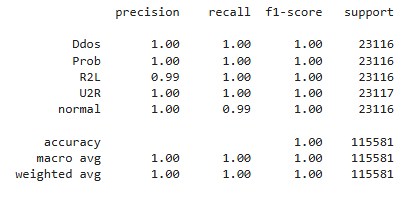
****

Рисунок 54. Отчет об классификации для модели экстремального градиентного бустинга

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

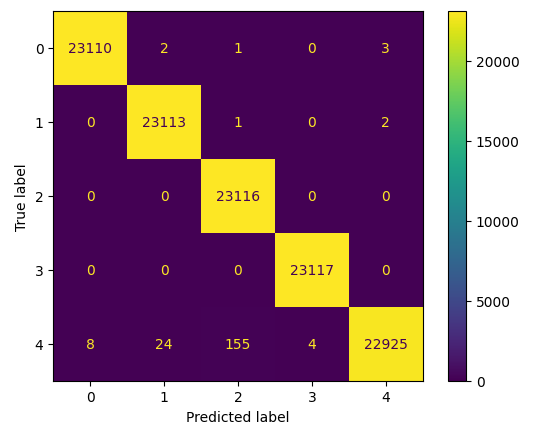
****

Рисунок 55. Матрица ошибок для модели экстремального градиентного бустинга

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction) print(accuracy)



Рисунок 56. Значение accuracy для модели экстремального градиентного бустинга

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["XGBClassifier"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Decision Tree:**

#Дерево решений (Decision Tree)

from sklearn.tree import DecisionTreeClassifier

# Создаем модель дерева решений

dt\_classifier = DecisionTreeClassifier()

# Обучаем модель

dt\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = dt\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

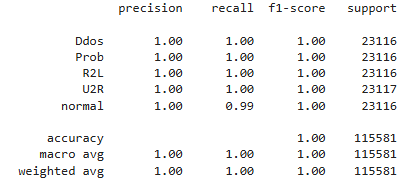
****

Рисунок 57. Отчет об классификации для модели дерева решений

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

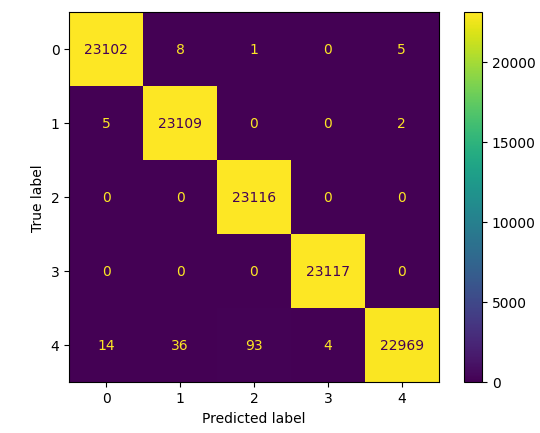
****

Рисунок 58. Матрица ошибок для модели дерева решений

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 59. Значение accuracy для модели дерева решений

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["DecisionTreeClassifier"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Support Vector Machine:**

#Метод опорных векторов (Support Vector Machine)

from sklearn.svm import SVC

# Создаем модель

SVM svm\_classifier = SVC()

# Обучаем модель

svm\_classifier.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = svm\_classifier.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

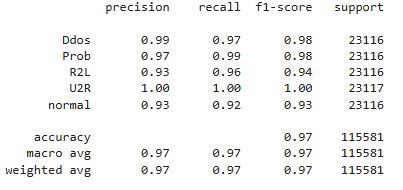


Рисунок 60. Отчет об классификации для модели метода опорных векторов

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

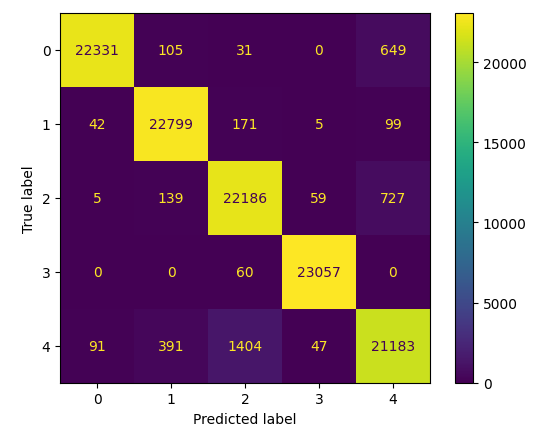
****

Рисунок 61. Матрица ошибок для модели метода опорных векторов

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 62. Значение accuracy для модели метода опорных векторов

