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 и машинное обучение»

**“****Trojan Detection”**

Разработчики проекта (ФИО):

Меньшиков Александр Олегович

Шарапов Дмитрий Андреевич

Пермь, 2024

**ПАСПОРТ ПРОЕКТА**

**Название проекта:**

“Trojan Detection”

**Сведения об авторах:**

Меньшиков Александр Олегович – студент 4 курса, группа БАС-21

Шарапов Дмитрий Андреевич – студент 4 курса, группа БАС-21

**Цель и задачи проекта:**

**Цель:**

1. Провести анализ данных и на основе сетевых характеристик программ выявить среди них зараженный трафик (определить троян на этапе сетевой передачи данных).

**Задачи:**

1. Провести анализ проблемы и обосновать важность системы анализа трафика
2. Подготовить данные (Нормализация, Очистка выбросов, Балансировка классов, Выделение наиболее важных признаков)
3. Обучение и анализ метрик моделей для выбора наиболее эффективной из них
4. Выбор лучшей модели
5. Оценка лучшей модели

**Краткое описание проекта:**

Требуется проанализировать данные с сетевыми характеристиками и определить, возможно ли выявить троян на этапе сетевой передачи. Визуализировать и проанализировать результаты. Сделать выводы.

**Конкретные ожидаемые результаты:**

Обученная модель-классификатор троянского трафика и рекомендации по ее использованию.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

Троянский вирус (троян, троянец) – вид вредоносного ПО, которое маскируется под легитимное приложение для того, чтобы путём обмана пользователя быть загруженным на компьютер. Под видом обычного вложения к электронному письму или файла, скачанного из Интернета, троянец активируется, когда пользователь запускает его на своем устройстве. Такой вирус содержит в себе вредоносную программу, которая инициирует вредоносные действия.

Трояны могут значительно угрожать как личным финансам пользователей, так и финансовым активам компаний по нескольким причинам:

**1. Кража личных данных**: Трояны могут собирать конфиденциальную информацию, такую как пароли и данные банковских карт. Эта информация может быть использована злоумышленниками для несанкционированного доступа к банковским счетам, что ведет к прямым финансовым потерям.

**2. Шифрование данных**: Некоторые трояны работают как программы-вымогатели, шифруя важные файлы и требуя выкуп за их восстановление. Это может привести как к потерям для индивидуальных пользователей, так и к серьезным финансовым расходам для компаний, которые могут потерять доступ к критически важной информации.

**3. Ущерб репутации**: В случае утечки данных, компании могут понести убытки не только из-за фактической потери средств, но и из-за ухудшения репутации. Клиенты могут потерять доверие к компании, что также может привести к снижению доходов.

**4. Отрицательные воздействия на действия бизнеса**: Трояны могут отслеживать действия сотрудников, что может повлиять на производительность и эффективность работы компании. Если сотрудники будут отвлечены на решение вопросов безопасности, это может привести к экономическим потерям.

**5. Неоплаченные выставления счетов**: Переходя на более серьезные аспекты, злоумышленники могут использовать устройство как часть ботнета для проведения DDoS-атак или других мошеннических действий, что может обернуться для компаний значительными штрафами и убытками.

Таким образом, трояны представляют собой серьезную угрозу для финансов как отдельных пользователей, так и организаций, что подчеркивает важность защиты от вредоносного программного обеспечения.

**Реальные примеры атак с использованием троянских вирусов**

**NetSupport RAT**: нацеленный на образовательный сектор, госучреждения и предприятия сферы услуг, троянец использует для распространения множество разных методов: от ложных обновлений и автоматических загрузок при просмотре веб-сайтов до вредоносных загрузчиков и фишинговых кампаний.

**Банковский троян Chameleon**: Android-троян, нацеленный преимущественно на европейских пользователей банковских приложений. Главная особенность – полный захват устройства через службу специальных возможностей Android. Так хакеры могут выполнять любые несанкционированные действия от имени жертвы незаметно для нее.

**Zeus (Zbot):** троян предназначен для всех версий Windows, и из-за своей структуры, позволяющей работать без подключения программы к драйверам - может заразить компьютер даже из гостевой учетной записи. После попадания в систему, программа перехватывает регистрационные данные пользователя и способна переводить денежные средства на счета других жертв небольшими суммами, что делает невозможным поиски и отслеживание ресурса хакеров.

Ключевым подходом проекта является построение системы, которая позволяет выявлять вредоносный трафик среди всего трафика, если таковое вообще возможно, исходя из множества данных нам сетевых характеристик (гипотеза – исходя из сетевых характеристик получится выявить троянский трафик).

*Цель:* Выполнить анализ данных с сетевыми характеристиками и разработать анализатор троянского трафика, который в дальнейшем может использоваться в антивирусном ПО.

*Задачи:*

1. Провести анализ проблемы и обосновать актуальность анализатора троянского трафика.

2. Подготовить данные: очистка, нормализация, балансировка и отбор ключевых признаков.

3. Провести обучение моделей и оценить их эффективность.

4. Найти лучшую модель, проанализировать ее результаты.

**Исходные данные**

Данные взяты с

<https://www.kaggle.com/datasets/subhajournal/trojan-detection/data>

О наборе данных:

Набор данных содержит вредоносные (категория Trojan) и доброкачественные сетевые характеристики, которые могут быть использованы для обнаружения троянского коня.

В исходных данных есть 84 уникальных столбца, посмотреть их все можно по ссылке, ниже описаны только те, которые остались после подготовки данных:

1. **IP-адрес источника**: Уникальный адрес устройства, отправляющего данные в сети. Он идентифицирует отправителя пакета.

2. **IP-адрес назначения:** Уникальный адрес устройства, которое получает данные в сети. Он указывает, куда должны быть отправлены пакеты.

3. **Порт источника:** Номер порта на устройстве-отправителе, который используется для обозначения конкретного приложения или сервисов, отправляющих данные.

4. **InitWinbytesbackward**: Количество байт, доступных для отправки в потоке данных, рассматриваемое в обратном направлении (от назначения к источнику) в начале соединения.

5. **Длительность потока**: Время, в течение которого поток данных активен, начиная с момента его инициализации до завершения передачи данных.

6. **Среднее значение потока**: Средняя величина объема данных, передаваемых в потоке за единицу времени, которая позволяет оценить интенсивность передачи данных.

7. **Количество пакетов/с**: Общая величина пакетов, переданных в сети за одну секунду. Это значение используется для оценки скорости передачи данных.

8. **Максимальное количество пакетов/с:** Наивысшее значение пакетов, переданных в сети за секунду, что может свидетельствовать о пиковых нагрузках на сеть.

9**. Количество пакетов/с (Min)** Минимальное значение пакетов, переданных за секунду, которое позволяет определить низшие уровни активности потока данных.

10. **InitWinbytesforward:** Количество байт, доступных для отправки в потоке данных, рассматриваемое в прямом направлении (от источника к назначению) в начале соединения.

11**. Порт назначения:** Номер порта на устройстве-получателе, который используется для обозначения конкретного приложения или сервиса, принимающего данные.

12. **Количество флагов URG:** Количество флагов, установленных в заголовке TCP-пакета, указывающих на то, что участок данных является срочным и требует немедленной обработки.

13. **Длина заголовка Bwd:** Общая длина заголовков пакетов, передаваемых в обратном направлении (от назначения к источнику) в потоке данных, что может влиять на обработку и анализ данных.

