



ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ  
КИБЕРНЕТИЧЕСКИХ СИСТЕМ  
Кафедра  
«Криптология и кибербезопасность»

---

**Отчет  
о практике**

«Обнаружение внутреннего нарушителя путём выявления стрессового состояния пользователя»

Исполнитель:

студент гр. Б16-505

\_\_\_\_\_

Султанов А.Э.

(подпись, дата)

Научный руководитель:

\_\_\_\_\_

Когос К.Г.

(подпись, дата)

Зам. зав. каф. № 42:

\_\_\_\_\_

Когос К.Г.

(подпись, дата)

---

**Москва – 2020**

---

## **РЕФЕРАТ**

Отчёт 33 с., 6 рис., 4 табл., 7 прил., 33 источника.

### **ВНУТРЕННИЙ НАРУШИТЕЛЬ, ВЫЯВЛЕНИЕ СТРЕССА, ДИНАМИКА НАЖАТИЯ КЛАВИШ, МАШИННОЕ ОБУЧЕНИЕ, АНОМАЛИИ.**

Объектом исследования в данной работе являются методы выявления стресса на основе алгоритмов машинного обучения с применением информации о взаимодействии с клавиатурой и мышью.

Цель работы — оценить возможность выявления стрессового состояния пользователя на основе анализа взаимодействия с клавиатурой и мышью.

Для оценки данной возможности проведён анализ существующих работ, использующих выявление стресса для обнаружения внутреннего нарушителя на основе алгоритмов машинного обучения с применением данных о биометрических показателях. В рамках рассмотренных работ приведена информация об используемых датасетах, процессах накопления данных и применённых алгоритмах машинного обучения для обнаружения угрозы внутреннего нарушителя.

В результате исследования реализованы процессы накопления данных, предобработки данных, обучения и оценки моделей классификаторов и моделей обнаружения аномалий на основе различных алгоритмов машинного обучения с учителем и без учителя с применением информации взаимодействия пользователя с клавиатурой и мышью. Полученные точности выявления стресса с помощью алгоритмов с учителем и алгоритмов обнаружения аномалий составили 88% и 100% соответственно.

## СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ .....	5
ВВЕДЕНИЕ .....	7
1    Сопутствующие работы .....	10
1.1    Обнаружение угрозы с применением алгоритмов с учителем .....	11
1.2    Обнаружение угрозы с применением алгоритмов без учителя .....	12
1.3    Обнаружение угрозы на основе детекции аномалий.....	13
1.4    Обнаружение угрозы путём выявления стрессового состояния.....	14
2    Процесс накопления данных .....	16
3    Выделение признаков и их предобработка .....	19
4    Модели классификаторов и их оценка .....	25
5    Модели обнаружения аномалий и их оценка.....	28
ЗАКЛЮЧЕНИЕ.....	30
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	31
Приложение А Логирование событий клавиатуры и мыши .....	35
Приложение Б Выделение частотных и временных признаков из файлов логирования.....	39
Приложение В Предобработка и выделение наиболее информативных признаков.....	54
Приложение Г Модели классификаторов .....	58
Приложение Д Визуализированные результаты моделей классификаторов.....	60
Приложение Е Модели обнаружения аномалий.....	62

Приложение Ж Визуализированные результаты моделей обнаружения аномалий.....	64
---	----

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящем отчете применяются следующие термины с соответствующими определениями, обозначениями и сокращениями:

RF	— метод случайного леса (Random Forest).
k-NN	— метод k-ближайших соседей (k-nearest Neighbors Algorithm).
SVM	— метод опорных векторов (Support Vector Machine).
GB	— метод градиентного бустинга (Gradient Boosting).
DT	— дерево принятия решений (Decision Tree).
LR	— логистическая регрессия (Logistic Regression).
ROC	— рабочая характеристика приёма (Receiver Operating Characteristic).
AUC	— площадь под ROC-кривой (Area Under ROC curve).
DBN	— сеть глубоких убеждений (Deep Belief Network).
GSS	— метод золотого сечения (Golden Section Search).
IF	— изолирующий лес (Isolation Forest).
PCA	— метод главных компонент (Principal Component Analysis).
DNN	— глубокая нейронная сеть (Deep Neural Network).
RNN	— рекуррентная нейронная сеть (Recurrent Neural Network).
BA	— бэггинг или бутстрэп-агрегирование (Bootstrap aggregating).
MLP	— многослойный перцептрон (Multilayer Perceptron).
IF	— изолирующий лес (Isolation Forest).
OCSVM	— одноклассовый метод опорных векторов (One-Class Support Vector Machine).
SKB	— метод выбора k-лучших (SelectKBest).
RC	— метод робастной ковариации (Robust Covariance).
LOF	— локальный уровень выброса (Local Outlier Factor).
TP	— истинно положительный (True Positive).
FP	— ложно положительный (False Positive).
TN	— истинно отрицательный (True Negative).

FN — ложно отрицательный (False Negative).

## **ВВЕДЕНИЕ**

В современном мире информационных технологий обеспечение информационной безопасности становится всё более сложной задачей. Несмотря на огромные усилия со стороны компаний, разрабатывающих средства эффективного противодействия информационным угрозам, для некоторых из угроз данная задача остаётся всё ещё нерешённой.

Одной из таких угроз, с которой сталкиваются большинство компаний, в той или иной степени связанных с миром технологий, является угроза внутреннего нарушителя. Данный вид угрозы представляет наибольшую опасность в силу огромного числа порождающих её источников и недостатка эффективных средств противодействия этой угрозе, вследствие чего компании несут огромные финансовые потери. Именно поэтому угроза внутреннего нарушителя является актуальной и на данный момент ей уделяется значительное внимание со стороны исследователей в мире информационной безопасности.

Объяснением отсутствия достаточного количества средств борьбы против угрозы внутреннего нарушителя является характер данной угрозы. Несмотря на то, что результат действий внутреннего нарушителя очевиден, очень сложно найти информативные показатели, которые позволили бы обнаружить аномальное поведение пользователя информационной системы и отличить его от нормального. Для анализа поведения можно использовать данные электрокардиограммы (ЭКГ) или электроэнцефалограммы (ЭЭГ) [1,2,3], температуры тела [1], проводимости кожи [2], движения глаз [4] и других биометрических показателей.

Однако способы получения большинства видов биометрических показателей являются инвазивными, то есть требуют использования специального и дорогостоящего оборудования в виде датчиков и камер, которые чаще всего находятся в непосредственном контакте с участником эксперимента в процессе накопления данных. Этот недостаток является причиной того, что методы, разработанные на основе инвазивных способов получения биометрических показателей, сложно применить на практике.

При этом существуют некоторые неинвазивные способы накопления данных для анализа поведения, которые основы на использовании весьма доступных инструментов, например клавиатуры и мыши, которые легко можно найти в любом офисе. Эти недорогостоящие инструменты могут выступать в качестве датчиков, которые предоставляют поведенческую информацию или, иными словами, поведенческую характеристику, состоящую из клавиатурного почерка, динамики нажатия клавиш и жестов. Поведенческую характеристику можно использовать в решении задач аутентификации [5,6,7,8,9], детекции эмоционального состояния [10,11,12,13,14,15], а также в задачах обнаружения угрозы внутреннего нарушителя [16].

Данная работа посвящена исследованию методов обнаружения внутреннего нарушителя путём выявления стрессового состояния пользователя на основе анализа взаимодействия с клавиатурой и мышью. Как и в работе [16], исследование построено на проверке предположения о том, что при совершении неправомерных действий у внутреннего нарушителя под воздействием индуцированного стрессового состояния меняются поведенческие показатели. Однако в отличие от работы [16] применяется более обширный диапазон выделенных из сырых данных признаков и используется большее число алгоритмов для обнаружения угрозы.

В первом разделе проведён анализ сопутствующих работ. Сначала проанализирован класс работ, основанный на применении алгоритмов машинного обучения с учителем, в пределах каждой работы рассмотрены использованные датасеты и приведены результаты классификации применённых алгоритмов. Аналогично рассмотрен класс работ с применением алгоритмов машинного обучения без учителя. Также рассмотрены методы обнаружения угрозы внутреннего нарушителя путём выявления стресса.

Во втором разделе рассмотрен процесс накопления данных, включающий в себя требования к сценариям для накопления данных, их описание и обоснование использования данных сценариев.

В третьем разделе описан процесс выделения признаков из файлов логирования, их обработка и предобработка.

В четвёртом разделе рассмотрены модели классификаторов аномального и нормального поведения, построенные на основе алгоритмов машинного обучения с учителем, приведены результаты оценок моделей, а также проведена их сравнительная характеристика.

В пятом разделе рассмотрены модели обнаружения аномалий, построенные на основе алгоритмов машинного обучения без учителя, приведены результаты оценок данных моделей, а также проведена их сравнительная характеристика.

## **1 Сопутствующие работы**

Угрозу внутреннего нарушителя могут представлять текущий или бывший работник, сотрудник или бизнес-партнёр, которые имеют или имели привилегии доступа к сети, системе или данным организации и преднамеренно или непреднамеренно выполняют действия, которые негативно влияют на конфиденциальность, целостность и доступность организации или информационной системы [17].

Основными типами внутренних нарушителей являются те, кто действуют преднамеренно и злонамеренно и те, кто представляют угрозу организации в силу своей невнимательности, небрежности и совершения нечаянных действий [18]. К основным целям и задачам первой категории можно отнести:

- саботаж;
- кража интеллектуальной собственности;
- шпионаж;
- мошенничество с целью финансовой выгоды.

Причины угроз со стороны членов второй категории:

- человеческие ошибки;
- фишинг;
- вредоносные программы;
- непреднамеренное пособничество;
- украденные учётные записи.

Угроза внутреннего нарушителя рассматривается в качестве основной проблемы безопасности организаций [19] и по данным, приведённым в работе [20] приблизительно 87% случаев угроз безопасности организации зафиксированы, как угрозы со стороны внутренних нарушителей. В данной работе будут рассмотрены только случаи преднамеренного создания угрозы безопасности внутренним нарушителем и приведены основные методы обнаружения с применением алгоритмов машинного обучения.

Для обнаружения угрозы внутреннего нарушителя применяются три основные категории алгоритмов машинного обучения:

- алгоритмы с учителем;
- алгоритмы без учителя;
- алгоритма обнаружения аномалий из группы алгоритмов без учителя;

### **1.1 Обнаружение угрозы с применением алгоритмов с учителем**

В работе [21] проведён сравнительный анализ классификаторов, обученных на общедоступном датасете CERT Insider Threat database, являющийся набором большого количества реальных случаев угроз внутреннего нарушителя с информацией о трафике веб-ресурсов и почтовых ресурсов, а также логах файловой системы. В качестве алгоритмов, составляющих основу классификаторов, были выбраны такие, как RF, k-NN, GB, DT, LR, их вариации и сочетания. Лучшие результаты были показаны классификаторами на основе алгоритма RF и его вариаций. Достигнутая точность как на полном, так и на урезанном датасете варьировалась от 94% до 98%. Однако классификаторы на основе RF проиграли в быстроте обучения классификаторам на основе k-NN, которые в лучшем случае выдали точность предсказания немножко хуже максимума точности в случае RF.

Аналогичный датасет был проанализирован в работе [22], однако в данной работе помимо алгоритмов RF, SVM, DT и LR были применены нейронные сети. Для каждой из моделей была также создана версия с использованием алгоритмов бустинга, что позволило увеличить точность предсказаний почти для всех моделей. Результаты моделей на основе нейронных сетей не превзошли точности моделей на основе RF, DT и LR, при этом минимальная точность классификации среди всех классификаторов составила почти 92%.

Работа [23] выделяется среди остальных тем, что обучение моделей классификаторов происходит не на статическом датасете, а на непрерывном потоке данных. В качестве алгоритма для модели классификатора используется OCSVM, который после приведения обучаемой выборки в пространство

признаков с большей размерностью, рассматривается как обычный SVM. Максимально полученная точность классификации составила 71%.