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["SVC"] = f"{round(accuracy, 4) \* 100}%"

**Обучим модель Logistic Regression:**

#Логистическая регрессия (Logistic Regression)

from sklearn.linear\_model import LogisticRegression

# Создаем модель логистической регрессии

log\_reg = LogisticRegression()

# Обучаем модель

log\_reg.fit(X\_train, y\_train)

**Предскажем целевую переменную по тестовым данным:**

prediction = log\_reg.predict(X\_test)

**Выведем основные метрики, для проверки качества работы модели классификации:**

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

print(classification\_report(y\_test, prediction, target\_names=labels))

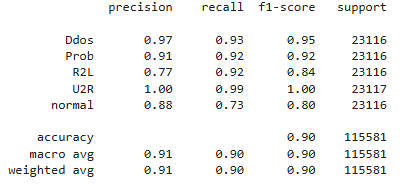
****

Рисунок 63. Отчет об классификации для модели метода логистической регрессии

**Выведем матрицу ошибок:**

cm = confusion\_matrix(y\_test, prediction)

disp = C onfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

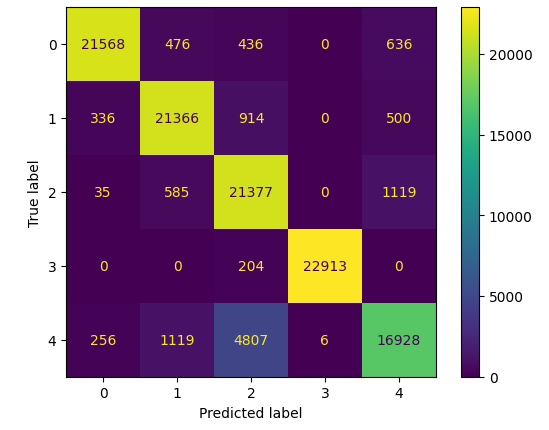
****

Рисунок 64. Матрица ошибок для модели метода логистической регрессии

**Выведем отдельно accuracy:**

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)



Рисунок 65. Значение accuracy для модели метода логистической регрессии

**Запишем значение accuracy в словарь, для дальнейшего анализа:**

results\_test["LogisticRegression"] = f"{round(accuracy, 4) \* 100}%"

**Выведем все значения accuracy:**

for key, value in results\_test.items():

print(f"{key}: {value}")

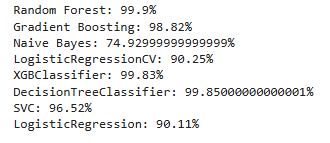


Рисунок 66. Все значения accuracy для бинарной классификации

**Исходя их полученных результатов, можно выделить модель Random Forest, которая показала лучшие результаты.**

# **Заключение**

В ходе выполнения проекта была создана часть системы обнаружения атак в сети, основанная на методах машинного обучения. В процессе работы были проведены эксперименты с различными методами машинного обучения, такими как случайный лес, градиентный бустинг, наивный байесовский классификатор и другие.

Результаты показали, что в случае бинарной классификации наиболее оптимальным методом является XGBClassifier с показателем accuracy = 0,9956. В случае мультиномиальной классификации лучшей моделью оказалась Random Forest со значением accuracy = 0,99896. Эти модели можно использовать в дальнейшем в готовых системах обнаружения атак в сети (IDS).

Таким образом, поставленная цель была достигнута. Был выполнен анализ сетевых соединений и построены модели классификаций, позволяющие отличить нормальное соединение от сетевой атаки на основе имеющихся признаков и делать прогнозы с высокой точностью.