**Реализация проекта**

**Этап 1. Подготовка среды выполения кода и просмотр данных для обучения**

Импортируем используемые библиотеки

|  |
| --- |
| import numpy as np  import pandas as pd  import seaborn as sns  sns.set\_palette('husl')  import matplotlib.pyplot as plt  %matplotlib inline  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder  from sklearn.preprocessing import StandardScaler  from sklearn import model\_selection, preprocessing, feature\_selection, ensemble, linear\_model, metrics, decomposition  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import cross\_val\_score  from sklearn.model\_selection import StratifiedKFold  from sklearn.metrics import classification\_report  from sklearn.metrics import accuracy\_score  from sklearn.model\_selection import GridSearchCV  from sklearn.linear\_model import LogisticRegression  from sklearn.svm import SVC  from sklearn.neighbors import KNeighborsClassifier  from sklearn.tree import DecisionTreeClassifier  from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier  from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  from sklearn.naive\_bayes import GaussianNB |

Чтение датасета из файла

|  |
| --- |
| dataset = pd.read\_csv('Trojan\_Detection.csv') |

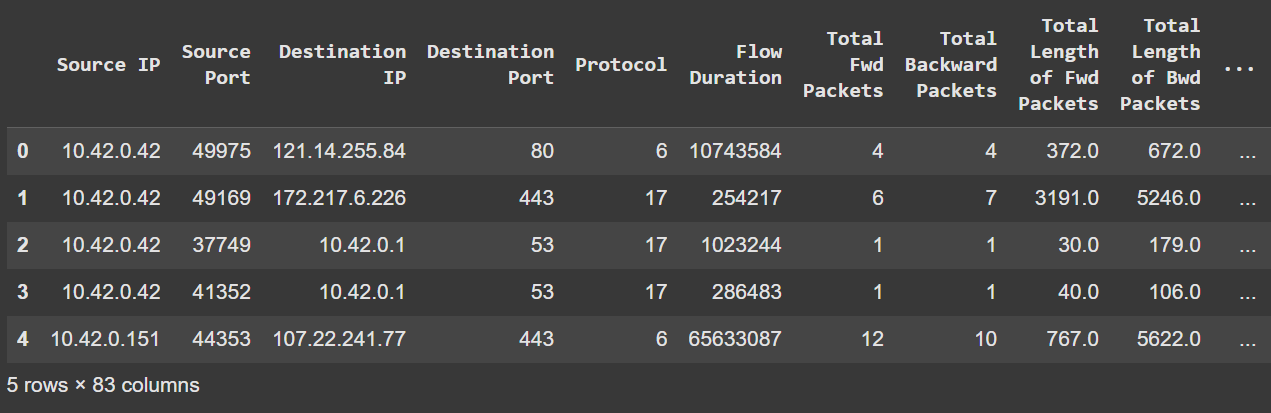
Сразу удаляем ненужные фичи, которые ни на что не влияют

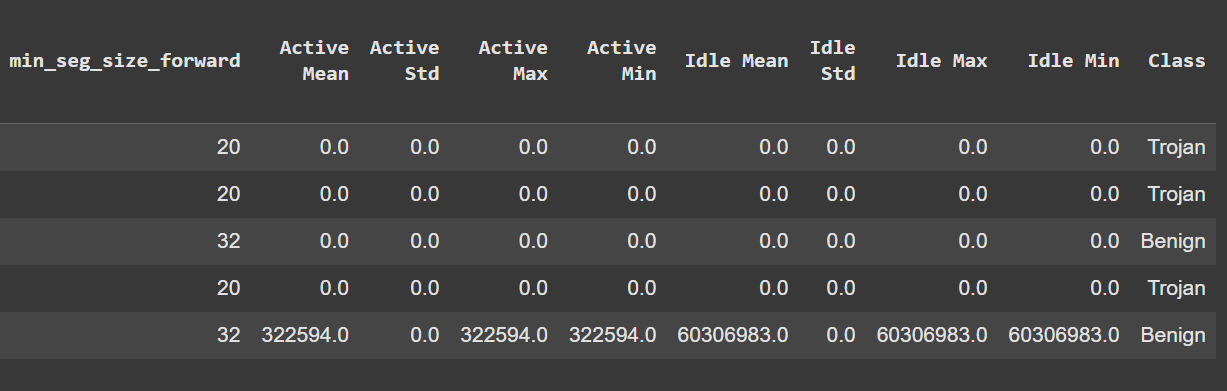
“Идентификатор потока” и “Временная метка” так как они не относятся к параметрам сети (по сути являются уникальными на протяжении всего датасета).

|  |
| --- |
| # Удаляем ненужные фичи  dataset = dataset.drop(['Unnamed: 0', 'Flow ID', ' Timestamp'], axis=1) |

Смотрим как выглядит наш датасет

|  |
| --- |
| dataset.head() |





**Этап 2. Подготовка данных**

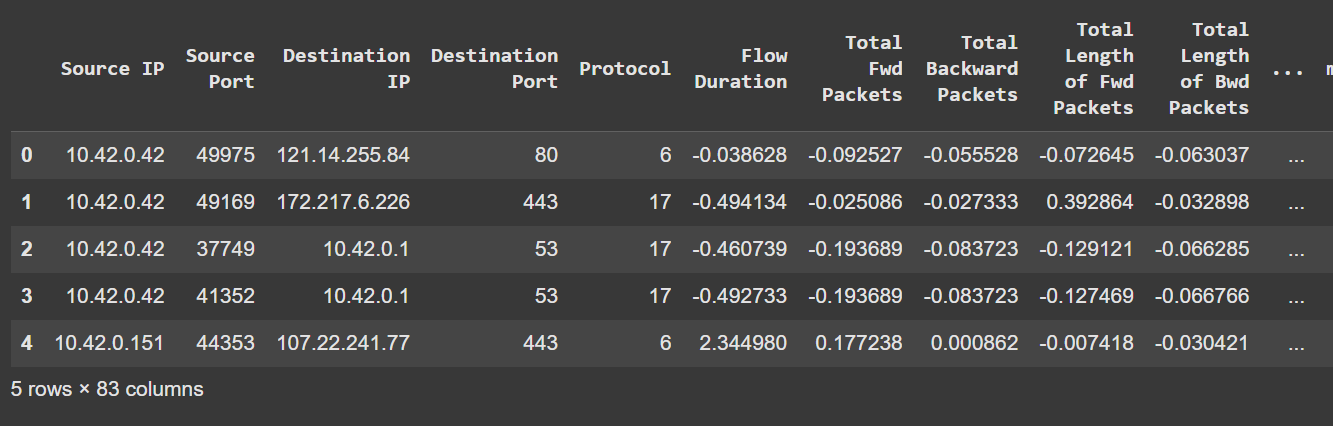
**Нормализация**

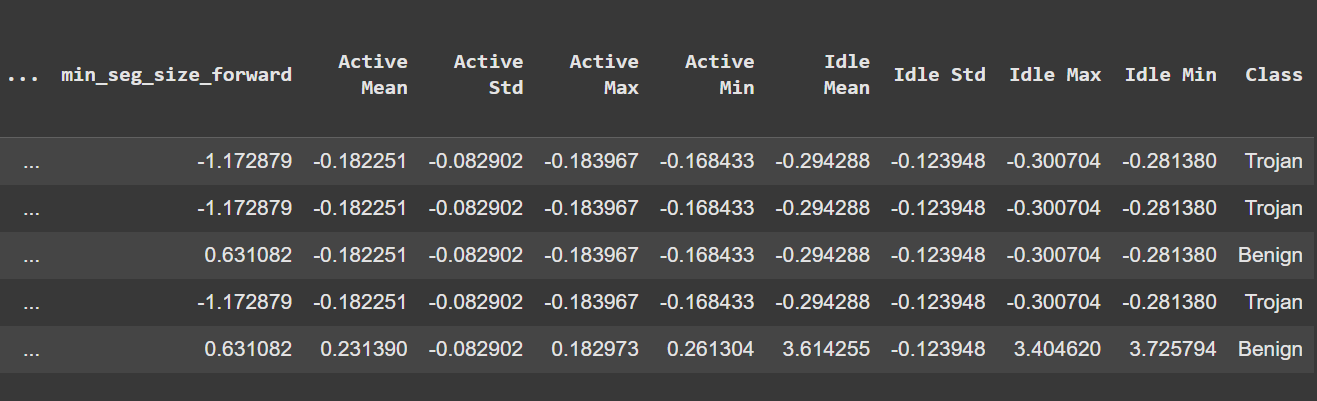
Определяем фичи для нормализации

|  |
| --- |
| # Сохраняем только количественные фичи. Категориальные кодируем в числовые путем переопределения в метки (LabelEncoder)  # Целевую переменную в нормализации не рассматриваем  cols\_for\_normalization = [ ' Bwd PSH Flags', ' Fwd URG Flags', ' Bwd URG Flags', ' RST Flag Count',  ' CWE Flag Count', ' ECE Flag Count', 'Fwd Avg Bytes/Bulk',  ' Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk',  ' Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate', ' act\_data\_pkt\_fwd',  ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets',  'Total Length of Fwd Packets',' Total Length of Bwd Packets',  ' Fwd Packet Length Max', ' Fwd Packet Length Min',  ' Fwd Packet Length Mean', ' Fwd Packet Length Std',  'Bwd Packet Length Max', ' Bwd Packet Length Min',  ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s',  ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max',  ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std',  ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean',  ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min',  ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s',  ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length',  ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance',  ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size',  ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets',  ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes',  'Init\_Win\_bytes\_forward', ' Init\_Win\_bytes\_backward',  ' min\_seg\_size\_forward', 'Active Mean',  ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std',  ' Idle Max', ' Idle Min'] |