В статье [24] предпринята попытка создания модели мультиклассового классификатора на основе алгоритма k-NN для решения задачи двухфакторной аутентификации. Классификатор предсказывал принадлежность пользователя, основываясь на распознавании лица, к одной из четырёх групп:

- разрешённый;
- возможно разрешённый;
- возможно неразрешённый;
- неразрешённый.

## **1.2 Обнаружение угрозы с применением алгоритмов без учителя**

В данном пункте описаны работы, использующие на разных этапах анализа данных алгоритмы без учителя, в частности методы кластеризации.

В работе [25] обнаружение угрозы внутреннего нарушителя реализовано с применением предварительной кластеризации данных на основе графов и последующем выделении аномалий. В качестве источника данных использована база данных CERT. При оценке модели наилучший результат показателя AUC составил 0.76.

В работе [26] для обнаружения угрозы использована модель, основанная на одной из разновидностей глубоких нейронных сетей DBN. Используемый датасет является частью общедоступной базы CERT. Сеть DBN была оптимизирована с помощью алгоритма GSS, а результат работы сети был использован для обучения классификатора на основе SVM. В результате полученная точность обнаружения угрозы внутреннего нарушителя составила 97.8%.

В статье [27] было использовано несколько вариаций метода k-средних. Датасет взят из базы данных CERT. Точность детекции аномалий для наилучшей модели составила 76.71%, однако доля ложно положительных предсказаний составила 20%.

### **1.3 Обнаружение угрозы на основе детекции аномалий**

Наиболее распространённые методы обнаружения угрозы внутреннего нарушителя основаны на алгоритмах детекции аномалий. В данных подходах модели чаще всего обучаются на данных, почти полностью или полностью состоящих из примеров с нормальными поведением, трафиком или иными характеристиками. Обучившись на таких данных, модели способны обнаруживать примеры, сильно отличающиеся по свойствам, что позволяет эффективно выявлять аномалии, в частности угрозу внутреннего нарушителя.

В работе [28] авторы используют нейронные сети, которые обучаются на преобразованных с помощью алгоритма word2vec векторах, используя датасет UNSW-NB15, содержащий реальные примеры нормального и синтетические примеры аномального траффика. Оптимальная конфигурация привела к точности 99,20%, полноте 82,07% и ложно положительному показателю 0,61%.

В статье [29] для исследования используется датасет из базы данных CERT. Применены стандартные алгоритмы обнаружения аномалий, в числе которых IF и OCSVM. Модели построены таким образом, что анализируют датасет, предварительно разделив его по временным циклам. В самом худшем случае метрика AUC выдала значение 0.87, при этом во всех экспериментах модели на основе OCSVM показали себя лучше.

Одна из разновидностей датасета CERT (CERT Insider Threat v6.2) проанализирована на предмет выявления угрозы внутреннего нарушителя с применением подхода детекции аномалий в работе [30]. Модели основаны на глубоких и рекуррентных нейронных сетях. Лучшая модель в среднем с уверенностью в 95.53% обнаруживает аномалии, присутствующие в датасете.

Ещё один пример использования алгоритмов детекции аномалий для обнаружения угрозы внутреннего нарушителя приведён в работе [31]. Авторы используют автоэнкодеры, анализируя датасет NSL-KDD с информацией о сетевых подключениях. Предложенный метод выдал точность обнаружения в 91.70%.

## **1.4 Обнаружение угрозы путём выявления стрессового состояния**

Существуют работы, в которых обнаружение угрозы внутреннего нарушителя основано на предположении о том, что при совершении аномальных действий пользователь, который является текущим или бывшим сотрудником организации, испытывает стресс. Стресс в свою очередь влияет на такие биометрические показатели как пульс, кровяное давление, температуру тела, а также на ритм и динамику нажатия клавиш на устройствах, посредством которых происходит взаимодействие с информационной системой. Следовательно, выявив стресс с использованием биометрических показателей, можно выстроить цепь в обратном порядке и с определённой долей вероятности предполагать, что стресс был вызван из-за неправомерных действий, представляющих собой угрозу внутреннего нарушителя. Такое предположение сделано в работах [1,2,3,4].

В ресурсе [3] проведено комплексное исследование, включающее процессы сбора данных, предобработки, обучения моделей классификаторов и их оценки. В качестве основных биометрических показателей для отслеживания стресса были выбраны сигналы электрокардиограммы (ЭКГ) или электроэнцефалограммы (ЭЭГ). Наибольший интерес представляет процесс сбора данных, в котором используются сценарии, описывающие как действия внутреннего нарушителя, вызывающие стрессовое состояние, так и действия обычного сотрудника компании, не представляющие угрозу безопасности и не индуцирующие стресс. В качестве основы модели классификатора выбран алгоритм SVM. В результате, точность предсказания модели классификатора составила 86%.

Большим недостатком с точки зрения практического применения в работе [3] является необходимость наличия специального оборудования, отслеживающего изменения биометрических показателей. Для решения данной проблемы в статье [16] в качестве отслеживающих датчиков используются клавиатура и мышь, которые дают ценную информацию о динамике нажатия клавиш и кнопок. Как уже было показано в работах [10,11,12,13,14,15]

клавиатурный почерк меняется под воздействием различных эмоциональных состояний. В работе [16] было построено четыре модели классификаторов на основе SVM, RF, k-NN и ВА. Модели были обучены на данных, собранных с помощью двух категорий сценариев, как и в работе [3], описывающих действия внутреннего нарушителя и обычного сотрудника. Полученные точности моделей варьировались от 67.5% до 72.5%. Авторы объясняют плохие результаты недостатком информативных признаков, выделенных из динамики использования клавиатуры. Большинство признаков было выделено из динамики использования мыши, однако в выводах отмечено, что влияние стресса на динамику очевидно, и это подтверждает предположение о том, что умение выявлять стресс посредством биометрических показателей позволяет обнаруживать угрозу внутреннего нарушителя.

Текущее исследование во многом схоже с работой [16], однако присутствуют существенные различия в процессах накопления данных и выделении признаков, а также в использованных алгоритмах машинного обучения для моделей классификаторов и в дополнительном использовании алгоритмов обнаружения аномалий. Далее в работе термины стресс и аномалия будут считаться взаимозаменяемыми для следования алгоритмической терминологии.

## **2 Процесс накопления данных**

По причине отсутствия датасетов для текущего исследования, возникла необходимость организации сбора данных и создания датасета для последующего обучения моделей классификаторов и моделей обнаружения аномалий. Для этого были придуманы сценарии двух категорий, описывающие действия внутреннего нарушителя и действия сотрудника, не представляющего угрозу.

Сценарии первой категории были спроектированы таким образом, чтобы во время их выполнения у участника эксперимента индуцировался стресс. Для этого сценарии первой категории в отличие от сценариев второй категории были ограничены по времени, а действия, которые выполняли участники эксперимента под видом внутреннего нарушителя, описывали реальные ситуации правонарушений, которые могли бы возникнуть в организациях.

В первом сценарии данной категории внутренний нарушитель (сотрудник или бывший сотрудник) садится за компьютер другого сотрудника. Находит архив с конфиденциальной информацией о зарплатах в компании. Архив защищен паролем, однако злоумышленник хорошо знаком с владельцем компьютера и имеет список возможных паролей, один из которых гарантированно подходящий. В распакованном архиве находится информация о зарплате каждого сотрудника, которая сохранена в виде скриншота. Нарушитель собирает как можно больше данных из скриншотов, объединяет их в текстовый документ и отправляет по почте известному списку пользователей, вводя описание и тему. При этом нельзя вместо текстового файла отправлять скриншоты, так как в сети компании стоит сервис, который отслеживает все исходящие файлы, превышающие 20 кб. По возможности нарушитель очищает историю.

Во втором сценарии данной категории внутренний нарушитель (сотрудник) отправил начальнику заявление на повышение заработной платы в виде документа. Начальник прочитал письмо, но не распечатал. Пока его нет на рабочем месте, злоумышленник пытается изменить содержимое заявления на

компьютере начальника, который он оставил включенным, добавляя дополнительные пункты в заявление. Нарушитель входит в почту начальника, находит своё письмо, скачивает документ, удаляет письмо, изменяет содержимое документа, входит на свою почту и отправляет новое письмо с измененным документом начальнику через другой браузер. Помечает письмо в почтовом ящике начальника, как прочитанное. По возможности чистит за собой следы.

В третьем сценарии внутренний нарушитель является главным конструктором в компании, производящей процессоры. На его компьютере хранятся чертежи и характеристики нового процессора. Злоумышленнику требуется переслать данные конкурентам, однако сделать это путём копирования данных, создания нового документа и отправки по почте категорически запрещено, так как в компании установлена система противодействия утечкам информации. Однако нарушитель осведомлён о недостатках системы. Каждые 3 часа в течение 4 минут система архивирует собранные данные и при этом кейлогер отключается. Злоумышленнику необходимо используя ресурс для временного хранения текста, перепечатать туда характеристики и опубликовать пост на страничке в социальной сети, где ссылка к сохранённым на ресурсе характеристикам будет поделена на несколько частей.

Сценарии второй категории не были ограничены во времени, но аналогично сценариям первой категории, они описывали действия, которые мог бы выполнять сотрудник в организации, делая свою работу.

В первом сценарии сотруднику необходимо в браузере найти ответы на предоставленные в списке вопросы. Ответы перепечатать в созданный документ. Далее войти в почтовый ящик с помощью учётных данных, выданных в ходе эксперимента. Электронный адрес и пароль выбраны так, что достаточно легко вводятся, чтобы сымитировать работника, который часто пользуется почтовым ящиком и знает свои электронный адрес и пароль наизусть. Необходимо создать новое письмо, прикрепив файл с ответами, ввести тему и описание письма,

список получателей с электронными адресами различной сложности и отправить его.

Во втором сценарии данной категории сотрудник работает в компании с большим количеством филиалов. Отчётность филиалов компонуется в головном офисе, и занимается этим сотрудник, рассматриваемый в сценарии. Ему на почту пришли отчёты за прошлый месяц в виде таблиц. Необходимо скачать все документы с почтового ящика, скомпоновать их так, чтобы в обобщающем документе обязательно присутствовали поля: филиал, дата, доходы и расходы. В последней строке посчитать суммы полей доходов и расходов. При этом копировать данные из отчётов филиалов нельзя, все данные необходимо вводить вручную. После завершения общего отчёта, необходимо отправить его по почте этим же филиалам с произвольной темой и текстом письма.

В третьем сценарии сотруднику компании по организации мероприятий пришло письмо, в котором ему требуется выслать информацию о достижениях компании за предыдущие 2 года для участия в тендере на проведение ежегодного хакатона для биологов. Информация хранится на компьютере у сотрудника, однако разбросана в нескольких файлах. Необходимо собрать требуемую информацию с локального хранилища и отправить в ответном письме, предварительно объединив собранные данные в виде документа. При этом информацию необходимо перепечатать, а не копировать.

В эксперименте по сбору данных приняло участие 8 человек. Все участники — это студенты со средним возрастом в 20 лет. При этом все участники выполняли сценарии на одном и том же устройстве, во избежание влияния типов клавиатур и мышей на данные.

Для логирования событий клавиатуры и мыши во время выполнения сценариев было написано программное обеспечение на языке питон (Приложение А).

### 3 Выделение признаков и их предобработка

Для анализа данных взаимодействия пользователя с клавиатурой и мышью чаще всего выделяются временные и частотные признаки, как в работах [10,11,14,16]. Временные признаки, которые используются в текущем исследовании, наглядно представлены на рисунке 1.