Для достижения цели были решены следующие задачи:

- Разработана модель биномиальной классификации для определения, является ли сетевая активность нормальной или представляет собой атаку.

- Разработана модель мультиномиальной классификации для классификации сетевой активности на несколько категорий: нормальная (не атака), DOS, PROBE, R2L, U2R.

Построенные модели классификации описывают зависимость типа соединения (нормальное или атака) и может применяться для его прогнозирования, что может быть полезно при разработке систем обнаружения атак в сети (IDS).

**Приложения**

Приложение 1

Программный код

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from collections import Counter

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from imblearn.over\_sampling import SMOTE, ADASYN

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

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score, classification\_report, precision\_recall\_curve

import warnings

warnings.filterwarnings("ignore")

df = pd.read\_csv("Train.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

df = df.drop('lastflag', axis = 1)

df

df\_test = pd.read\_csv("Test.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

df\_test = df\_test.drop('lastflag', axis = 1)

df\_test

combined\_df = pd.concat([df, df\_test], ignore\_index=True)

combined\_df.describe() *#вывод статистической информации*

combined\_df.drop(['land','urgent','numfailedlogins','numoutboundcmds'],axis=1,inplace=True)

combined\_df.isnull().values.any() *#Проверка отсутствующих значений NaN*

combined\_df.select\_dtypes(include=['category', 'object']).columns *#вывод категориальных переменных*

combined\_df['attack'].loc[combined\_df['attack']!='normal']='attack' *#заменяем все атаки на одно значение (для бинарной классификации)*

count = combined\_df['attack'].value\_counts()

*# Вывод количества значений*

print(count)

*# Построение гистограммы*

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"')

plt.xlabel('Тип атаки')

plt.ylabel('Количество')

plt.xticks(rotation=0)

plt.show()

percentage\_counts = combined\_df['attack'].value\_counts(normalize=True) \* 100

print("Нормальный трафик:", percentage\_counts.iloc[0], '%')

print("Трафик злоумышленника:", percentage\_counts.iloc[1], '%')

le=LabelEncoder()

*# Преобразуем столбец 'protocoltype' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['protocoltype'] = le.fit\_transform(combined\_df['protocoltype'])

*# Преобразуем столбец 'service' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['service'] = le.fit\_transform(combined\_df['service'])

*# Преобразуем столбец 'flag' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['flag'] = le.fit\_transform(combined\_df['flag'])

*# Преобразуем столбец 'attack' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['attack'] = le.fit\_transform(combined\_df['attack'])

*# Создаем фигуру и ось для отображения матрицы корреляций*

f, ax = plt.subplots(1, 1, figsize=(35, 24))

*# Вычисляем корреляцию между признаками нового сбалансированного DataFrame*

correlation\_matrix = combined\_df.corr()

*# Построим тепловую карту для визуализации матрицы корреляций*

*# Используем палитру 'coolwarm' для улучшения восприятия*

sns.heatmap(correlation\_matrix, annot=True, fmt=".4f", cmap='coolwarm', ax=ax)

*# Устанавливаем заголовок к графику*

ax.set\_title('Матрица корреляций сбалансированной подвыборки', fontsize=14)

*# Отображаем график*

plt.show()

combined\_df.drop(['numroot','serrorrate','rerrorrate','dsthostrerrorrate', 'srvserrorrate', 'dsthostserrorrate', 'srvrerrorrate', ],axis=1,inplace=True)

*# Создаем фигуру и ось для отображения матрицы корреляций*

f, ax = plt.subplots(1, 1, figsize=(35, 24))

*# Вычисляем корреляцию между признаками нового сбалансированного DataFrame*

correlation\_matrix = combined\_df.corr()

*# Построим тепловую карту для визуализации матрицы корреляций*

*# Используем палитру 'coolwarm' для улучшения восприятия*

sns.heatmap(correlation\_matrix, annot=True, fmt=".4f", cmap='coolwarm', ax=ax)

*# Устанавливаем заголовок к графику*

ax.set\_title('Матрица корреляций сбалансированной подвыборки', fontsize=14)