Нормализация определенных ранее фич

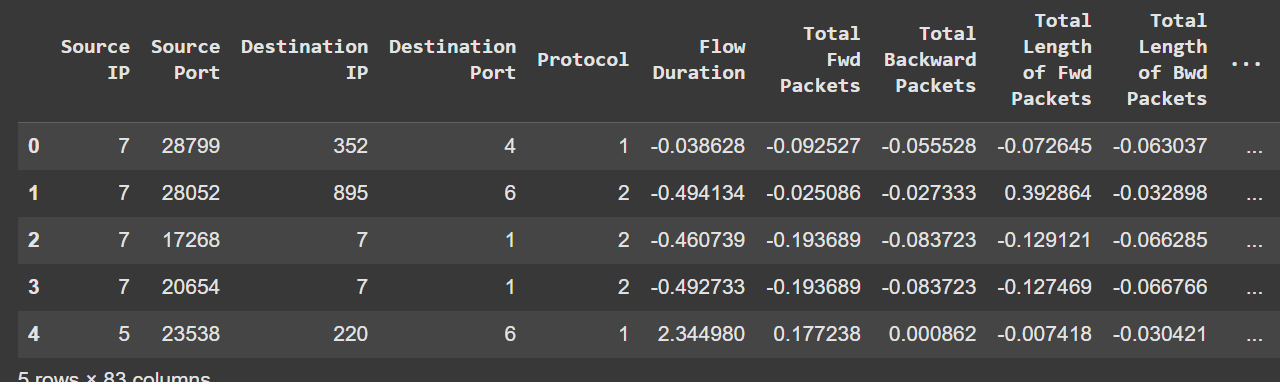
|  |
| --- |
| # Нормализуем количественные признаки  scaler = StandardScaler()  dataset[cols\_for\_normalization] = scaler.fit\_transform(dataset[cols\_for\_normalization])  dataset.head() |

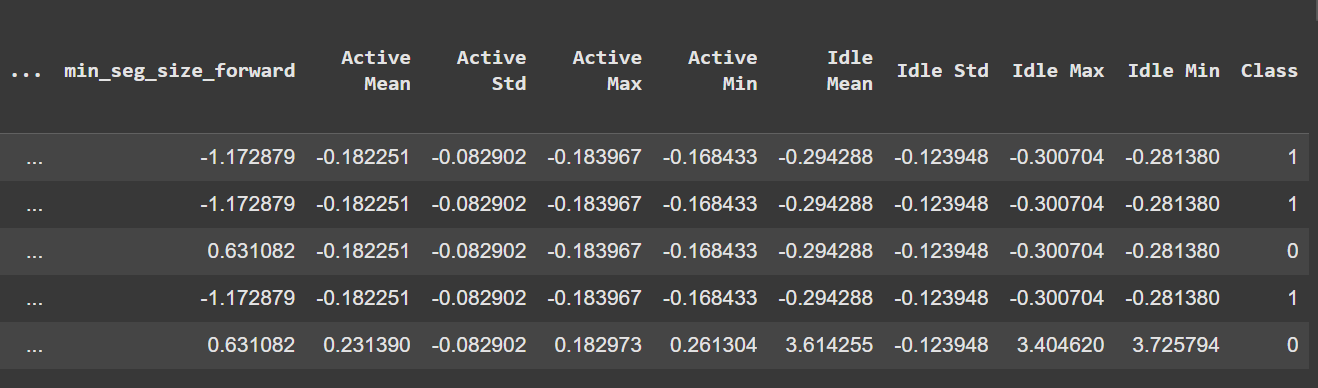




Категориальные фичи и целевую переменную заменяем на метки

|  |
| --- |
| label\_enc = LabelEncoder()  # Для Class: 1 - троян, 0 - все чисто  dataset['Class'] = label\_enc.fit\_transform(dataset['Class'])  dataset[' Source Port'] = label\_enc.fit\_transform(dataset[' Source Port'])  dataset[' Destination Port'] = label\_enc.fit\_transform(dataset[' Destination Port'])  dataset[' Protocol'] = label\_enc.fit\_transform(dataset[' Protocol'])  dataset[' Source IP'] = label\_enc.fit\_transform(dataset[' Source IP'])  dataset[' Destination IP'] = label\_enc.fit\_transform(dataset[' Destination IP']) |