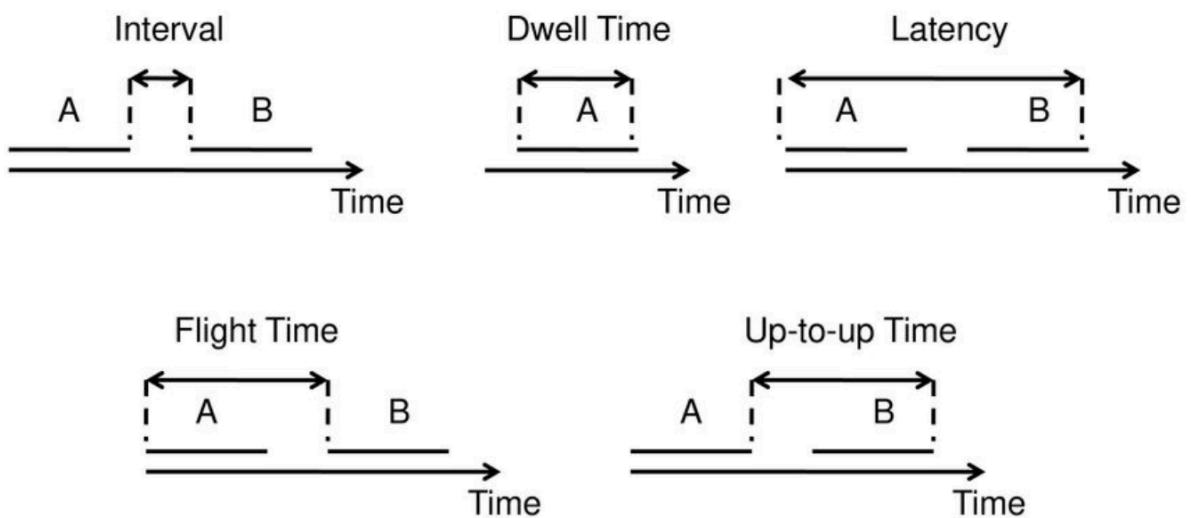


Рисунок 1 — схематическое представление временных показателей [32]

Списки временных и частотных признаков представлены ниже в таблицах 1 и 2.

Таблица 1 — временные признаки

№	Название	Определение
1	dwell time (время удержания)	время (в миллисекундах) между нажатием и отпусканием одной и той же клавиши
2	flight time или down-to-down (время «полёта» или нажатие-нажатие)	время (в миллисекундах) между нажатием одной клавиши и нажатием другой клавиши
3	latency (время задержки)	время (в миллисекундах) между нажатием одной клавиши и отпусканием другой клавиши

## Продолжение таблицы 1

4	interval (интервал)	время (в миллисекундах) между отпусканием одной клавиши и нажатием другой клавиши
5	up-to-up (отпускание-отпускание)	время (в миллисекундах) между отпусканием одной клавиши и отпусканием другой клавиши

Таблица 2 — частотные признаки

№	Название	Определение
1	typing speed (скорость набора)	количество нажатий клавиш или слов в минуту
3	frequency of presses (частота нажатий)	количество использования клавиши в минуту

В отличие от работы [16], где признаки были выделены в большей степени для мыши, в текущем исследовании временные и частотные признаки, кроме скорости набора, выделяются для четырёх больших групп:

- нажатия кнопок мыши;
- специальные клавиши такие, как backspace, del, capslock, shift, tab, alt;
- диграфы, представляющие собой сочетания из двух букв алфавита;
- триграфы, являющиеся сочетаниями из трёх букв алфавита.

Признаки для диграфов и триграфов были вычислены только для кириллицы, так как почти все сценарии требовали использования русской раскладки большую часть времени, а латинская раскладка использовалась в крайне редких случаях и не во всех сценариях, что делало признаки для диграфов и триграфов латинского алфавита неинформативными в силу их полного или почти полного отсутствия в логах событий.

Диграфы и триграфы были выбраны не случайным образом. Выбранные сочетания являются самыми часто встречающимися в русском языке. Для подсчёта

частоты диграфов и триграфов было использовано около 275 миллионов символов текста на русском языке [33].

Как частотные, так и временные признаки были посчитаны с помощью написанного на языке питон скрипта (Приложение Б). С использованием данного скрипта для каждого из файлов логирования были высчитаны частотные признаки для специальных клавиш и временные признаки для всех остальных групп, включая специальные клавиши. Перед сохранением в датасет, для каждого из признаков итогового вектора признаков, кроме частотных, было посчитано среднее значение в пределах каждого файла логирования.

Размерность пространства признаков в полученном датасете, с учётом некоторых отброшенных признаков, составила 191. Отбрасывание признаков стало первом шагом в процессе предобработки данных, в ходе которого были исключены те признаки, которые встретились всего лишь несколько раз во всех файлах логирования:

- правая кнопка мыши;
- caps lock;
- esc;
- alt.

Следующим этапом были удалены те признаки, для значений которых среднеквадратичное отклонение в примерах с аномальным поведением меньше заданного допустимого значения. При вычислении среднеквадратичных отклонений значений признаков учитывались только примеры с аномальным поведением.

Объясняется это тем, что именно в данных примерах отсутствовали значения для большей части признаков, и в случае их сохранения алгоритмы выделения наиболее информативных признаков отобрали бы признаки с близким к нулю стандартным отклонением, так как именно такие признаки хорошо описывают примеры с аномальным поведением в используемом датасете. Однако эти признаки невозможно использовать для максимизации предсказательной способности моделей в силу низкой информативности таких

признаков относительно новых примеров с аномальным поведением. В результате такого отброса размерность пространства признаков уменьшилась до 150.

В силу того, что в использованных сценариях наборы встречающихся элементов из вышеупомянутых групп разные, в датасете присутствовали пустоты для некоторых из признаков. В качестве второго шага предобработки данных эти пустоты были заполнены медианами, которые были рассчитаны отдельно для аномальной и нормальной категорий экземпляров из датасета (Приложение В).

Относительное распределение медиан признаков примеров с нормальным и аномальным поведением приведено на рисунках 2 и 3.

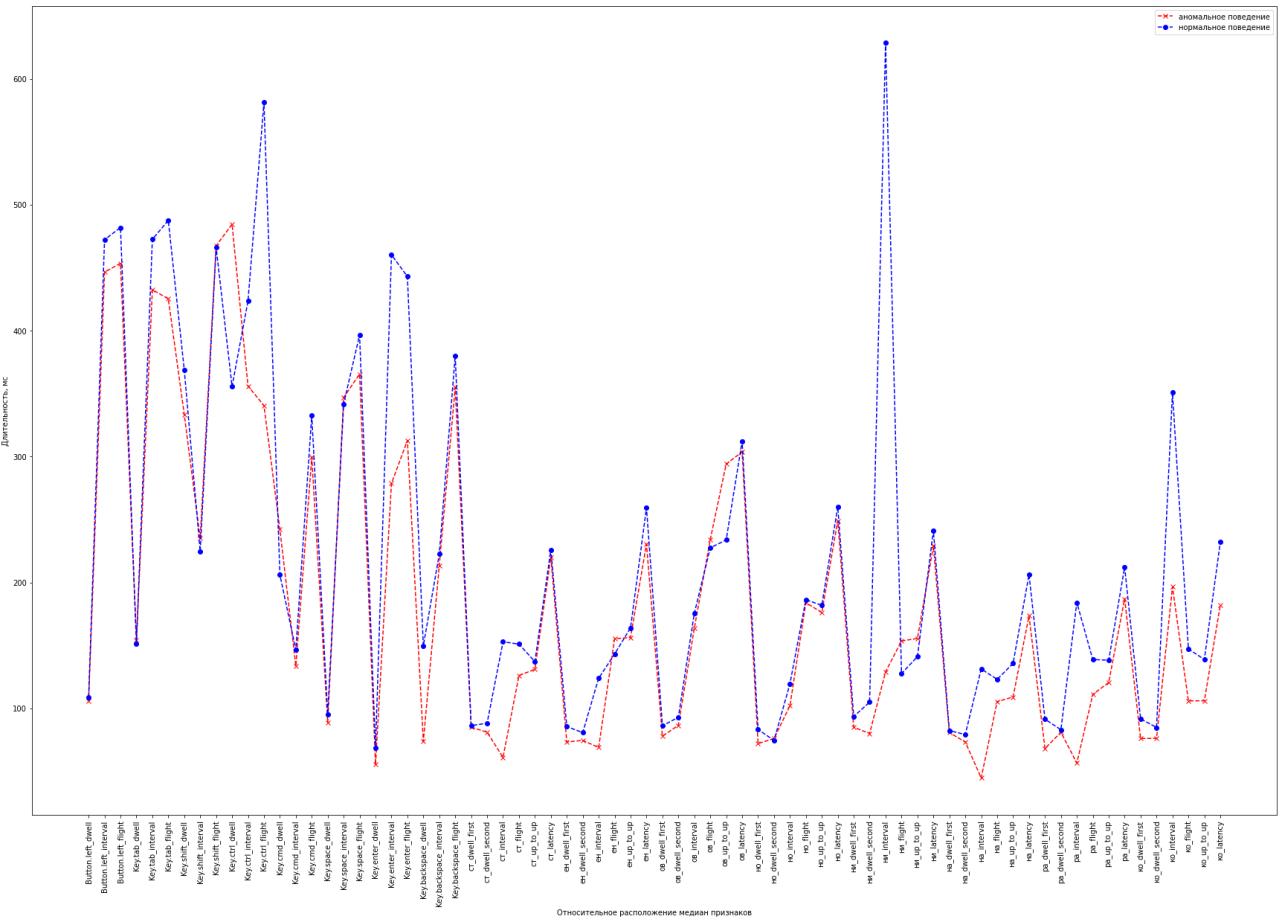


Рисунок 2 — относительное распределение медиан признаков, часть 1

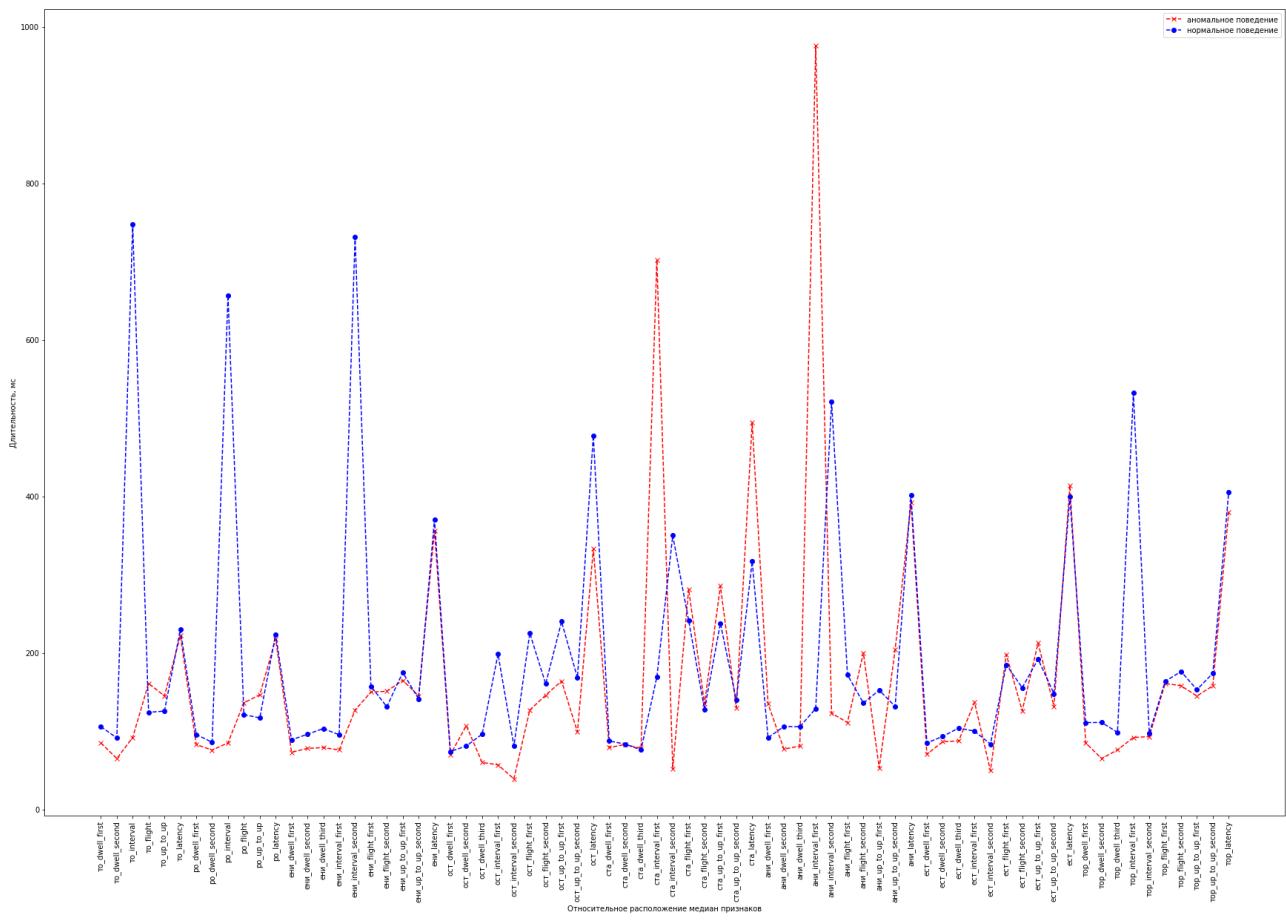


Рисунок 3 — относительное расположение медиан признаков, часть 2

На последнем шаге процесса предобработки был применён алгоритм SKB для выделения трёх наиболее информативных признаков. Данный алгоритм проводит ряд одномерных статистических тестов, задавая каждому признаку некоторую оценку, по которой происходит сортировка и выбор k-лучших признаков. В результате в список самых информативных признаков попали следующие:

- to\_interval;
- sta\_interval\_first;
- ani\_interval\_first.