*# Отображаем график*

plt.show()

X=combined\_df.drop(['attack'],axis=1)

y=combined\_df['attack']

from sklearn import ensemble *# Импортируем библиотеку для работы с моделями ансамблей*

'''

Оцениваем важность каждой фичи с помощью RandomForestClassifier

'''

*# Создаем объект StandardScaler*

scaler = StandardScaler()

*# Обучаем scaler на данных X*

scaler.fit(X)

*# Применяем трансформацию к тренировочным и тестовым данным*

X\_dop = scaler.transform(X)

*# Превращаем numpy массивы обратно в DataFrame*

X\_dop = pd.DataFrame(X\_dop, columns=X.columns)

*# Преобразование сбалансированного набора данных в массивы значений*

X\_ = X\_dop.values *# Признаки*

y\_ = y.values *# Целевая переменная*

*# Получаем имена признаков из DataFrame*

feature\_names = X.columns.tolist() *# Список имен столбцов в X*

*# Инициализация модели Random Forest для классификации*

model = ensemble.RandomForestClassifier(n\_estimators=100, *# Количество деревьев в лесу*

criterion="entropy", *# Критерий для оценки качества разбиения*

random\_state=0) *# Задаем seed для воспроизводимости*

*# Обучение модели на подготовленных данных*

model.fit(X\_, y\_)

*# Оценка важности признаков*

importances = model.feature\_importances\_ *# Оценки важности всех признаков модели*

*# Создаем DataFrame для хранения важностей признаков*

dtf\_importances = pd.DataFrame({

"IMPORTANCE": importances, *# Столбец с важностями*

"VARIABLE": feature\_names *# Столбец с именами признаков*

}).sort\_values("IMPORTANCE", ascending=False) *# Сортируем по убыванию важности*

*# Добавляем столбец с кумулятивной суммой важностей*

dtf\_importances['cumsum'] = dtf\_importances['IMPORTANCE'].cumsum(axis=0)

*# Устанавливаем имена признаков в качестве индексов*

dtf\_importances = dtf\_importances.set\_index("VARIABLE")

*## Настройка графика*

'''

Формируем надписи для графика

- ticks: местоположения меток по оси x (можно передать пустой список для удаления меток).

- labels: метки для размещения в заданных местах тиков (можно передать только в случае наличия тиков).

- kwargs: текстовые свойства для изменения оформления меток.

Функция Matplotlib xticks() возвращает:

- locs: список местоположений xtick.

- labels: список текстовых объектов xlabel.

plt.grid: Настройка линий сетки.

'''

*# Создание графика*

fig, ax = plt.subplots(nrows=1, ncols=1, sharex=False, sharey=False, figsize=(10, 5), dpi=100) *# Инициализация фигуры и осей*

fig.suptitle("Features Importance", fontsize=20) *# Заголовок графика*

ax.title.set\_text('Variables') *# Подпись оси*

*# Построение горизонтальной столбчатой диаграммы важности признаков*

dtf\_importances[["IMPORTANCE"]].sort\_values(by="IMPORTANCE").plot(

kind="barh", *# Тип графика - горизонтальная столбчатая диаграмма*

legend=False, *# Отключаем легенду*

ax=ax *# Указываем ось для построения*

).grid(axis="x") *# Настраиваем сетку по оси x*

ax.set(ylabel="") *# Удаляем метку по оси y, если она не нужна*

plt.xticks(rotation=70) *# Поворот меток по оси x на 70 градусов для лучшей читаемости*

plt.grid(axis='both') *# Настройка линий сетки по обеим осям*

'''

Отрисовываем график

'''

plt.show() *# Показываем график на экране*

feature\_names = X.columns.tolist()*# имена столбцов в data.*

print(feature\_names)

*# Указанные столбцы, которые мы хотим сохранить*

columns\_to\_keep = ['srcbytes', 'protocoltype', 'count', 'dstbytes', 'service', 'diffsrvrate', 'flag', 'srvcount', 'samesrvrate', 'dsthostsrvcount', 'dsthostdiffsrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate', 'dsthostsrvserrorrate', 'loggedin', 'dsthostsamesrvrate']