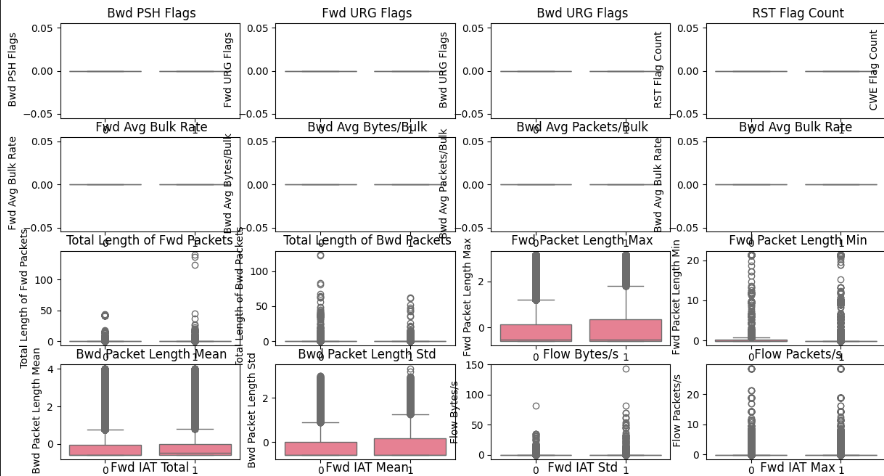


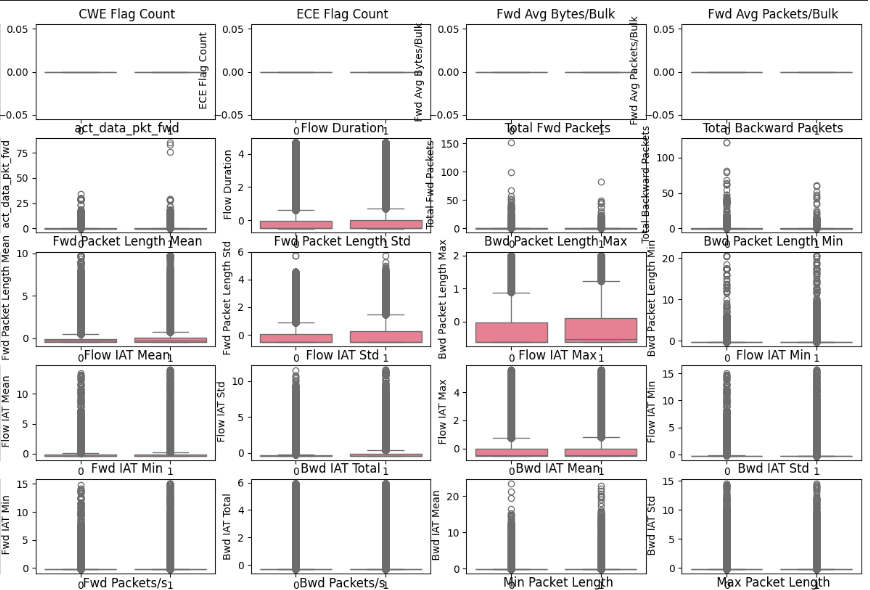
**Этап 2. Подготовка данных**

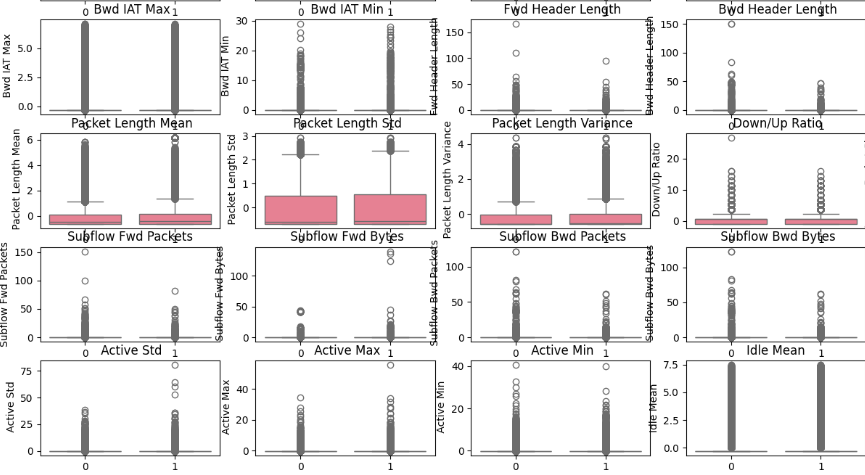
**Очистка выбросов**

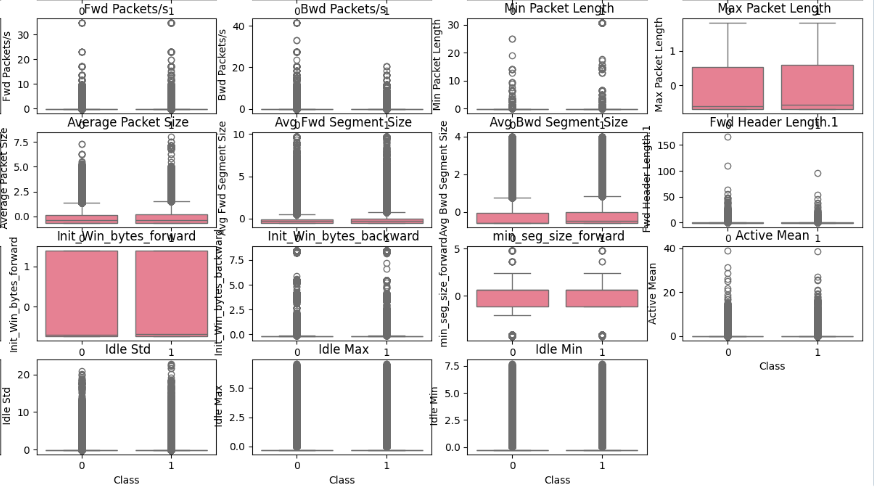
Для начала построим боксплоты для каждой фичи относительно целевой переменной, а затем проанализируем: нужно удалять выбросы или нет.

|  |
| --- |
| # Создаем boxplot для каждого числового признака  plt.figure(figsize=(30, 20))  for i, feature in enumerate(cols\_for\_normalization):  plt.subplot(10, 8, i + 1)  sns.boxplot(x='Class', y=feature, data=dataset)  plt.title(feature)  plt.show() |

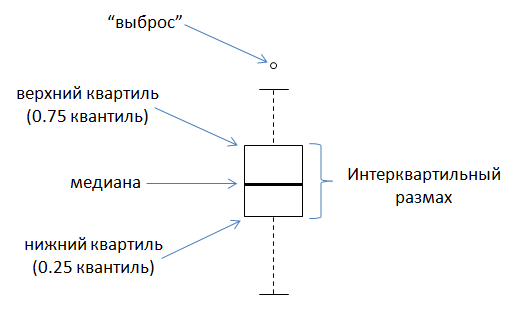








**Вывод по графикам:**

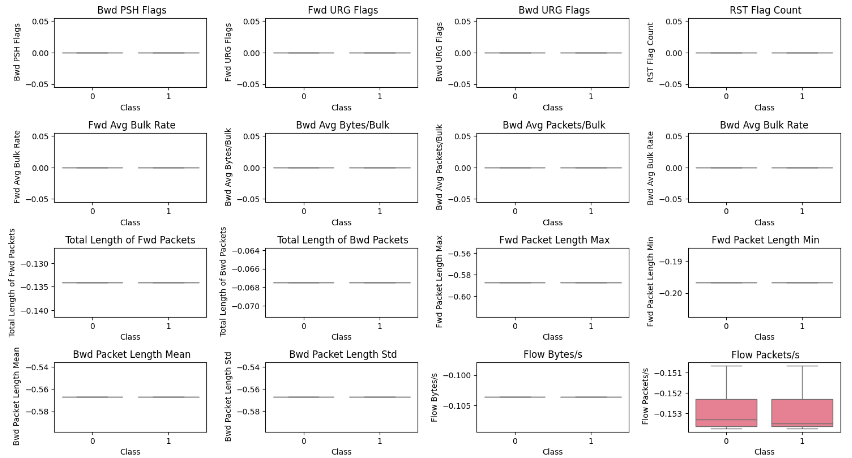


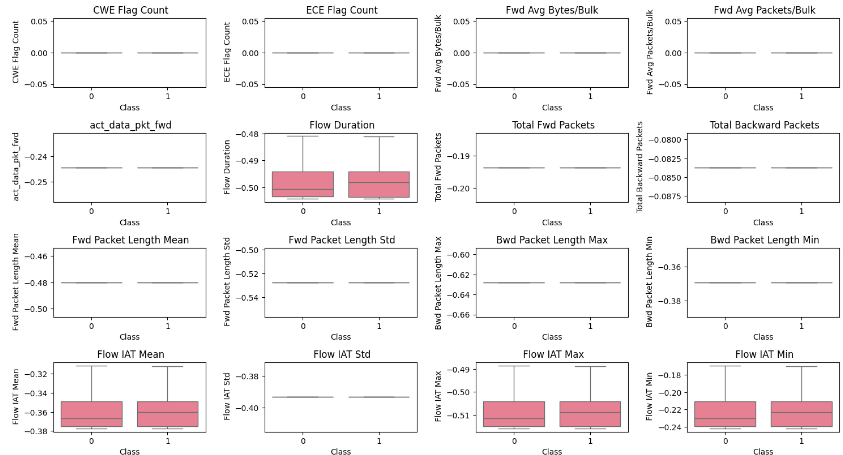
Если посмотреть на структуру графика (что на рисунке) и на полученные графики из нашего датасета, то станет ясно, что наблюдается большое количество выбросов. Необходимо их удалить для каждой количественной характеристики. Коэффициент 1.1 выбран опытным путем, обычно за стандартное значение берется 1.5

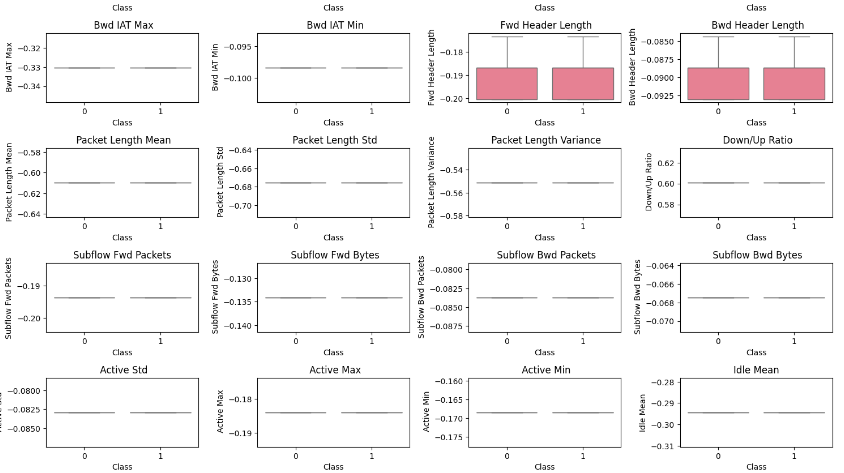
|  |
| --- |
| # убираем otliers  for column in cols\_for\_normalization:  Q1 = dataset[column].quantile(0.25)  Q3 = dataset[column].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.1 \* IQR  upper\_bound = Q3 + 1.1 \* IQR  dataset = dataset[(dataset[column] >= lower\_bound) & (dataset[column] <= upper\_bound)] |

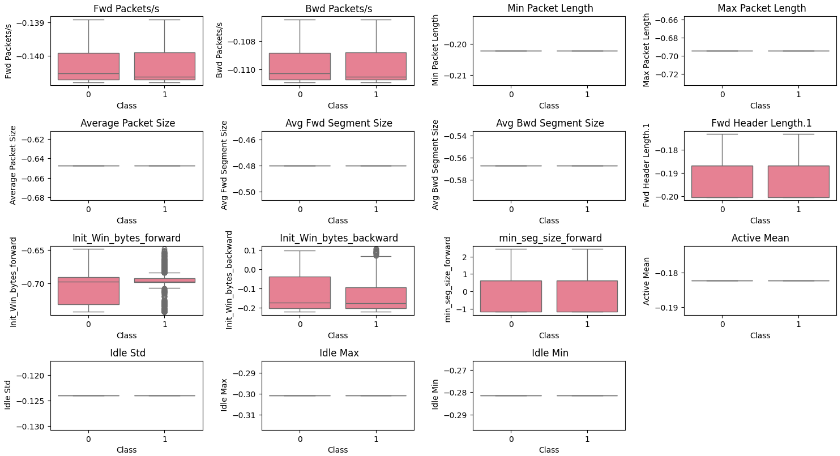
Теперь строим графики заново, чтобы посмотреть наличие выбросов снова.