Уменьшение размерности пространства признаков до трёх позволило визуализировать распределение примеров в пространстве этих признаков с различных ракурсов. Данное распределение приведено на рисунке 4.

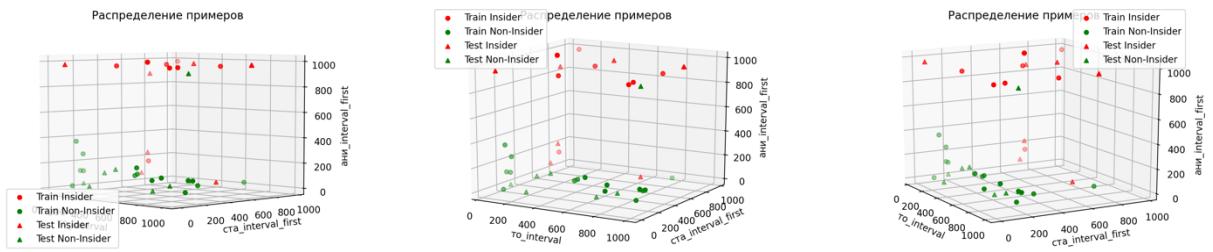


Рисунок 4 — Пространственное распределение примеров

## 4 Модели классификаторов и их оценка

В данном пункте приведены результаты применения алгоритмов машинного обучения с учителем для классификации нормального и аномального поведений пользователя.

Для построения моделей классификаторов были выбраны алгоритмы LR, k-NN, RF, MLP и GB (Приложение Г). Построенные модели были обучены на 49% всего датасета, прошли валидацию на 21% датасета и были протестированы на оставшейся части датасета. При этом использовался набор признаков, который был выделен на шаге предобработки данных. Полученные показатели представлены в таблице 3.

Таблица 3 — Оценка моделей классификаторов

№	Алгоритм	Precision Not-Stress (train / test)	Precision Stress (train / test)	Recall Not-Stress (train / test)	Recall Stress (train / test)	Accuracy (train / test)
1	LR	1.00 / 0.70	1.00 / 0.83	1.00 / 0.88	1.00 / 0.62	1.00 / 0.75
2	k-NN	0.92 / 0.73	1.00 / 1.00	1.00 / 1.00	0.91 / 0.62	0.95 / 0.81
3	RF	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	1.00 / 0.75	0.91 / 0.88
4	MLP	1.00 / 0.70	1.00 / 0.83	1.00 / 0.88	1.00 / 0.62	1.00 / 0.75
5	GB	1.00 / 0.70	1.00 / 0.83	1.00 / 0.88	1.00 / 0.62	1.00 / 0.75

По таблице 3 видно, что для моделей на основе LR, MLP и GB получены одинаковые значения метрик. Вероятнее всего такое совпадение в показателях произошло из-за слишком малого количества данных и удачного пространственного распределения аномальных и нормальных примеров в пространстве признаков.

Последнее позволило всем моделям достичь максимальных или почти максимальных показателей на обучаемой выборке. Однако модели на основе LR, MLP и GB выдали худшие показатели на тестовой выборке. В случае LR это объясняется слабой предсказательной способностью данного алгоритма, в

следствие чего модель затрудняется уловить закономерность распределения данных. А в случае MLP и GB, такие результаты можно объяснить лишь недостаточным количеством данных, так как и MLP, и GB могут улавливать очень сложные закономерности в данных. Возможно, показатели модели на основе LR будут лучше с увеличением объёма данных, но очевидно, что при усложнении функции распределения данных, доля неправильно предсказанных примеров будет расти.

При этом модель на основе k-NN немного превзошла в показателях упомянутые ранее модели. Такие результаты можно объяснить тем, что k-NN для классификации использует расстояние между объектами в пространстве признаков, и если обратить внимание на расположение примеров с нормальным и аномальным поведениями на рисунке 4, то можно заметить их очевидное пространственное разделение. Такое разделение позволяет легко определить принадлежность примера к определённому классу. Однако с увеличением количества данных могут появиться новые пространственные кластеры, и показатели модели на основе k-NN могут существенно снизиться.

Наилучшей оказалась модель на основе алгоритма RF. Это объясняется устройством алгоритма RF, использующем большое количество слабых классификаторов, в данном случае деревьев принятия решений, объединённых в ансамбль, что позволяет рассмотреть большое количество функций, описывающих распределение и усреднив результаты, получить существенную прибавку в сравнении с теми же деревьями принятия решений, взятых в отдельности. Однако в данном случае могло произойти переобучение модели, так как на валидационной выборке показатели точности значительно хуже, чем у моделей на основе LR, GB и MLP.

На рисунке 5 представлены визуализированные результаты классификации на обучаемой и тестовой выборке для самой лучшей модели. Визуализированные результаты для остальных моделей доступны в Приложении Д.

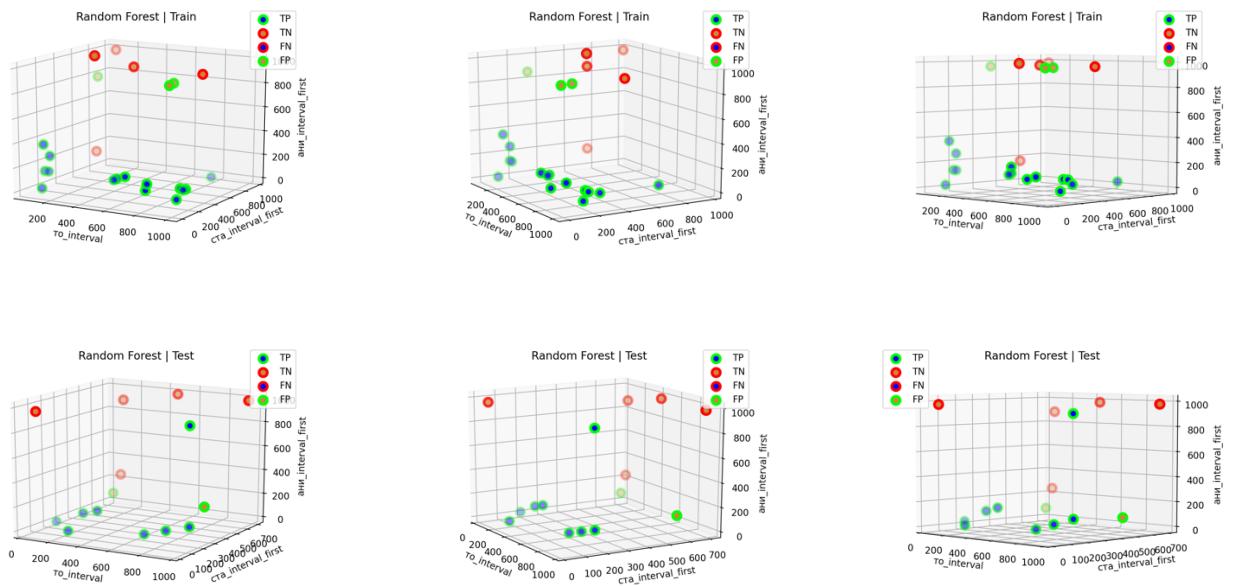


Рисунок 5 — результаты классификации для модели на основе RF

## 5 Модели обнаружения аномалий и их оценка

В данном пункте приведены результаты применения алгоритмов обнаружения аномалий. Алгоритмы обнаружения аномалий основаны на двух предположениях [29]:

- во-первых, большинство шаблонов поведения пользователей в системе являются нормальными с небольшим процентом аномальных случаев;
- во-вторых, аномальные случаи статистически отличаются от нормальных.

Помимо того факта, что нет необходимости маркировать данные для обучения модели, основным преимуществом использования алгоритмов обнаружения аномалий является способность адаптироваться к новым шаблонам аномального поведения.

Для построения моделей обнаружения аномалий выбраны алгоритмы IF, OCSVM, RC и LOF (Приложение Е). Модели обучены только на нормальных примерах, число которых составляет 33% из общего датасета. Модели протестираны на примерах, включающих все аномальные и остаток нормальных. Результаты обучения и тестирования приведены в таблице 4.

Таблица 4 — Оценка моделей обнаружения аномалий

№	Алгоритм	Precision Not-Stress (train / test)	Precision Stress (train / test)	Recall Not-Stress (train / test)	Recall Stress (train / test)	Accuracy (train / test)
1	RC	1.00 / 0.60	— / 0.91	0.81 / 0.75	— / 0.83	0.81 / 0.81
2	OCSVM	1.00 / 0.00	— / 0.75	0.81 / 0.00	— / 1.00	0.81 / 0.75
3	IF	1.00 / 1.00	— / 1.00	0.81 / 1.00	— / 1.00	0.81 / 1.00
4	LOF	1.00 / 0.88	— / 0.96	0.88 / 0.88	— / 0.96	0.88 / 0.94

Модель на основе OCSVM показала худшие результаты, не определив ни одного примера с нормальным поведением в тестовой выборке. Это возможно

объяснить только тем, что гиперплоскость, отделяющая класс с нормальным поведением, в ходе обучения была расположена некорректно.

Оставшиеся модели показали результаты намного лучше модели на основе OCSVM. Модель на основе LOF на тестовой выборке выдала результаты близкие к идеальным. Это объясняется тем, что примеры удачно разделены в пространстве признаков, и так как идея алгоритма LOF заключается в использовании пространственных характеристик относительно k-ближайших соседей, то в данном случае алгоритму удалось с лёгкостью определить расположение группы примеров с нормальным поведением.

Модель на основе IF выдала идеальные показатели точности на тестовой выборке. И возможное объяснение заключается в том, что IF максимально точно сумел разделить примеры в пространстве признаков, рекурсивно перебрав все возможные варианты пространственных разделов.

На рисунке 6 представлены визуализированные результаты модели на основе алгоритма IF. Визуализированные результаты для остальных моделей доступны в Приложении Ж.