*# Удаляем все столбцы, кроме указанных*

X = X[columns\_to\_keep]

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

*# Создаем объект StandardScaler, который будет использоваться для стандартизации данных*

scaler = StandardScaler()

*# Применяем метод fit к данным X, чтобы вычислить среднее и стандартное отклонение,*

*# необходимые для стандартизации*

scaler.fit(X)

*# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,*

*# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)*

X\_transformed = scaler.transform(X\_train)

*# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,*

*# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)*

X\_transformed\_test = scaler.transform(X\_test)

*#Случайный лес (Random Forest)*

*# Создаем модель случайного леса*

rf\_classifier = RandomForestClassifier()

*# Обучаем модель*

rf\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = rf\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test = {

"Random Forest": f"{round(accuracy, 4) \* 100}%"

}

*#Градиентный бустинг (Gradient Boosting)*

from sklearn.ensemble import GradientBoostingClassifier

*# Создаем модель градиентного бустинга*

gb\_classifier = GradientBoostingClassifier()

*# Обучаем модель*

gb\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = gb\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["Gradient Boosting"] = f"{round(accuracy, 4) \* 100}%"

*#Наивный байесовский классификатор (Naive Bayes Classifier)*

from sklearn.naive\_bayes import GaussianNB

*# Создаем модель наивного байесовского классификатора*

nb\_classifier = GaussianNB()

*# Обучаем модель*

nb\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = nb\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["Naive Bayes"] = f"{round(accuracy, 4) \* 100}%"

*#Логистическая регрессия с регуляризацией (Regularized Logistic Regression)*

from sklearn.linear\_model import LogisticRegressionCV

*# Создаем модель логистической регрессии*

log\_reg\_cv = LogisticRegressionCV()

*# Обучаем модель*

log\_reg\_cv.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = log\_reg\_cv.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["LogisticRegressionCV"] = f"{round(accuracy, 4) \* 100}%"

*#Extreme Gradient Boosting (XGBoost)*

from xgboost import XGBClassifier

*# Создаем модель XGBoost*

xgb\_classifier = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

*# Обучаем модель*

xgb\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = xgb\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["XGBClassifier"] = f"{round(accuracy, 4) \* 100}%"

*#Дерево решений (Decision Tree)*

from sklearn.tree import DecisionTreeClassifier

*# Создаем модель дерева решений*

dt\_classifier = DecisionTreeClassifier()

*# Обучаем модель*

dt\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = dt\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["DecisionTreeClassifier"] = f"{round(accuracy, 4) \* 100}%"

*#Метод опорных векторов (Support Vector Machine)*

from sklearn.svm import SVC

*# Создаем модель SVM*

svm\_classifier = SVC()

*# Обучаем модель*

svm\_classifier.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = svm\_classifier.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["SVC"] = f"{round(accuracy, 4) \* 100}%"

*#Логистическая регрессия (Logistic Regression)*

from sklearn.linear\_model import LogisticRegression

*# Создаем модель логистической регрессии*

log\_reg = LogisticRegression()

*# Обучаем модель*

log\_reg.fit(X\_transformed, y\_train)

labels = ['Non attack', 'attack']

prediction = log\_reg.predict(X\_transformed\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["LogisticRegression"] = f"{round(accuracy, 4) \* 100}%"

for key, value in results\_test.items():

print(f"{key}: {value}")

df = pd.read\_csv("Train.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

df = df.drop('lastflag', axis = 1)

df

df\_test = pd.read\_csv("Test.txt",sep=",",names=["duration","protocoltype","service","flag","srcbytes","dstbytes","land", "wrongfragment","urgent","hot","numfailedlogins","loggedin", "numcompromised","rootshell","suattempted","numroot","numfilecreations", "numshells","numaccessfiles","numoutboundcmds","ishostlogin",