|  |
| --- |
| # Создаем boxplot для каждого признака  plt.figure(figsize=(30, 20))  for i, feature in enumerate(cols\_for\_normalization):  plt.subplot(10, 8, i + 1)  sns.boxplot(x='Class', y=feature, data=dataset)  plt.title(feature)  plt.tight\_layout()  plt.show() |









**Вывод по графикам:**

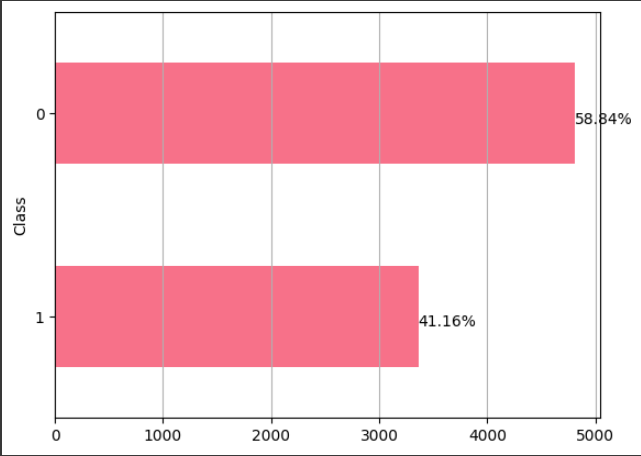
Количество выбросов существенно уменьшилось.

**Этап 2 Подготовка данных**

**Балансировка классов**

Посмотрим распределение классов Benign (=0, безопасно) и Trojan (=1, троян)

|  |
| --- |
| ax = dataset["Class"].value\_counts().sort\_values().plot(kind="barh")  totals= []  for i in ax.patches:  totals.append(i.get\_width())  total = sum(totals)  for i in ax.patches:  ax.text(i.get\_width()+.3, i.get\_y()+.20,  str(round((i.get\_width()/total)\*100, 2))+'%',  fontsize=10, color='black')  ax.grid(axis="x")  plt.show() |



**Вывод:**

Целевая переменная не сбалансирована, исправим это.

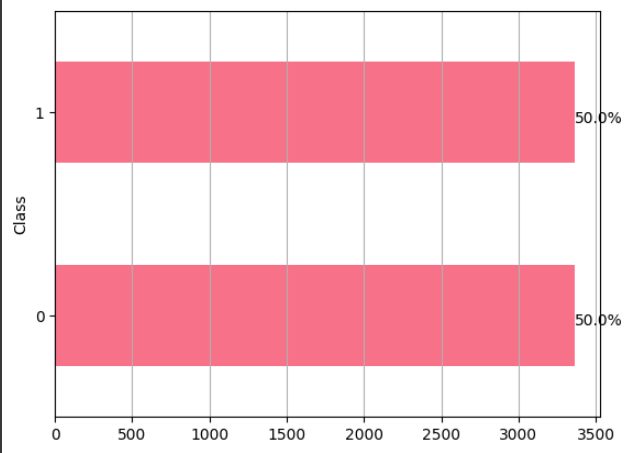
**Будем балансировать методом Random Under Sampling**

Random Under Sampling — это быстрый и простой способ сбалансировать данные путем случайного выбора подмножества данных для целевых классов:

|  |
| --- |
| # Запоминаем количество строк в датасете, где Class = Tojan (так как строк с троянами меньше)  num\_of\_trojan\_rows = len(dataset[dataset['Class'] == 1])  # Перемешиваем датасет  dataset = dataset.sample(frac=1)  # Берем все строки троянов (так как их меньше)  trojan\_dataset = dataset.loc[dataset['Class'] == 1]  # Берем строки, где все хорошо, размером с "троянский" датасет  benign\_dataset = dataset.loc[dataset['Class'] == 0][:num\_of\_trojan\_rows]  # Склеиваем оба датасета и получаем сбалансированный датасет  normal\_distributed\_dataset = pd.concat([trojan\_dataset, benign\_dataset])  # Перемешиваем датасет снова  dataset = normal\_distributed\_dataset.sample(frac=1, random\_state=42)  dataset.head() |

Смотрим на результат:

|  |
| --- |
| ax = dataset["Class"].value\_counts().sort\_values().plot(kind="barh")  totals= []  for i in ax.patches:  totals.append(i.get\_width())  total = sum(totals)  for i in ax.patches:  ax.text(i.get\_width()+.3, i.get\_y()+.20,  str(round((i.get\_width()/total)\*100, 2))+'%',  fontsize=10, color='black')  ax.grid(axis="x")  plt.show() |



**Вывод:**

Балансировка прошла успешно

**Этап 2. Подготовка данных**

**Определение важности признаков при помощи RandomForestClassifier**

RandomForestClassifier представляет собой ансамбль моделей, который использует несколько деревьев решений. Важность каждого признака определяется несколькими факторами:

1. Случайная выборка характеристик: при создании дерева решения RandomForest выбирает случайное подмножество фич и из-за этого при сравнении проще понять насколько большой вклад вносит добавление той или иной фичи.

2. Оценка значимости через снижение неопределённости: Также важность фичи можно оценить насколько хуже становятся метрики если убрать ту или иную фичу. Для этого используются такие метрики, как Gini impurity или Information Gain, позволяющие измерять влияние каждой фичи.

3. Агрегирование значимостей: после создания всех деревьев леса важность каждой фичи усредняется (получаем наиболее точное значение «важности»). Те фичи, которые использовались чаще и привели к большему снижению неопределённости, получают более высокий рейтинг.

4. Непараметрическая природа подхода: Модели RandomForestClassifier без разницы что оценивать: линейные и нелинейные данные могут использоваться одинаково эффективно.

Разделяем датасет на наборы для тренировки и теста модели случайного леса:

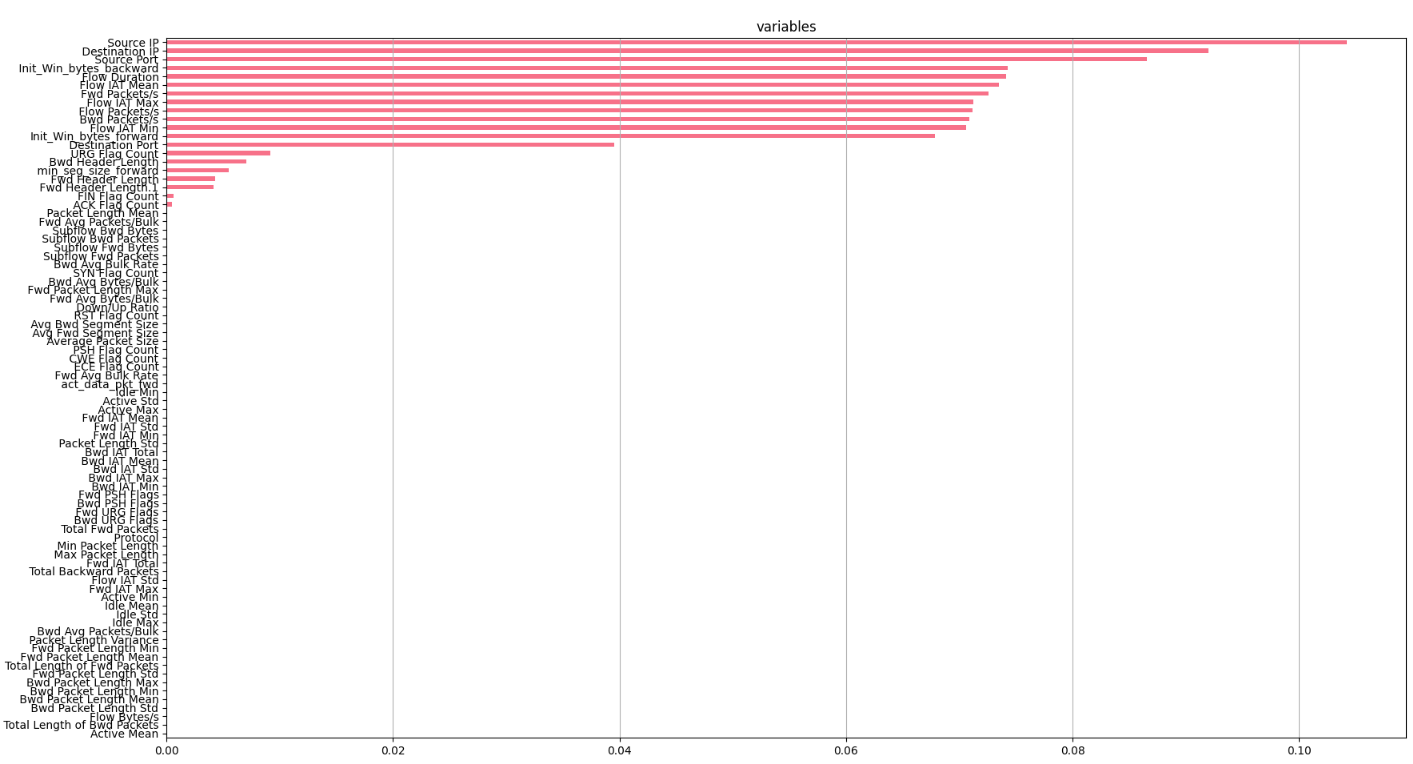
|  |
| --- |
| # Делаем копию, чтобы не было сюрпризов с исходным датасетом  df = dataset.copy()  # Определяем фичи и целевую переменную  y = df['Class'] # Целевая переменная  X = df.drop(columns=['Class']) # Все колонки кроме 'Class'  # Разделяем данные на тренировочную и тестовую выборки  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |