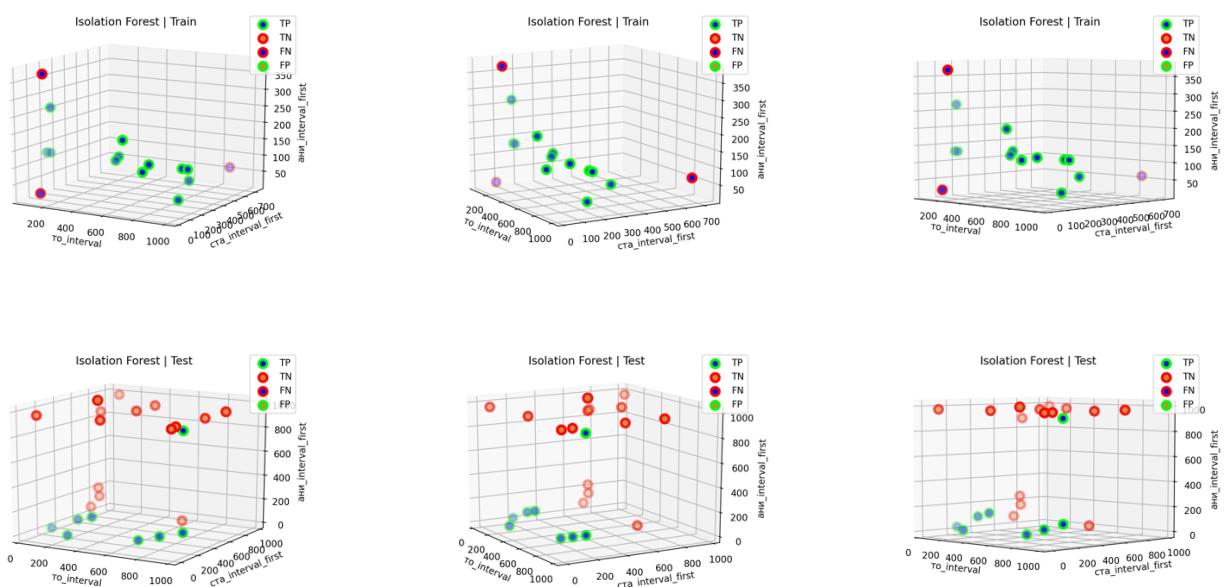


Рисунок 6 — результаты обнаружения аномалий алгоритмом IF

## **ЗАКЛЮЧЕНИЕ**

В данной работе был проведён анализ исследований в области обнаружения угрозы внутреннего нарушителя с использованием различных биометрических показателей, а также продемонстрирована возможность обнаружения аномалий путём выявления стресса с использованием данных взаимодействия с клавиатурой и мышью.

В ходе проведённого исследования был собран датасет, включающий в себя данные логов клавиатуры и мыши. В эксперименте по сбору данных поучаствовали 8 человек, выполнившие сценарии, описывающие как действия нормального сотрудника, так и действия внутреннего нарушителя.

Далее с помощью написанного программного обеспечения из сырых данных были выделены группы временных и частотных признаков, проделаны шаги по предобработке данных, выделению наиболее информативных признаков, построены модели классификаторов на основе 5 различных алгоритмов машинного обучения с учителем, а также построены модели обнаружения аномалий с использованием 4 алгоритмов машинного обучения без учителя. Точность наилучшего классификатора составила 88%. Точность наилучшей модели обнаружения аномалий составила 100%.

Исследование данных показало, что под воздействием стресса динамика использования клавиатуры и мыши существенно меняется, что приводит к увеличению скорости нажатия клавиш и кнопок. Результаты, полученные при тестировании моделей классификаторов, а также моделей обнаружения аномалий, подтвердили сделанное предположение о том, что выявление стресса может сыграть ключевую роль при обнаружении аномалий, одной из причин которых может быть внутренний нарушитель.

В дальнейшем исследовании планируется собрать намного больше данных из разных возрастных категорий пользователей с различными способностями использования клавиатуры, что позволит проанализировать все события на клавиатуре и мыши при выделении признаков, которые были опущены в данном исследовании.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 **Abdulaziz A.** On the Possibility of Insider Threat Detection Using Physiological Signal Monitoring [Текст] / A. Abdulaziz, E. Khalil // Proc. of the ACM 7th International Conference on Security of Information and Networks — 2014.
- 2 **Hojae L.** An Application of Data Leakage Prevention System based on Biometrics Signals Recognition Technology [Текст] / L. Hojae, J. Junkwon, K. Taeyoung, P. Minwoo, E. Jungho, T.M. Chung // SUComS — 2013.
- 3 **Yessir H.** Inside the Mind of the Insider: Towards Insider Threat Detection Using Psychophysiological Signals [Текст] / H. Yessir, T. Hasan, G. Mohammad, D. Ram // Journal of Internet Services and Information Security — 2016.
- 4 **Hassan T.** Prediction of Human Error Using Eye Movements Patterns for Unintentional Insider Threat Detection [Текст] / T. Hassan, H. Yessir, D. Ram // IEEE 4th International Conference on Identity, Security, and Behavior Analysis — 2018.
- 5 **Bergadano F.** Identity verification through dynamic keystroke analysis [Текст] / F. Bergadano, D. Gunneti, C. Picardi // Intelligence Data Analisys Journal 7 — 2003.
- 6 **Dowland P.** A Long-term trial of keystroke profiling using digraph, trigraph and keyword latencies [Текст] / P. Dowland, S. Furnell // In IFIP International Federation for Information Processing Journal — 2004.
- 7 **Joyce R.** Identity authentication based on keystroke latencies [Текст] / R. Joyce, G. Gupta // Commun. ACM 33 — 1990.
- 8 **Monrose F.** Keystroke dynamics as a biometric for authentication [Текст] / F. Monrose, A. D. Rubin // Future Generation Computing Systems 16 — 2000.
- 9 **Bender S.** Key sequence rhythm recognition system and method [Текст] / Bender, S. and Postley, H // (U.S. Patent № 7 206 938) — 2002.
- 10 **Kołakowska A.** Recognizing emotions on the basis of keystroke dynamics [Текст] / A. Kołakowska // 2015 8th International Conference on Human System Interaction (HSI) — 2015.

11 **Epp C.** Identifying Emotional States using Keystroke Dynamics [Текст] / C. Epp, M. Lippold, R. L. Mandryk // Proceedings of the International Conference on Human Factors in Computing Systems — 2011.

12 **Nazmul Haque F.M.** Identifying Emotion by Keystroke Dynamics And Text Pattern Analysis [Текст] / F.M. Nazmul Haque, J.M. Alam // Behaviour and Information Technology Journal — 2012.

13 **Suranga D.W.G.** Non-Invasive Human Stress Detection Using Key Stroke Dynamics and Pattern Variations [Текст] / D.W.G. Surang, M. De Silva Pasan, S.B.K. Dayan, M.K.D. Arunatileka Shiromi // International Conference on Advances in ICT for Emerging Regions (ICTer) — 2013.

14 **Kołakowska A.** Towards detecting programmers' stress on the basis of keystroke dynamics [Текст] / A. Kołakowska // Proceedings of the Federated Conference on Computer Science and Information Systems — 2016.

15 **Ulinskas M.** Recognition of human daytime fatigue using keystroke data [Текст] / M. Ulinskasa, R. Damaševičius, R. Maskeliūnas, M. Woźniak // The Workshop on Computational Intelligence in Ambient Assisted Living Systems (CIAALS 2018) — 2018.

16 **Yessir H.** Insider Threat Detection Based on Users' Mouse Movements and Keystrokes Behavior [Текст] / H. Yessir, T. Hassan, D. Ram // Conference Secure Knowledge Management Conference — 2017.

17 **CERT Definition of 'Insider Threat'** [Электронный ресурс] — Режим доступа: <https://insights.sei.cmu.edu/insider-threat/2017/03/cert-definition-of-insider-threat---updated.html> — Свободный. (Дата обращения: 25.03.2019)

18 **Insider Threat** [Электронный ресурс] — Режим доступа: <https://www.observeit.com/insider-threat/> — Свободный. (Дата обращения: 25.03.2019)

19 **Insider Threats as the Main Security Threat in 2017** [Электронный ресурс] — Режим доступа: <https://www.tripwire.com/state-of-security/security-data-protection/insider-threats-main-security-threat-2017/> — Свободный. (Дата обращения: 25.03.2019)

- 20 **Oladimeji T.O.** Review on Insider Threat Detection Techniques [Tekct] / T.O. Oladimeji, C.K. Ayo, S.E. Adewumi // 3rd International Conference on Science and Sustainable Development (ICSSD 2019) — 2019.
- 21 **David A.N.** Classifier Suites for Insider Threat Detection [Tekct] / A.N.David // Arxiv Journal, Machine Learning (cs.LG) — 2019.
- 22 **Adam J.H.** Predicting Malicious Insider Threat Scenarios Using Organizational Data and a Heterogeneous Stack-Classifier [Tekct] / J.H.Adam, P. Nikolas, J.B. William, M. Naghmeh // Arxiv Journal, Machine Learning (cs.LG) — 2019.
- 23 **Parveen P.** Supervised Learning for Insider Threat Detection Using Stream Mining [Tekct] / P. Parveen, R.Z. Weger, B. Thuraisingham, H. Kevin, K. Latifur // IEEE 23rd International Conference on Tools with Artificial Intelligence — 2019.
- 24 **Sarma M.S.** Insider Threat Detection with Face Recognition and KNN User Classification [Tekct] / M.S. Sarma, Y. Srinivas, M. Abhiram, L. Ullala, M.S. Prasanthi, J.R. Rao // IEEE International Conference on Cloud Computing in Emerging Markets (CCEM) — 2017.
- 25 **Anagi G.** Insider Threat Detection Through Attributed Graph Clustering [Tekct] / G. Anagi, B. Serdar // Proceedings of the 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications — 2017.
- 26 **Zhang J.** Insider threat detection of adaptive optimization DBN for behavior logs [Tekct] / J. Zhang, Y. Chen, J. Ankang // Turkish Journal of Electrical Engineering & Computer Sciences — 2017.
- 27 **Syarif I.** Unsupervised Clustering Approach for Network Anomaly Detection [Tekct] / I. Syarif, A. Prugel-Bennett, G. Wills // International Conference on Networked Digital Technologies — 2012 — C. 135-145.
- 28 **Carrasco R.** Unsupervised intrusion detection through skip-gram models of network behavior [Tekct] / R. Carrasco, M. Sicilia // Computers & Security — 2018 — C. 187-197.
- 29 **Aldairi M.** A Trust Aware Unsupervised Learning Approach For Insider Threat Detection [Tekct] / M. Aldairi, L. Karimi, J. Joshi // 2019 IEEE 20th

International Conference on Information Reuse and Integration for Data Science (IRI) — 2019.

30 **Tour A.** Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams [Текст] / A. Tour, S. Kaplan, B. Hutchinson // Proceedings of AI for Cyber Security Workshop at AAAI 2017 — 2017.

31 **Choi H.** Unsupervised learning approach for network intrusion detection system using autoencoders [Текст] / H. Choi, M. Kim, G. Lee, W. Kim // The Journal of Supercomputing volume 75 — 2019 — C. 5597–5621.

32 **Moskovitch R.** Identity Theft, Computers and Behavioral Biometrics [Текст] / R. Moskovitch, C. Feher, A. Messerman, N. Kirschnick, T. Mustafić, A. Camtepe, B. Löhlein, B. Heister, S. Möller, L. Rokach, Y. Elovici // 2016 9th International Conference on Electrical and Computer Engineering (ICECE) — 2016.