"isguestlogin","count","srvcount","serrorrate", "srvserrorrate",

"rerrorrate","srvrerrorrate","samesrvrate", "diffsrvrate", "srvdiffhostrate","dsthostcount","dsthostsrvcount","dsthostsamesrvrate", "dsthostdiffsrvrate","dsthostsamesrcportrate",

"dsthostsrvdiffhostrate","dsthostserrorrate","dsthostsrvserrorrate",

"dsthostrerrorrate","dsthostsrvrerrorrate","attack", "lastflag"])

df\_test = df\_test.drop('lastflag', axis = 1)

df\_test

combined\_df = pd.concat([df, df\_test], ignore\_index=True)

*# Подсчет количества всех типов атак*

attack\_counts = combined\_df['attack'].value\_counts()

*# Вывод результата*

print("Количество атак по типам:")

print(attack\_counts)

combined\_df.drop(['land','urgent','numfailedlogins','numoutboundcmds'],axis=1,inplace=True)

combined\_df['attack'].loc[combined\_df['attack'] == 'neptune'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'satan'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'ipsweep'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'smurf'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'portsweep'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'nmap'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'back'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'guess\_passwd'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'mscan'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'warezmaster'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'teardrop'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'warezclient'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'apache2'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'processtable'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'snmpguess'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'saint'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'mailbomb'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'pod'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'snmpgetattack'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'httptunnel'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'buffer\_overflow'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'land'] = 'Ddos'

combined\_df['attack'].loc[combined\_df['attack'] == 'multihop'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'rootkit'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'named'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'ps'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'sendmail'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'xterm'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'imap'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'loadmodule'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'ftp\_write'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'xlock'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'phf'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'perl'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'xsnoop'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'spy'] = 'Prob'

combined\_df['attack'].loc[combined\_df['attack'] == 'worm'] = 'U2R'

combined\_df['attack'].loc[combined\_df['attack'] == 'sqlattack'] = 'R2L'

combined\_df['attack'].loc[combined\_df['attack'] == 'udpstorm'] = 'Ddos'

count = combined\_df['attack'].value\_counts()

*# Вывод количества значений*

print(count)

*# Построение гистограммы*

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"')

plt.xlabel('Тип атаки')

plt.ylabel('Количество')

plt.xticks(rotation=0)

plt.show()

le=LabelEncoder()

*# Преобразуем столбец 'protocoltype' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['protocoltype'] = le.fit\_transform(combined\_df['protocoltype'])

*# Преобразуем столбец 'service' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['service'] = le.fit\_transform(combined\_df['service'])

*# Преобразуем столбец 'flag' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['flag'] = le.fit\_transform(combined\_df['flag'])

*# Преобразуем столбец 'attack' в числовые значения с помощью кодировщика LabelEncoder*

combined\_df['attack'] = le.fit\_transform(combined\_df['attack'])

*# Подсчет количества всех типов атак*

attack\_counts = combined\_df['attack'].value\_counts()

*# Вывод результата*

print("Количество атак по типам:")

print(attack\_counts)

combined\_df.drop(['numroot','serrorrate','rerrorrate','dsthostrerrorrate', 'srvserrorrate', 'dsthostserrorrate', 'srvrerrorrate', ],axis=1,inplace=True)

X=combined\_df.drop(['attack'],axis=1)

y=combined\_df['attack']

*#Применяем метод RandomOverSampler over\_sampling*

from imblearn.over\_sampling import RandomOverSampler

'''

Этот метод случайным образом выбирает примеры из класса-меньшинства и дублирует их, пока не достигнется баланс.