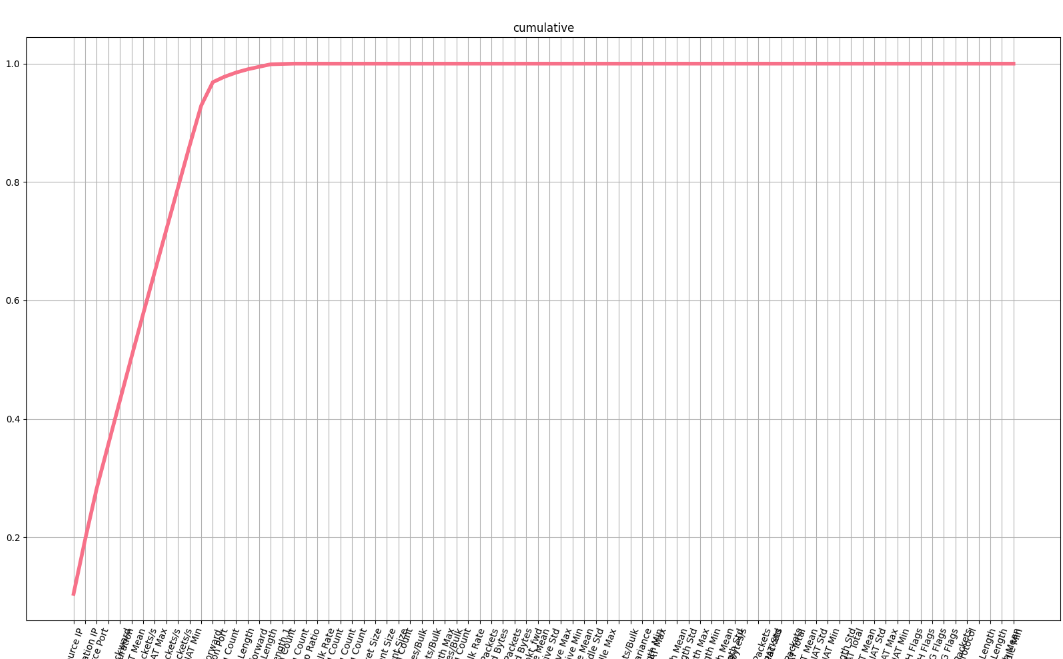
Обучаем модель и запоминаем метрики.

|  |
| --- |
| X = X\_train.values  y = y\_train.values  # Имена столбцов в data  feature\_names = X\_train.columns.tolist()  model = ensemble.RandomForestClassifier(n\_estimators=100,  criterion="entropy", random\_state=0)  model.fit(X,y)  # Это оценка, присваиваемая функциям модели машинного обучения, которая определяет, насколько «важной» является функция для прогноза модели.  importances = model.feature\_importances\_  dtf\_importances = pd.DataFrame({"IMPORTANCE":importances,  "VARIABLE":feature\_names}).sort\_values("IMPORTANCE",  ascending=False)  dtf\_importances['cumsum'] = dtf\_importances['IMPORTANCE'].cumsum(axis=0)  dtf\_importances = dtf\_importances.set\_index("VARIABLE") |

Строим 2 графика: важности и кумулятивности фич - а затем запоминаем названия тех фич, которые описывают 99% важности в сумме (для этого ввели переменную threshold)

|  |
| --- |
| fig, ax = plt.subplots(nrows=2, ncols=1, sharex=False, sharey=False,figsize=(20,25),dpi=100)  fig.suptitle("Features Importance", fontsize=20)  ax[0].title.set\_text('variables')  dtf\_importances[["IMPORTANCE"]].sort\_values(by="IMPORTANCE").plot(  kind="barh", legend=False, ax=ax[0]).grid(axis="x")  ax[0].set(ylabel="")  ax[1].title.set\_text('cumulative')  dtf\_importances[["cumsum"]].plot(kind="line", linewidth=4,  legend=False, ax=ax[1])  ax[1].set(xlabel="", xticks=np.arange(len(dtf\_importances)),  xticklabels=dtf\_importances.index)  plt.xticks(rotation=70)  plt.grid(axis='both')  plt.show()  threshold = 0.99  important\_features = dtf\_importances[dtf\_importances['cumsum'] <= threshold]  important\_features\_list = important\_features.index.tolist()  print(important\_features\_list)  important\_features\_list.append('Class') |





Выводим в консоль те фичи, которые удовлетворили нашим критериям

|  |
| --- |
| print(important\_features\_list) |

Результат

[' Source IP', ' Destination IP', ' Source Port', ' Init\_Win\_bytes\_backward', ' Flow Duration', ' Flow IAT Mean', 'Fwd Packets/s', ' Flow IAT Max', ' Flow Packets/s', ' Bwd Packets/s', ' Flow IAT Min', 'Init\_Win\_bytes\_forward', ' Destination Port', ' URG Flag Count', ' Bwd Header Length', 'Class']

**Этап 3. Обучение разных моделей.**

1. Logistic Regression (LR): Модель, используемая для бинарной классификации, которая предсказывает вероятность принадлежности к классу, используя логистическую функцию. Она оценивает зависимость между одной или несколькими независимыми переменными и бинарной зависимой переменной.

2. Linear Discriminant Analysis (LDA): Статистический метод, используемый для классификации и уменьшения размерности. Он ищет линейные комбинации признаков, которые лучше всего разделяют два или более классов.

3. K-Nearest Neighbors (KNN): Алгоритм классификации, который классифицирует объект на основе ближайших соседей в пространстве признаков. Он основывается на расстоянии между объектами и может использоваться как для классификации, так и для регрессии.

4. Classification and Regression Trees (CART): Метод, который использует дерево решений для классификации или регрессии. Он разбивает данные на подмножества на основе значений признаков, создавая дерево, где каждый узел представляет собой условие на признак.

5. Gaussian Naive Bayes (NB): Вероятностный классификатор, основанный на применении теоремы Байеса с предположением о независимости признаков. Он предполагает, что признаки имеют гауссовское распределение.

6. Gradient Boosting Classifier (GBC): Метод ансамблевого обучения, который строит модель путем последовательного добавления слабых моделей и корректировки ошибок предыдущих моделей. Он использует градиентный спуск для минимизации функции потерь.

7. Random Forest Classifier (RFC): Ансамблевый метод, который использует множество деревьев решений для улучшения точности классификации. Каждое дерево обучается на случайной подвыборке данных, и итоговый результат определяется голосованием всех деревьев.