33 **Russian Letter Frequencies** [Электронный ресурс] — Режим доступа: <http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/russian-letter-frequencies/> — Свободный. (Дата обращения: 28.03.2019)

## Приложение А

### Логирование событий клавиатуры и мыши

```
from pyinput import keyboard
from pyinput import mouse
import logging

formatter = logging.Formatter('%(asctime)s|%(message)s')
path = "../data/"
listeners_started = False
record_events = False

def setup_logger(name, log_file, level=logging.INFO):
    # удаление всех старых обработчиков
    logger = logging.getLogger(name)
    for hdlr in logger.handlers[:]:
        logger.removeHandler(hdlr)

    # создание нового обработчика
    handler = logging.FileHandler(path + log_file)
    handler.setFormatter(formatter)

    # соединение обработчика и логера
    logger = logging.getLogger(name)
    logger.setLevel(level)
    logger.addHandler(handler)

    return logger
```

```
keyboard_logger = setup_logger("keyboard", "keyboard.csv")
mouse_logger = setup_logger("mouse", "mouse.csv")
```

```

# нажатие клавиши

def on_press(key):
    if record_events:
        keyboard_logger.info("press|{0}".format(key))

# отпускание клавиши

def on_release(key):
    if record_events:
        keyboard_logger.info("release|{0}".format(key))

# нажатие и отпускание кнопок мыши

def on_click(x, y, button, pressed):
    if record_events:
        if pressed:
            mouse_logger.info("press|{0}".format(button))
        else:
            mouse_logger.info("release|{0}".format(button))

keyboard_listener = keyboard.Listener(
    on_press=on_press,
    on_release=on_release)

mouse_listener = mouse.Listener(
    on_click=on_click)

def main():
    global listeners_started, record_events
    # информация об участнике
    name = input("Please, enter your name: ")

```

```

age = input("Please, enter your age: ")
gender = input("Please, enter your gender, m - male, f - female: ")

# сохранение информации об участнике
with open(path + "participants.csv", "a") as f:
    f.write(name + "|" + age + "|" + gender + "\n")

# меню сценариев
while True:
    print("1) Non-insider scenario\n2) Insider scenario\n3) Exit")
    choice = int(input("Your choice: "))
    if choice == 3:
        break
    else:
        keyboard_logger_filename = "keyboard_"
        mouse_logger_filename = "mouse_"
        if choice == 1:
            scenario = input("Enter non-insider scenario number: ")
            keyboard_logger_filename += name + "_non_insider_" + scenario + ".csv"
            mouse_logger_filename += name + "_non_insider_" + scenario + ".csv"
        else:
            scenario = input("Enter insider scenario number: ")
            keyboard_logger_filename += name + "_insider_" + scenario + ".csv"
            mouse_logger_filename += name + "_insider_" + scenario + ".csv"
        # логер для клавиатуры
        setup_logger("keyboard", keyboard_logger_filename)
        # логер для мыши
        setup_logger("mouse", mouse_logger_filename)
        # начать записи
        record_events = True

```

```
# слушатели запускаются один раз за всю сессию работы скрипта
if listeners_started == False:
    keyboard_listener.start()
    mouse_listener.start()
    listeners_started = True
stop_scenario = input("Enter S to stop scenario: ")
if stop_scenario == "S": # остановить логирование и перейти в меню
    сценариев
record_events = False

main()
```

## Приложение Б

### Выделение частотных и временных признаков из файлов логирования

```
import pandas as pd
import copy
from time import mktime
from datetime import datetime as dt

# кнопки мыши
mouse_buttons = [
    "Button.left",
    "Button.right",
]

# специальные клавиши
special_keys = [
    "Key.escape",
    "Key.tab",
    "Key.caps_lock",
    "Key.shift",
    "Key.ctrl",
    "Key.alt",
    "Key.cmd",
    "Key.space",
    "Key.enter",
    "Key.backspace",
]

# диграфы и триграфы - сочетания из 2-х и 3-х букв
eng_di = ["th", "he", "in", "er", "an", "re", "es", "on", "st", "nt", "en", "at", "ed", "nd",
          "to", "or", "ea"]
```

```

eng_tri = ["the", "and", "ing", "ent", "ion", "her", "for", "tha", "nth", "int", "ere",
"tio", "ter", "est", "ers", "ati", "hat"]

ru_di = ["ст","ен","ов","но","ни","на","па","ко","то","ро"] #
,"ан","ос","по","го","ер","од", "ре"]

ru_tri = ["ени","ост","ого","ств","ско","ста","ани","про","ест","топ"] #
,"льн","ова","ния","ние","при","енн","год"]

# признаки для специальных признаков и кнопок мыши
spec_features = {

    "dwell": [0, 0], # длительность нажатия

    "interval" : [0, 0], # промежуток между отпусканием текущей и нажатием
    следующей

    "flight": [0,0] # промежуток между нажатием текущей и нажатием следующей
}

# признаки для диграфов
di_features = {

    "dwell_first": [0,0], # длительность нажатия первой буквы

    "dwell_second": [0,0], # длительность нажатия второй буквы

    "interval": [0,0], # промежуток между отпусканием первой и нажатием второй
    буквы

    "flight": [0,0], # промежуток между нажатием первой и нажатием второй
    буквы

    "up_to_up": [0,0], # промежуток между отпусканием первой и отпусканием
    второй

    "latency": [0,0], # промежуток между нажатием первой и отпусканием второй
}

# признаки для триграфов
tri_features = {

```

```

    "dwell_first": [0,0], # длительность нажатия первой буквы
    "dwell_second": [0,0], # длительность нажатия второй буквы
    "dwell_third": [0,0], # длительность нажатия третьей буквы
    "interval_first": [0,0], # промежуток между отпусканием первой и нажатием
    второй буквы
    "interval_second": [0,0], # промежуток между отпусканием второй и нажатием
    третьей буквы
    "flight_first": [0,0], # промежуток между нажатием первой и нажатием второй
    буквы
    "flight_second": [0,0], # промежуток между нажатием второй и нажатием
    третьей буквы
    "up_to_up_first": [0,0], # промежуток между отпусканием первой и
    отпусканием второй
    "up_to_up_second": [0,0], # промежуток между отпусканием второй и
    отпусканием третьей
    "latency": [0,0], # промежуток между нажатием первой и отпусканием третьей
}

```

# здесь будут храниться все вышеприведённые признаки  
features = {}

```

class FeatureProcessor:

    # вычисление признаков для кнопок мыши и униграфов
    def mouse_and_special_keys(self, df, events):
        for event in events:
            if event not in features: # в словаре признаков пока не выделена память
                под эту кнопку
                    features[event] = copy.deepcopy(spec_features) # выделяем память
            for i, row in df.iterrows():
                if row[1] == "press" and row[2] == event:

```

```
press = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f') #
```

фиксируем время нажатия кнопки

```
# время отпускания нажатой кнопки
```

```
# по умолчанию задаётся значением press,
```

```
# так как release для последнего события в датасете может
```

отсутствовать

```
release = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f')
```

```
# время нажатия кнопки после отпускания текущей
```

```
# по умолчанию задаётся значением press,
```

```
# так как press следующей кнопки для последнего события в
```

датасете может отсутствовать

```
press_next = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f')
```

```
j = 1
```

```
while True: # ищем время отпускания кнопки
```

```
try: # пробуем обратиться по индексу i + j
```

```
next_row = df.iloc[i + j]
```

```
if next_row[1] == "release" and next_row[2] == event:
```

```
# фиксируем время отпускания текущей кнопки
```

```
release = dt.strptime(next_row[0], '%Y-%m-%d
```

```
%H:%M:%S.%f')
```

```
try: # пробуем обратиться по индексу i + j + 1
```

```
# фиксируем время нажатия следующей кнопки
```

```
press_next = dt.strptime(df.iloc[i + j + 1][0], '%Y-%m-%d
```

```
%H:%M:%S.%f')
```

```
except IndexError: # в случае отсутствия такого индекса, т.е. в
```

случае выхода за границу датафрейма

```
# фиксируем время нажатия следующей кнопки, как время
```

отпускания текущей

```
press_next = dt.strptime(next_row[0], '%Y-%m-%d
```

```
%H:%M:%S.%f')
```

```

        finally:

            # выходим из вспомогательного цикла в любом случае, так
как release был найден

                break

            else:

                j+=1 # переход к следующей строке

            except IndexError:

                # выходим из вспомогательного цикла

                break

            # считаем признаки

            # dwell

            features[event]["dwell"][0] += (release - press).microseconds // 1000 #

прибавляем длительность

            features[event]["dwell"][1] += 1 # увеличиваем количество

обработанных кнопок event

            # interval

            features[event]["interval"][0] += (press_next - release).microseconds //

1000 # прибавляем длительность

            features[event]["interval"][1] += 1 # увеличиваем количество

обработанных кнопок event

            # flight

            features[event]["flight"][0] += (press_next - press).microseconds // 1000

# прибавляем длительность

            features[event]["flight"][1] += 1 # увеличиваем количество

обработанных кнопок event

# вычисление признаков для диграфов

def digraph_features(self, df, events):

    for event in events:

```

```
if event not in features: # в словаре признаков пока не выделена память под эту кнопку
```

```
    features[event] = copy.deepcopy(di_features) # выделяем память  
    k = 0 # отвечает за индекс текущего символа в диграфе  
    first_press = ""  
    first_release = ""  
    for i, row in df.iterrows():  
        if row[1] == "press":  
            if row[2][1:-1].lower() == event[k]:  
                press = dt.strptime(row[0], "%Y-%m-%d %H:%M:%S,%f") #
```

фиксируем время нажатия кнопки

```
# время отпускания нажатой кнопки  
# по умолчанию задаётся значением press,  
# так как release для последнего события в датасете может
```

отсутствовать

```
release = dt.strptime(row[0], "%Y-%m-%d %H:%M:%S,%f")  
# время нажатия кнопки после отпускания текущей  
# по умолчанию задаётся значением press,  
# так как press следующей кнопки для последнего события в  
датасете может отсутствовать
```

```
press_next = dt.strptime(row[0], "%Y-%m-%d %H:%M:%S,%f")  
j = 1  
while True: # ищем время отпускания кнопки  
    try: # пробуем обратиться по индексу i + j  
        next_row = df.iloc[i + j]  
        if next_row[1] == "release" and next_row[2][1:-1] == event[k]:  
            # фиксируем время отпускания текущей кнопки  
            release = dt.strptime(next_row[0], "%Y-%m-%d  
%H:%M:%S,%f")
```

```
try: # пробуем обратиться по индексу i + j + 1
```

```

        # фиксируем время нажатия следующей кнопки
        press_next = dt.strptime(df.iloc[i + j + 1][0], '%Y-%m-%d
%H:%M:%S,%f')

    except IndexError: # в случае отсутствия такого индекса,
        т.е. в случае выхода за границу датафрейма

        # фиксируем время нажатия следующей кнопки, как
        время отпускания текущей

        press_next = dt.strptime(next_row[0], '%Y-%m-%d
%H:%M:%S,%f')

    finally:

        # выходим из вспомогательного цикла в любом случае,
        так как release был найден

        break

    else:
        j+=1 # переход к следующей строке

except IndexError:
    # выходим из вспомогательного цикла
    break

if k == 0:
    # сохраняем момент нажатия и отпускания первой клавиши
    first_press = press
    first_release = release
    k+=1

else:
    # считаем признаки
    # dwell_first
    features[event]["dwell_first"][0] += (first_release -
first_press).microseconds // 1000 # прибавляем длительность
    features[event]["dwell_first"][1] += 1 # увеличиваем количество
    обработанных кнопок event

```

```

# interval
    features[event]["interval"][0] += (press -
first_release).microseconds // 1000 # прибавляем длительность
    features[event]["interval"][1] += 1 # увеличиваем количество
обработанных кнопок event

# flight
    features[event]["flight"][0] += (press - first_press).microseconds //
1000 # прибавляем длительность
    features[event]["flight"][1] += 1 # увеличиваем количество
обработанных кнопок event

# dwell_second
    features[event]["dwell_second"][0] += (release -
press).microseconds // 1000 # прибавляем длительность
    features[event]["dwell_second"][1] += 1 # увеличиваем
количество обработанных кнопок event

# up_to_up
    features[event]["up_to_up"][0] += (release -
first_release).microseconds // 1000 # прибавляем длительность
    features[event]["up_to_up"][1] += 1 # увеличиваем количество
обработанных кнопок event

# latency
    features[event]["latency"][0] += (release - first_press).microseconds
// 1000 # прибавляем длительность
    features[event]["latency"][1] += 1 # увеличиваем количество
обработанных кнопок event

k = 0
first_press = ""
first_release = ""

else:
    k = 0