'''

*# Создание экземпляра RandomOverSampler*

ros = RandomOverSampler()

*# Применение увеличения выборки к данным*

X\_resampled, y\_resampled = ros.fit\_resample(X, y)

from collections import Counter

Counter(y\_resampled)

balance\_df = pd.concat([X\_resampled, y\_resampled], axis=1)

count = balance\_df['attack'].value\_counts()

*# Вывод количества значений*

print(count)

*# Построение гистограммы*

count.plot(kind='bar')

plt.title('Количество значений "normal" и "attack"')

plt.xlabel('Тип атаки')

plt.ylabel('Количество')

plt.xticks(rotation=0)

plt.show()

from sklearn import ensemble *# Импортируем библиотеку для работы с моделями ансамблей*

'''

Оцениваем важность каждой фичи с помощью RandomForestClassifier

'''

*# Создаем объект StandardScaler*

scaler = StandardScaler()

*# Обучаем scaler на данных X*

scaler.fit(X\_resampled)

*# Применяем трансформацию к тренировочным и тестовым данным*

X\_dop = scaler.transform(X\_resampled)

*# Превращаем numpy массивы обратно в DataFrame*

X\_dop = pd.DataFrame(X\_dop, columns=X\_resampled.columns)

*# Преобразование сбалансированного набора данных в массивы значений*

X = X\_dop.values *# Признаки*

y = y\_resampled.values *# Целевая переменная*

*# Получаем имена признаков из DataFrame*

feature\_names = X\_resampled.columns.tolist() *# Список имен столбцов в X\_resampled*

*# Инициализация модели Random Forest для классификации*

model = ensemble.RandomForestClassifier(n\_estimators=100, *# Количество деревьев в лесу*

criterion="entropy", *# Критерий для оценки качества разбиения*

random\_state=0) *# Задаем seed для воспроизводимости*

*# Обучение модели на подготовленных данных*

model.fit(X, y)

*# Оценка важности признаков*

importances = model.feature\_importances\_ *# Оценки важности всех признаков модели*

*# Создаем DataFrame для хранения важностей признаков*

dtf\_importances = pd.DataFrame({

"IMPORTANCE": importances, *# Столбец с важностями*

"VARIABLE": feature\_names *# Столбец с именами признаков*

}).sort\_values("IMPORTANCE", ascending=False) *# Сортируем по убыванию важности*

*# Добавляем столбец с кумулятивной суммой важностей*

dtf\_importances['cumsum'] = dtf\_importances['IMPORTANCE'].cumsum(axis=0)

*# Устанавливаем имена признаков в качестве индексов*

dtf\_importances = dtf\_importances.set\_index("VARIABLE")

*## Настройка графика*

'''

Формируем надписи для графика

- ticks: местоположения меток по оси x (можно передать пустой список для удаления меток).

- labels: метки для размещения в заданных местах тиков (можно передать только в случае наличия тиков).

- kwargs: текстовые свойства для изменения оформления меток.

Функция Matplotlib xticks() возвращает:

- locs: список местоположений xtick.

- labels: список текстовых объектов xlabel.

plt.grid: Настройка линий сетки.

'''

*# Создание графика*

fig, ax = plt.subplots(nrows=1, ncols=1, sharex=False, sharey=False, figsize=(10, 5), dpi=100) *# Инициализация фигуры и осей*

fig.suptitle("Features Importance", fontsize=20) *# Заголовок графика*

ax.title.set\_text('Variables') *# Подпись оси*

*# Построение горизонтальной столбчатой диаграммы важности признаков*

dtf\_importances[["IMPORTANCE"]].sort\_values(by="IMPORTANCE").plot(

kind="barh", *# Тип графика - горизонтальная столбчатая диаграмма*

legend=False, *# Отключаем легенду*

ax=ax *# Указываем ось для построения*

).grid(axis="x") *# Настраиваем сетку по оси x*

ax.set(ylabel="") *# Удаляем метку по оси y, если она не нужна*

plt.xticks(rotation=70) *# Поворот меток по оси x на 70 градусов для лучшей читаемости*

plt.grid(axis='both') *# Настройка линий сетки по обеим осям*

'''

Отрисовываем график

'''

plt.show() *# Показываем график на экране*

feature\_names = X\_resampled.columns.tolist()*# имена столбцов в data.*

print(feature\_names)