Создаем новый датасет, состоящий из самых важных фич (на основе этапа 2).

|  |
| --- |
| dataset = dataset[important\_features\_list] |

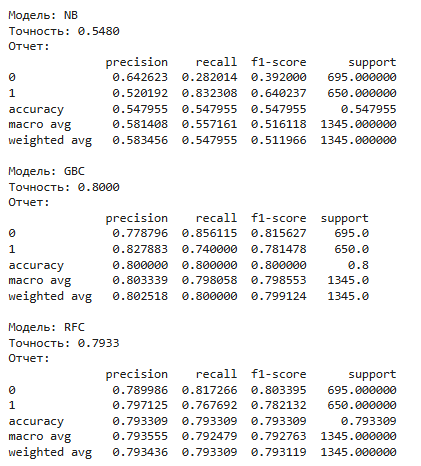
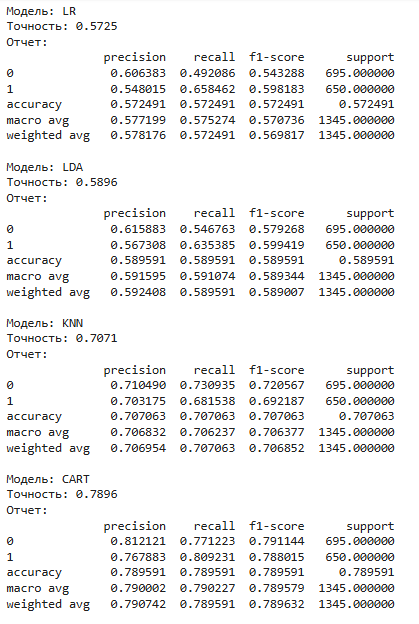
Разделяем датасет на тестовую и тренировочную выборки.

|  |
| --- |
| y = dataset['Class']  X = dataset.drop(columns=['Class'])  # Разделяем данные на тренировочную и тестовую выборки  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |

Для начала просто обучим модели и посмотрим, какие из них лучше всего справляются

|  |
| --- |
| # создаем лист для тех моделей, которые будем изучать  models = []  models.append(('LR', LogisticRegression(solver='lbfgs', max\_iter=5000)))  models.append(('LDA', LinearDiscriminantAnalysis()))  models.append(('KNN', KNeighborsClassifier()))  models.append(('CART', DecisionTreeClassifier()))  models.append(('NB', GaussianNB()))  models.append(('GBC', GradientBoostingClassifier()))  models.append(('RFC', RandomForestClassifier()))  # Обучение и вывод результатов  for name, model in models:  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  report = classification\_report(y\_test, y\_pred, output\_dict=True)  print(f"Модель: {name}")  print(f"Точность: {accuracy:.4f}")  print(f"Отчет:\n{pd.DataFrame(report).transpose()}\n") |

Результаты:



Вывод:

Исходя из результатов, видно, что лучше всего справились RandomForestClassifier, DecisionTreeClassifier, GradientBoostingClassifier и KNeighborsClassifier

**Этап 4. Усиленное обучение и строгая валидация наилучших моделей**

Применим поиск по сетке оптимальных гиперпараметров для выбранных моделей и оценим их показатели при помощи кросс-валидации, основанной на 3 фолдах.

Что такое **поиск оптимальных гиперпараметров по сетке** – если по-простому, то мы предоставляем набор различных гиперпараметров (настройки модели), с которыми может работать модель, а далее алгоритм поиска определяет при каких гиперпараметрах модель лучше всего справилась с задачей.

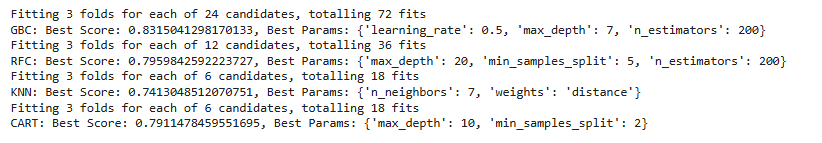
**Кросс-валидация** – это алгоритм, который разбивает набор данных на сплиты и фолды и затем производит тренировку модели на разных наборах и тестирует ее также на разных наборах. Из этого получается некое «усредненное значение» модели. Благодаря кросс-валидации можно избежать переобучения. Вот схема разбивки набора данных на 5 фолдов при кросс-валидации (у нас фолдов 3):



Эти две функции сочетает в себе функция GridSearchCV. Ее мы далее и будем использовать

|  |
| --- |
| # Определяем список моделей с их гиперпараметрами  models = []  params = {  'GBC': {  'model': GradientBoostingClassifier(),  'param': {  'n\_estimators': [100, 200],  'learning\_rate': [0.01, 0.1, 0.5],  'max\_depth': [3, 5, 7, 9]  }  },  'RFC': {  'model': RandomForestClassifier(),  'param': {  'n\_estimators': [100, 200],  'max\_depth': [None, 10, 20],  'min\_samples\_split': [2, 5]  }  },  'KNN': {  'model': KNeighborsClassifier(),  'param': {  'n\_neighbors': [3, 5, 7],  'weights': ['uniform', 'distance']  }  },  'CART': {  'model': DecisionTreeClassifier(),  'param': {  'max\_depth': [None, 10, 20],  'min\_samples\_split': [2, 5]  }}}  # для дальнейшего запоминания наилучших моделей  best\_models = {}  # Перебор моделей и применение GridSearchCV  for name, model\_params in params.items():  model = model\_params['model']  param\_grid = model\_params['param']  grid\_search\_model = GridSearchCV(estimator=model,  param\_grid=param\_grid,  scoring='accuracy',  cv=3,  verbose=3,  n\_jobs=-1)  grid\_result = grid\_search\_model.fit(X\_train, y\_train)  best\_models[name] = grid\_search\_model.best\_estimator\_  # Вывод результатов  print(f"{name}: Best Score: {grid\_result.best\_score\_}, Best Params: {grid\_result.best\_params\_}") |

Результаты:



В итоге, мы получили наилучший показатель в ~83% у модели GradientBoostingClassifier. Далее будем анализировать только ее показатели

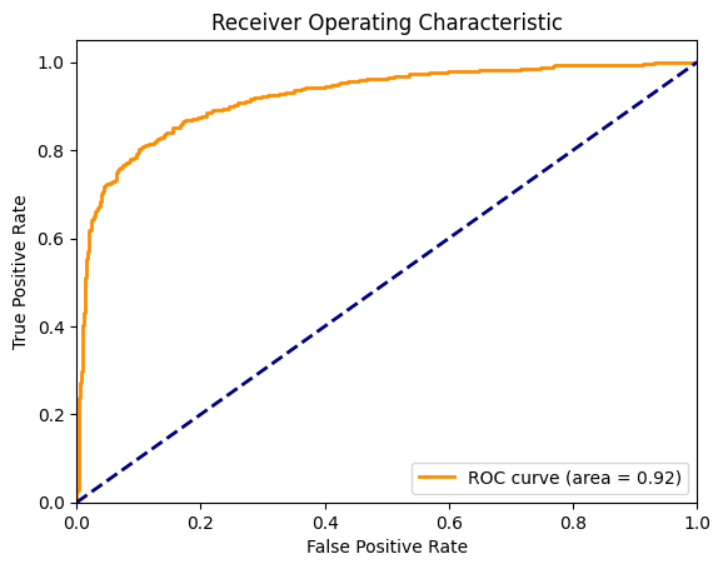
**Этап 5. Анализ метрик GradientBoostingClassifier**

|  |
| --- |
| from sklearn.metrics import roc\_curve, auc  y\_scores = best\_models["GBC"].predict\_proba(X\_test)[:, 1]  # Рассчитываем значения для ROC кривой  fpr, tpr, thresholds = roc\_curve(y\_test, y\_scores)  # Вычисляем площадь под кривой (AUC)  roc\_auc = auc(fpr, tpr)  # Строим ROC кривую  plt.figure()  plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  plt.xlim([0.0, 1.0])  plt.ylim([0.0, 1.05])  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic')  plt.legend(loc='lower right')  plt.show() |

Начнем с того, что такое ROC-кривая. Это кривая вероятности, которая отображает отношение TPR к FPR при различных пороговых значениях. Если по-простому: отделяет «сигнал» от «шума».

Площадь под кривой (AUC) является мерой способности классификатора различать классы. Чем больше площадь под кривой, тем лучше производительность. Когда AUC равен единице, тогда классификатор может правильно различать все положительные и отрицательные точки класса – модель обучена идеально.