```

```

first_press = ""
first_release = ""

# вычисление признаков для триграфов
def trigraph_features(self, df, events):
    for event in events:
        if event not in features: # в словаре признаков пока не выделена память
            под эту кнопку
                features[event] = copy.deepcopy(tri_features) # выделяем память
                k = 0 # отвечает за индекс текущего символа в диграфе
                first_press = ""
                first_release = ""
                second_press = ""
                seconds_release = ""
                for i, row in df.iterrows():
                    if row[1] == "press":
                        if row[2][1:-1].lower() == event[k]:
                            press = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f') #
фиксируем время нажатия кнопки
                            # время отпускания нажатой кнопки
                            # по умолчанию задаётся значением press,
                            # так как release для последнего события в датасете может
                            отсутствовать
                            release = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f')
                            # время нажатия кнопки после отпускания текущей
                            # по умолчанию задаётся значением press,
                            # так как press следующей кнопки для последнего события в
                            датасете может отсутствовать
                            press_next = dt.strptime(row[0], '%Y-%m-%d %H:%M:%S.%f')
                            j = 1

```

```

while True: # ищем время отпускания кнопки

    try: # пробуем обратиться по индексу i + j

        next_row = df.iloc[i + j]

        if next_row[1] == "release" and next_row[2][1:-1] == event[k]:

            # фиксируем время отпускания текущей кнопки

            release = dt.strptime(next_row[0], '%Y-%m-%d
%H:%M:%S,%f')

            try: # пробуем обратиться по индексу i + j + 1

                # фиксируем время нажатия следующей кнопки

                press_next = dt.strptime(df.iloc[i + j + 1][0], '%Y-%m-%d
%H:%M:%S,%f')

            except IndexError: # в случае отсутствия такого индекса,
т.е. в случае выхода за границу датафрейма

                # фиксируем время нажатия следующей кнопки, как
время отпускания текущей

                press_next = dt.strptime(next_row[0], '%Y-%m-%d
%H:%M:%S,%f')

            finally:

                # выходим из вспомогательного цикла в любом случае,
так как release был найден

                break

        else:

            j+=1 # переход к следующей строке

    except IndexError:

        # выходим из вспомогательного цикла

        break

    if k == 0:

        # сохраним момент нажатия и отпускания первой клавиши

        first_press = press

        first_release = release

```

```

k+=1

elif k == 1:
    # сохраняем момент нажатия и отпускания первой клавиши
    second_press = press
    second_release = release
    k+=1

else:
    # считаем признаки
    # dwell_first
    features[event]["dwell_first"][0] += (first_release -
first_press).microseconds // 1000 # прибавляем длительность
    features[event]["dwell_first"][1] += 1 # увеличиваем количество
обработанных кнопок event

    # interval_first
    features[event]["interval_first"][0] += (second_press -
first_release).microseconds // 1000 # прибавляем длительность
    features[event]["interval_first"][1] += 1 # увеличиваем
количество обработанных кнопок event

    # flight_first
    features[event]["flight_first"][0] += (second_press -
first_press).microseconds // 1000 # прибавляем длительность
    features[event]["flight_first"][1] += 1 # увеличиваем количество
обработанных кнопок event

    # up_to_up_first
    features[event]["up_to_up_first"][0] += (second_release -
first_release).microseconds // 1000 # прибавляем длительность
    features[event]["up_to_up_first"][1] += 1 # увеличиваем
количество обработанных кнопок event

    # dwell_second

```

```

        features[event]["dwell_second"][0] += (second_release -
second_press).microseconds // 1000 # прибавляем длительность

        features[event]["dwell_second"][1] += 1 # увеличиваем
количество обработанных кнопок event

# interval_second

        features[event]["interval_second"][0] += (press -
second_release).microseconds // 1000 # прибавляем длительность

        features[event]["interval_second"][1] += 1 # увеличиваем
количество обработанных кнопок event

# flight_second

        features[event]["flight_second"][0] += (press -
second_press).microseconds // 1000 # прибавляем длительность

        features[event]["flight_second"][1] += 1 # увеличиваем
количество обработанных кнопок event

# up_to_up_second

        features[event]["up_to_up_second"][0] += (release -
second_release).microseconds // 1000 # прибавляем длительность

        features[event]["up_to_up_second"][1] += 1 # увеличиваем
количество обработанных кнопок event

# dwell_third

        features[event]["dwell_third"][0] += (release - press).microseconds
// 1000 # прибавляем длительность

        features[event]["dwell_third"][1] += 1 # увеличиваем количество
обработанных кнопок event

# latency

        features[event]["latency"][0] += (release - first_press).microseconds
// 1000 # прибавляем длительность

        features[event]["latency"][1] += 1 # увеличиваем количество
обработанных кнопок event

k = 0

```

```

        first_press = ""
        first_release = ""
        second_press = ""
        second_release = ""

    else:

        k = 0

        first_press = ""
        first_release = ""
        second_press = ""
        second_release = ""

# расчёт средних показателей и запись в файл с признаками

def calc_average(self, is_insider, duration, filename):

    averaged = [] # список для хранения усреднённых признаков

    # расчёт средних показателей

    for k, v in features.items():

        if k in special_keys: # для специальных признаков кроме временных
            # показателей

            averaged.append([k, value[1] / duration]) # сохраняются ещё и
            # частотные показатели

        for key, value in v.items():

            averaged.append([k + "_" + key, round(value[0] / value[1], 2) if value[1] != 0 else 0])

# открываем файл с признаками для проверки существования заголовков

with open(filename, "r") as f:
    a = f.readlines()
    f.close()

```

```

# открываем файл с признаками для записи заголовков и последующей
записи признаков

with open(filename, "a") as f:
    if len(a) == 0: # в файле признаков нет заголовка
        headers = "" # зашоловки, объединённые в строку
    for i in range(len(averaged)):
        if i != len(averaged) - 1:
            headers += averaged[i][0] + "|"
        else: # не добавляем "|" для последнего заголовка
            headers += averaged[i][0] + "|is_insider\n"
    f.write(headers)

averaged_features = "" # усреднённые признаки, объединённые в строку
for i in range(len(averaged)):
    if i != len(averaged) - 1:
        averaged_features += str(averaged[i][1]) + "|"
    else: # не добавляем "|" для последнего заголовка
        averaged_features += str(averaged[i][1]) + "|" + str(1 if is_insider else 0)
        + "\n"
    f.write(averaged_features)

pt = pd.read_csv("../data/participants.csv", sep="|", header=None)
fp = FeatureProcessor()
# подсчёт признаков для сценариев без и с внутренними нарушителями
for i in ["non_insider", "insider"]:
    # пробегаемся по списку участников
    for _, row in pt.iterrows():
        # для каждого из 3-х сценариев
        for j in range(1,4):
            # очистка словаря для хранения признаков

```

```

features.clear()

# считывание данных

kb = pd.read_csv('..../data/keyboard_' + row[0] + '_' + i + '_' + str(j) + '.csv',
sep='|', header=None)

ms = pd.read_csv('..../data/mouse_' + row[0] + '_' + i + '_' + str(j) + '.csv',
sep='|', header=None)

# расчёт времени выполнения сценария в минутах

start = mktime(dt.strptime(kb.iloc[0][0], '%Y-%m-%d
%H:%M:%S,%f').timetuple())

end = mktime(dt.strptime(kb.iloc[-1][0], '%Y-%m-%d
%H:%M:%S,%f').timetuple())

duration = (end - start) / 60

# подсчёт признаков

fp.mouse_and_special_keys(ms, mouse_buttons)

fp.mouse_and_special_keys(kb, special_keys)

fp.digraph_features(kb, ru_di)

# fp.digraph_features(kb, eng_di)

fp.trigraph_features(kb, ru_tri)

# fp.trigraph_features(kb, eng_tri)

if k > 4 and j == 3:

    fp.calc_average(False if i == "non_insider" else True, duration,
"../data/test_features.csv")

else:

    fp.calc_average(False if i == "non_insider" else True, duration,
"../data/train_features.csv")

```

## Приложение В

### Предобработка и выделение наиболее информативных признаков

```
import pandas as pd
import numpy as np
import math

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, chi2, f_classif,
mutual_info_classif

df = pd.read_csv("../data/train_features.csv", sep="|")
test = pd.read_csv("../data/test_features.csv", sep="|")

threshold = 1

# удаляем столбцы с маленьким стандартным отклонением
cols = df[df['is_insider'] == 1].std()[df[df['is_insider'] == 1].std() <
threshold].index.values.tolist()

# не трогаем таргет столбец
cols.remove("is_insider")
df.drop(cols, axis=1, inplace=True)

# разделение датасета на обучаемую и тестовую выборки
# для алгоритмов классификации
y = df['is_insider']
x_train, x_test, y_train, y_test = train_test_split(df, y, random_state = 0, test_size =
0.3)
X_test = test
Y_test = test['is_insider']

medians = {} # словарь для хранения медиан для каждого столбца
```

```

# заполнение пустот в обучаемой выборке с помощью медиан

# и их использование для заполнения пустот в валидационной выборке

for col in x_train.columns:

    if col != 'is_insider':

        # медианы

        medians[col + "_non_insider"] = x_train.loc[(x_train[col] != 0) &
(x_train['is_insider'] == 0), col].median()

        medians[col + "_is_insider"] = x_train.loc[(x_train[col] != 0) &
(x_train['is_insider'] == 1), col].median()

        # заполнение пустот в обучаемой выборке с помощью медиан из
обучаемой выборки

        x_train.loc[(x_train[col] == 0) & (x_train['is_insider'] == 0), col] = medians[col +
"_non_insider"] if medians[col + "_non_insider"] is not np.nan else 0

        x_train.loc[(x_train[col] == 0) & (x_train['is_insider'] == 1), col] = medians[col +
"_is_insider"] if medians[col + "_is_insider"] is not np.nan else 0

        # заполнение пустот в валидационной выборке с помощью медиан из
обучаемой выборки

        x_test.loc[(x_test[col] == 0) & (x_test['is_insider'] == 0), col] = medians[col +
"_non_insider"] if medians[col + "_non_insider"] is not np.nan else 0

        x_test.loc[(x_test[col] == 0) & (x_test['is_insider'] == 1), col] = medians[col +
"_is_insider"] if medians[col + "_is_insider"] is not np.nan else 0

        # заполнение пустот в тестовой выборке с помощью медиан из обучаемой
выборки

        X_test.loc[(X_test[col] == 0) & (X_test['is_insider'] == 0), col] = medians[col +
"_non_insider"] if medians[col + "_non_insider"] is not np.nan else 0

        X_test.loc[(X_test[col] == 0) & (X_test['is_insider'] == 1), col] = medians[col +
"_is_insider"] if medians[col + "_is_insider"] is not np.nan else 0

# очистка памяти

del medians

```

```

# заполнение пустот медианами

medians_insider = []
medians_non_insider = []
cols = []
special_keys = [
    "Key.escape",
    "Key.tab",
    "Key.caps_lock",
    "Key.shift",
    "Key.ctrl",
    "Key.alt",
    "Key.cmd",
    "Key.space",
    "Key.enter",
    "Key.backspace",
]
for col in x_train.columns:
    # не считаем медианы для таргета и частотных признаков
    if col != 'is_insider' and col not in special_keys:
        cols.append(col)
        medians_insider.append(x_train[x_train['is_insider'] == 1][col].median())
        medians_non_insider.append(x_train[x_train['is_insider'] == 0][col].median())

# разделение данных на обучаемую и тестовую выборки
# для алгоритмов обнаружения аномалий

x_train_anomaly = pd.concat([x_train[x_train['is_insider'] == 0],
                             x_test[x_test['is_insider'] == 0]])
y_train_anomaly = pd.concat([y_train[y_train == 0], y_test[y_test == 0]])