*# Указанные столбцы, которые мы хотим сохранить*

columns\_to\_keep = ['srcbytes', 'wrongfragment', 'protocoltype', 'count', 'dstbytes', 'service', 'diffsrvrate', 'flag', 'srvcount', 'samesrvrate', 'dsthostsrvcount', 'dsthostdiffsrvrate', 'dsthostdiffsrvrate', 'dsthostsamesrcportrate', 'dsthostsrvserrorrate', 'loggedin', 'dsthostsamesrvrate']

*# Удаляем все столбцы, кроме указанных*

X\_resampled = X\_resampled[columns\_to\_keep]

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

*# Создаем объект StandardScaler, который будет использоваться для стандартизации данных*

scaler = StandardScaler()

*# Применяем метод fit к данным X, чтобы вычислить среднее и стандартное отклонение,*

*# необходимые для стандартизации*

scaler.fit(X\_resampled)

*# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,*

*# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)*

X\_train = scaler.transform(X\_train)

*# Применяем метод transform к данным X, чтобы преобразовать их на основе вычисленных значений,*

*# то есть стандартизируем данные (вычитаем среднее и делим на стандартное отклонение)*

X\_test = scaler.transform(X\_test)

*#Случайный лес (Random Forest)*

*# Создаем модель случайного леса*

rf\_classifier = RandomForestClassifier()

*# Обучаем модель*

rf\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = rf\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test = {

"Random Forest": f"{round(accuracy, 4) \* 100}%"

}

*#Градиентный бустинг (Gradient Boosting)*

from sklearn.ensemble import GradientBoostingClassifier

*# Создаем модель градиентного бустинга*

gb\_classifier = GradientBoostingClassifier()

*# Обучаем модель*

gb\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = gb\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["Gradient Boosting"] = f"{round(accuracy, 4) \* 100}%"

*#Наивный байесовский классификатор (Naive Bayes Classifier)*

from sklearn.naive\_bayes import GaussianNB

*# Создаем модель наивного байесовского классификатора*

nb\_classifier = GaussianNB()

*# Обучаем модель*

nb\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = nb\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["Naive Bayes"] = f"{round(accuracy, 4) \* 100}%"

*#Логистическая регрессия с регуляризацией (Regularized Logistic Regression)*

from sklearn.linear\_model import LogisticRegressionCV

*# Создаем модель логистической регрессии*

log\_reg\_cv = LogisticRegressionCV()

*# Обучаем модель*

log\_reg\_cv.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = log\_reg\_cv.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["LogisticRegressionCV"] = f"{round(accuracy, 4) \* 100}%"

*#Extreme Gradient Boosting (XGBoost)*

from xgboost import XGBClassifier

*# Создаем модель XGBoost*

xgb\_classifier = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')

*# Обучаем модель*

xgb\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = xgb\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["XGBClassifier"] = f"{round(accuracy, 4) \* 100}%"

*#Дерево решений (Decision Tree)*

from sklearn.tree import DecisionTreeClassifier

*# Создаем модель дерева решений*

dt\_classifier = DecisionTreeClassifier()

*# Обучаем модель*

dt\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = dt\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["DecisionTreeClassifier"] = f"{round(accuracy, 4) \* 100}%"

*#Метод опорных векторов (Support Vector Machine)*

from sklearn.svm import SVC

*# Создаем модель SVM*

svm\_classifier = SVC()

*# Обучаем модель*

svm\_classifier.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = svm\_classifier.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["SVC"] = f"{round(accuracy, 4) \* 100}%"

*#Логистическая регрессия (Logistic Regression)*

from sklearn.linear\_model import LogisticRegression

*# Создаем модель логистической регрессии*

log\_reg = LogisticRegression()

*# Обучаем модель*

log\_reg.fit(X\_train, y\_train)

labels = ['Ddos', 'Prob', 'R2L', 'U2R', 'normal']

prediction = log\_reg.predict(X\_test)

print(classification\_report(y\_test, prediction, target\_names=labels))

cm = confusion\_matrix(y\_test, prediction)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot()

accuracy = accuracy\_score(y\_test, prediction)

print(accuracy)

results\_test["LogisticRegression"] = f"{round(accuracy, 4) \* 100}%"

for key, value in results\_test.items():

print(f"{key}: {value}")