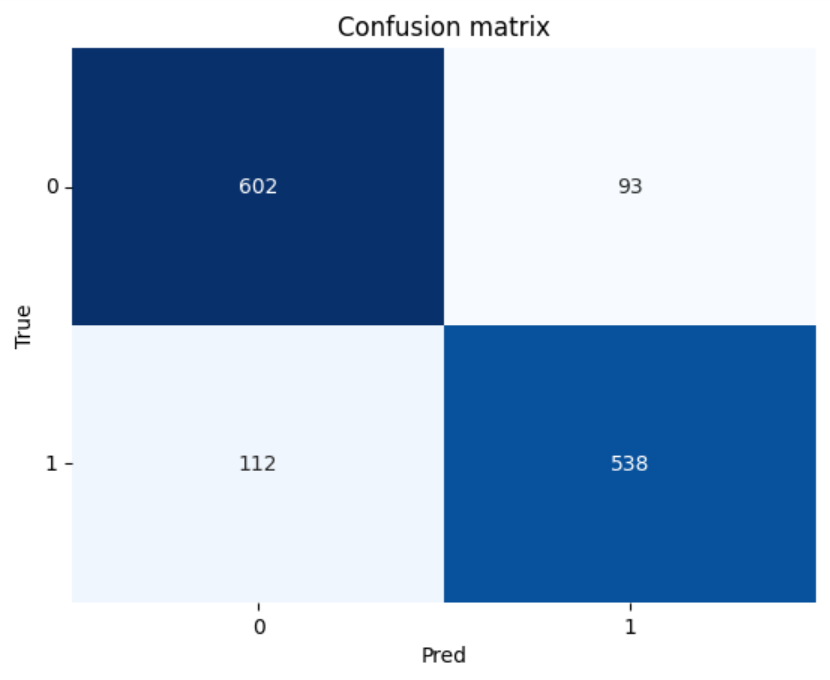
Кривая для нашей лучшей модели GradientBoostingClassifier:

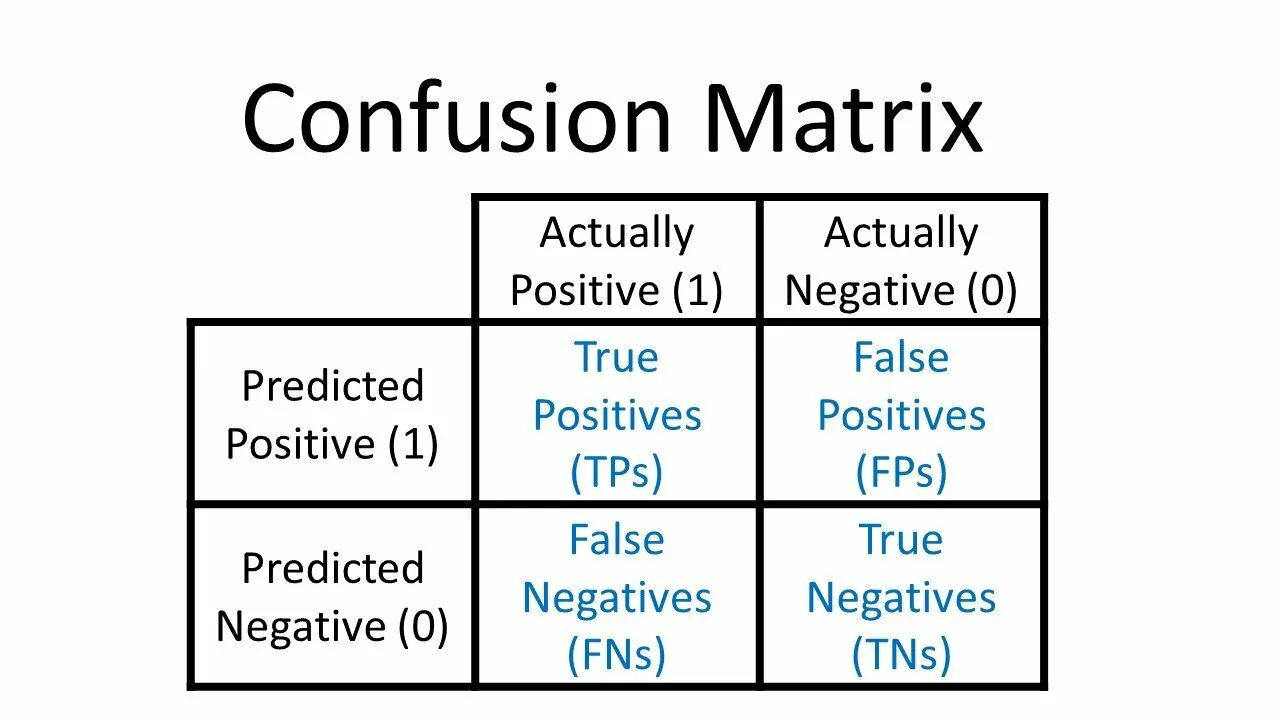


Далее построим матрицу ошибок, чтобы лучше видеть, насколько хорошо модель определяет троянский и безопасный трафики

Матрица ошибок представляет собой квадратную таблицу, в которой отображается количество предсказанных и фактических классов для классификационной модели. В этой матрице строки представляют истинные классы (реальные метки), а столбцы представляют предсказанные классы (метки, которые предсказала модель). Размер матрицы соответствует количеству классов.

|  |
| --- |
| predicted = best\_models["GBC"].predict(X\_test).astype('int32')  classes = np.unique(y\_test)  fig, ax = plt.subplots()  cm = metrics.confusion\_matrix(y\_test, predicted, labels=classes)  sns.heatmap(cm, annot=True, fmt='d', cmap=plt.cm.Blues, cbar=False)  ax.set(xlabel="Pred", ylabel="True", title="Confusion matrix")  ax.set\_yticklabels(labels=classes, rotation=0)  plt.show() |





Общее количество предсказаний:

Total = TP + FP + FN + TN

Подставим значения:

Total = 602 + 93 + 112 + 538

Total = 1345

Неправильные предсказания включают FP и FN:

Incorrect = FP + FN

Incorrect = 93 + 112

Incorrect = 205

Теперь мы можем найти процент неправильных предсказаний:

Incorrect Percentage = (Incorrect / Total) \* 100

Incorrect Percentage = (205 / 1345) \* 100 ≈ 15.24%

Таким образом, процент неправильных предсказаний составляет примерно 15.24%. Модель хорошая, так как процент неправильных предсказаний относительно низкий.

**Вывод**

Анализируя полученные результаты, можно утверждать, что разработанная модель демонстрирует высокую эффективность в распознавании зараженного и безопасного трафика, хотя и не достигает абсолютного совершенства. Тем не менее, она успешно выполняет свою задачу, обеспечивая надежную защиту системы, в которой она гипотетически может быть интегрирована.

**Заключение**

На основе результатов работы классификатора можно сделать вывод, что выдвинутая гипотеза подтвердилась: троянское ПО действительно оставляет след в сетевом трафике и может быть выявлено, если судить по сетевым характеристикам.

Исходя из результата после кросс-валидации у наилучшей модели, который равен 83% точности, можно заявить, что поставленная цель была достигнута (можно с хорошей точностью определить вредоносный трафик).

Для достижения цели были решены следующие задачи:

* Выполнен анализ проблемы, обоснована ее актуальность.
* Осуществлена загрузка данных и подготовка их к анализу (нормализация, обработка выбросов, балансировка классов, а также выявление наиболее важных признаков).
* Осуществлен анализ трафика различными моделями классификации, из которых выбраны наилучшие.
* Для наилучших моделей были произведены усиленное обучение и корректная проверка их результатов.
* Выполнена интерпретация полученных результатов наилучшей модели из всех.

В результате мы получили хороший классификатор, который с большой точностью может определить нежелательное поведение в сети (только касаемо троянского поведения). Данная модель может в дальнейшем использоваться в качестве модуля для антивирусного ПО.

# Список использованных источников и литературы

1. Pedregosa, F., et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 2011.
2. Герасимов, А. Н. "Машинное обучение и анализ данных." Москва: Издательство "Бином", 2021.
3. Aggarwal, C. C. "Recommender Systems: The Textbook." Springer, 2016.
4. McKinney, W. "Python for Data Analysis." O'Reilly Media, 2017.
5. Hastie, T., Tibshirani, R., Friedman, J. "The Elements of Statistical Learning." Springer, 2009.
6. Официальная документация библиотеки Scikit-learn: <https://scikit-learn.org>
7. Официальная документация библиотеки Pandas: <https://pandas.pydata.org>
8. Информация про троянское ПО: <https://www.kaspersky.ru/resource-center/threats/trojans>

<https://www.securitylab.ru/analytics/545053.php>

**Приложения**

Ссылка на GitHub с исходным кодом итоговой работы: <https://github.com/Sharapunzel/pythonML_IKM/blob/main/LabFinal.ipynb>