```

```
x_test_anomaly = pd.concat([x_train[x_train['is_insider'] == 1],  
x_test[x_test['is_insider'] == 1],  
X_test[X_test['is_insider'] == 1], X_test[X_test['is_insider'] == 0]])  
y_test_anomaly = pd.concat([y_train[y_train == 1],  
y_test[y_test == 1], Y_test[Y_test == 1], Y_test[Y_test == 0]])  
len(x_train_anomaly.index), len(y_train_anomaly.index), len(x_test_anomaly.index),  
len(y_test_anomaly.index)
```

```
# удаление столбца is_insider из x_train и x_test  
x_train.drop(columns = ['is_insider'], inplace = True)  
x_test.drop(columns = ['is_insider'], inplace = True)  
X_test.drop(columns = ['is_insider'], inplace = True)  
x_train_anomaly.drop(columns = ['is_insider'], inplace = True)  
x_test_anomaly.drop(columns = ['is_insider'], inplace = True)
```

```
# выделение признаков  
transformer = SelectKBest(chi2, k = 3)  
X_new = transformer.fit_transform(x_train, y_train)
```

```
selected_columns = x_train.columns[transformer.get_support()].values
```

```
x_train = x_train[selected_columns]  
x_test = x_test[selected_columns]  
X_test = X_test[selected_columns]
```

```
x_train_anomaly = x_train_anomaly[selected_columns]  
x_test_anomaly = x_test_anomaly[selected_columns]
```

## Приложение Г

### Модели классификаторов

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

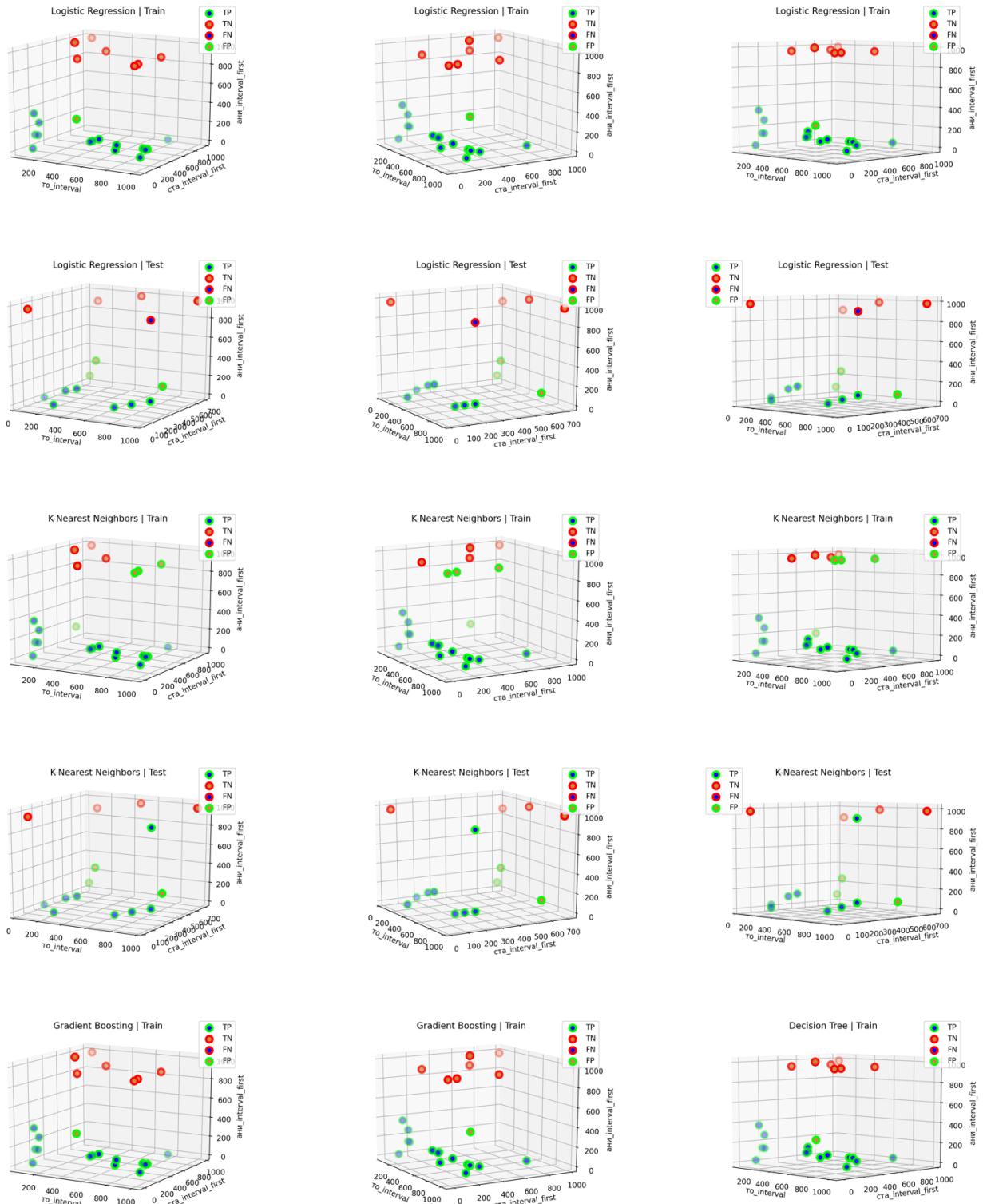
# расчёт метрик
def report(clf, x_train, y_train, x_test, y_test, X_test, Y_test):
    y_pred = clf.fit(x_train, y_train).predict(x_test)
    y_pred_train = clf.predict(x_train)
    Y_pred = clf.predict(X_test)
    print("accuracy train:", clf.score(x_train,y_train), "accuracy validation",
accuracy_score(y_test, y_pred), "accuracy_test", accuracy_score(Y_test, Y_pred))
    print(classification_report(y_train, clf.predict(x_train)))
    print(classification_report(y_test, y_pred))
    print(classification_report(Y_test, Y_pred))

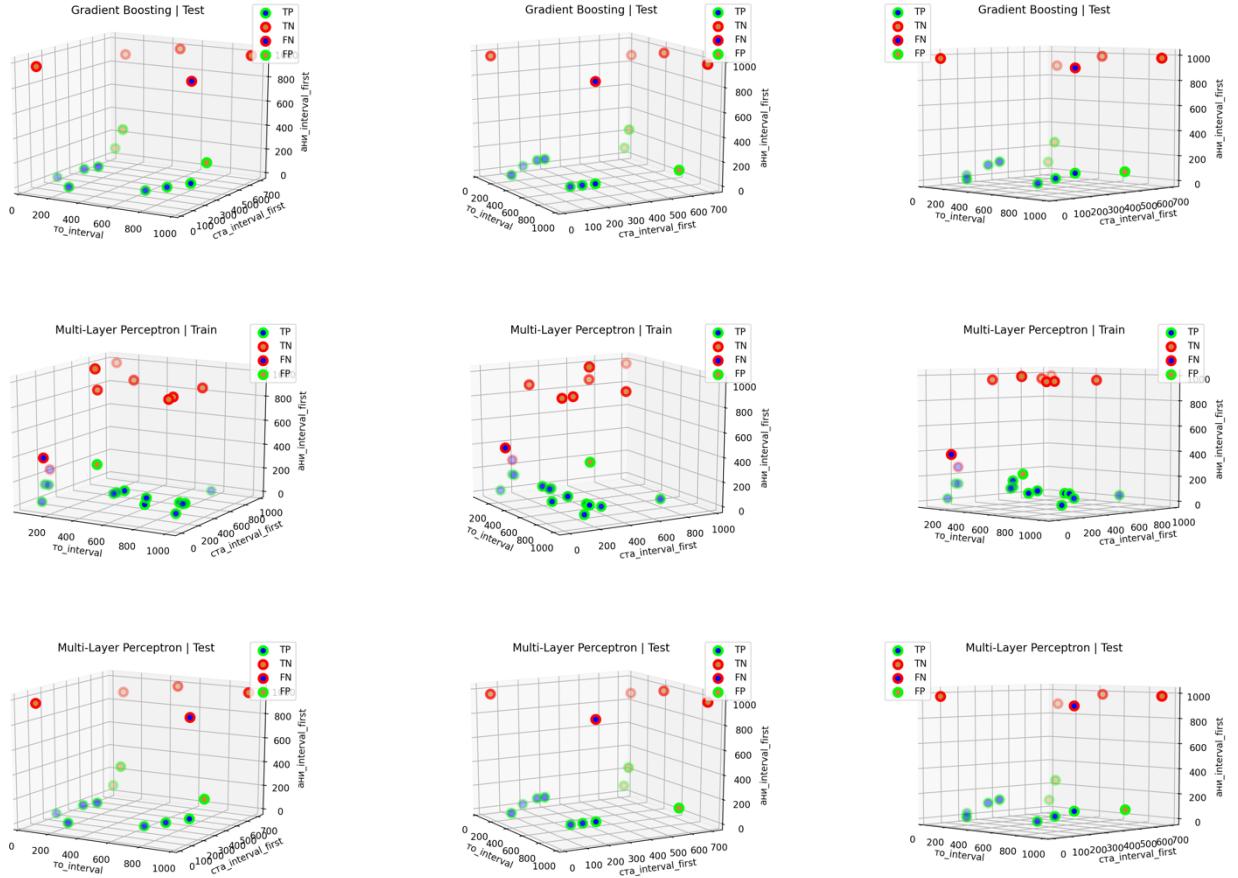
# модели
models = {
    'Logistic Regression': LogisticRegression(solver='lbfgs', random_state=0),
    'K-Nearest Neighbors': KNeighborsClassifier(n_neighbors=3),
    'Random Forest': RandomForestClassifier(random_state=0),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=500, max_depth=2,
random_state=0),
    'Multi-Layer Perceptron': MLPClassifier(hidden_layer_sizes=(512, 256, 128, 64,
32), alpha=0.001, max_iter=10000),
}
```

```
for k,v in models.items():
    print(k)
    report(v, x_train, y_train, x_test, y_test, X_test, Y_test)
```

## Приложение Д

### Визуализированные результаты моделей классификаторов





## Приложение E

### Модели обнаружения аномалий

```
from sklearn import svm
from sklearn.covariance import EllipticEnvelope
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

n_samples = len(x_train_anomaly)
outliers_fraction = 0.15
n_outliers = int(outliers_fraction * n_samples)
n_inliers = n_samples - n_outliers
LOF_novelty = False

# модели
anomaly_algorithms = {
    "Robust Covariance": EllipticEnvelope(contamination=outliers_fraction),
    "One-Class SVM": svm.OneClassSVM(nu=outliers_fraction, kernel="rbf",
                                      gamma=0.1),
    "Isolation Forest": IsolationForest(contamination=outliers_fraction,
                                         random_state=42),
    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=35,
                                                contamination=outliers_fraction,
                                                novelty=LOF_novelty)
}

y_train_anomaly = np.full(len(y_train_anomaly.index), 1)
y_test_anomaly = np.where(y_test_anomaly == 1, -1, 1)

for name, algorithm in anomaly_algorithms.items():
    algorithm.fit(x_train_anomaly)
```

```

if name == "Local Outlier Factor":
    if not LOF_novelty:
        y_pred = algorithm.fit_predict(x_train_anomaly)
    else:
        y_pred_test = algorithm.predict(x_test_anomaly)
else:
    y_pred = algorithm.predict(x_train_anomaly)
    y_pred_test = algorithm.predict(x_test_anomaly)

print(name)
print("*** TRAIN ***")
print("pred:", y_pred)
print("train:", y_train_anomaly)
print("REPORT:", classification_report(y_train_anomaly, y_pred))

print("*** TEST ***")
print("pred:", y_pred_test)
print("test:", y_test_anomaly)
print("REPORT:", classification_report(y_test_anomaly, y_pred_test))
print()

```

## Приложение Ж

### Визуализированные результаты моделей обнаружения аномалий

