ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра

«Криптология и кибербезопасность»

Отчет

о научно-исследовательской работе

«Обнаружение угрозы внутреннего нарушителя путём выявления стрессового состояния пользователя на основе анализа взаимодействия с клавиатурой и мышью с применением алгоритмов машинного обучения»

Исполнитель:

студент гр. Б16-506 _____ Султанов А.Э.

(подпись, дата)

Научный руководитель: _____ Когос К.Г.

(подпись, дата)

Зам. зав. каф. № 42: _____ Когос К.Г.

(подпись, дата)

Москва – 2019

РЕФЕРАТ

Отчёт 47 с., 3 рис., 4 табл., 5 прил., 29 источников.

ВНУТРЕННИЙ НАРУШИТЕЛЬ, ВЫЯВЛЕНИЕ СТРЕССА, ДИНАМИКА НАЖАТИЯ КЛАВИШ, МАШИННОЕ ОБУЧЕНИЕ.

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

Цель работы — оценить возможность обнаружения угрозы внутреннего нарушителя путём выявления стрессового состояния пользователя на основе анализа взаимодействия с клавиатурой и мышью.

Для оценки данной возможности проведён анализ существующих работ по обнаружению угрозы 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 на основе различных алгоритмов машинного обучения с применением информации взаимодействия пользователя с клавиатурой и мышью. При этом сделан положительный вывод о возможности обнаружения угрозы внутреннего нарушителя путём выявления стресса на основе данных динамики использования клавиатуры и мыши.

СОДЕРЖАНИЕ

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).

AB — метод адаптивного бустинга (Adaptive Boosting).

GB — метод градиентного бустинга (Gradient Boosting).

DT — дерево принятия решений (Decision Tree).

LR — логистическая регрессия (Logistic Regression).

ROC — рабочая характеристика приёмника (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)

NB — наивный байесовский классификатор (Naive Bayes)

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 различных алгоритмов машинного обучения, приведены результаты применённых 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 две основные категории алгоритмов машинного обучения:

- методы классификации из группы алгоритмов с учителем;
- методы кластеризации 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 траффике 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 немного 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 работе [25] обнаружение угрозы внутреннего нарушителя реализовано с применением предварительной кластеризации данных на основе графов и последующем выделении аномалий. В качестве датасета использована база данных CERT. При оценке модели наилучший результат показателя AUC составил 0.76.

Ещё один метод кластеризации был применён в статье [26]. Модель основана на одной из разновидностей глубоких нейронных сетей DBN. В качестве датасета выбран общедоступный CERT, однако модель обучена только на данных логов поведения пользователей. 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] авторы представили систему обнаружения угрозы внутреннего нарушителя из больших потоков данных в режиме реального времени, основанную на глубоких и рекуррентных нейронных сетях. Для уменьшения размерности пространства признаков использовался алгоритм PCA. Модели оценивались с помощью IF и SVM. В результате как для модели на основе DNN, так и для модели на основе RNN получен показатель полноты в 100%.

1.3 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, температура 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 BA. 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.

2 Процесс накопления данных

По причине отсутствия датасетов для текущего исследования возникла необходимость организации сбора данных и создания датасета для последующего обучения моделей классификаторов. Для этого были придуманы сценарии двух категорий (Приложение А), описывающие действия внутреннего нарушителя и действия обычного сотрудника.

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.

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 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 Б).

3 Выделение признаков и их предобработка

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

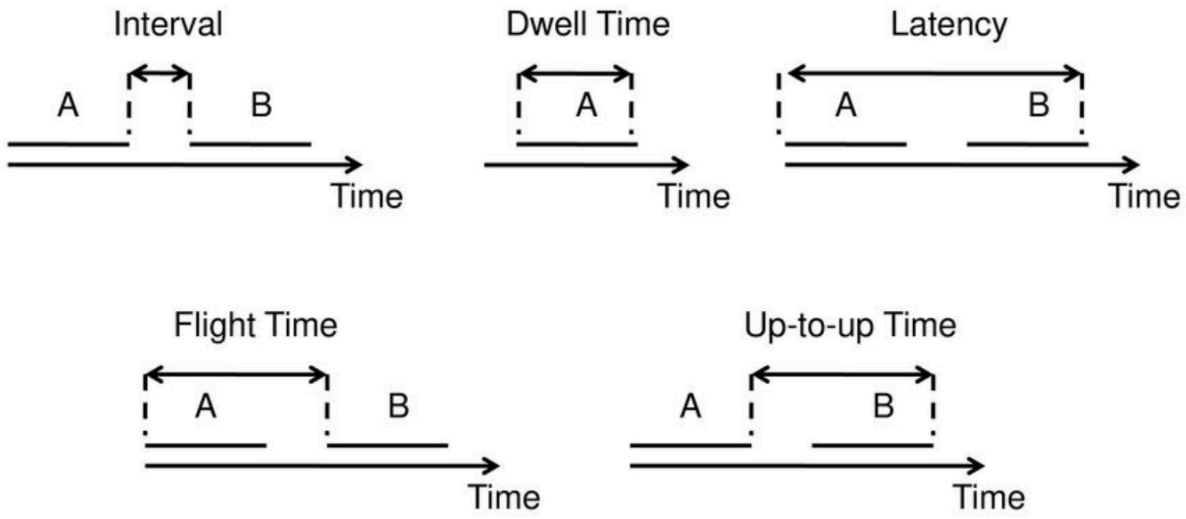

Рисунок 1 — схематическое представление временных показателей [28]

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

- нажатия кнопок мыши;
- специальные клавиши такие, как backspace, del, capslock, shift, tab, alt, esc;
- 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 встречаемыми в русском языке. Для подсчёта частоты диграфов и триграфов было использовано около 275 миллионов символов текста на русском языке [29].

Как частотные, так и временные признаки были посчитаны с помощью написанного на языке питон скрипта (Приложение Г). 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 было 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.

В силу того, что во всех сценариях наборы встречающихся элементов из вышеупомянутых групп разные, в итоговом датасете появились 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 Д).

4 Модели классификаторов и их оценка

Для построения моделей классификаторов были выбраны алгоритмы LR, k-NN, NB, SVM, DT, RF, MLP и два алгоритмов бустинга AB и GB. Построенные модели были обучены на 49% всего датасета, прошли валидацию на 21% датасета и были протестированы на оставшейся части датасета.

Сначала классификаторы были обучены на признаках, рассчитанных только для событий мыши. Результаты оценок моделей классификаторов, обученных на данных признаках, приведены в таблице 1.

Таблица 1 — Оценка моделей классификаторов (события мыши)

№	Алгоритм	Precision Non-Insider (train / test)	Precision Insider (train / test)	Recall Non-Insider (test / test)	Recall Insider (test / test)	Accuracy (train / test)
1	LR	0.83 / 0.33	0.90 / 0.40	0.91 / 0.25	0.82 / 0.50	0.86 / 0.38
2	k-NN	0.79 / 0.50	1.00 / 0.50	1.00 / 0.50	0.73 / 0.50	0.86 / 0.50
3	NB	0.50 / 0.50	0.00 / 0.00	1.00 / 1.00	0.00 / 0.00	0.50 / 0.50
4	SVM	0.73 / 0.44	1.00 / 0.43	1.00 / 0.50	0.64 / 0.38	0.81 / 0.44
5	DT	1.00 / 0.50	1.00 / 0.50	1.00 / 0.25	0.82 / 0.75	1.00 / 0.50
6	RF	0.85 / 0.50	1.00 / 0.50	1.00 / 0.38	0.82 / 0.62	0.91 / 0.50
7	MLP	0.53 / 0.46	0.60 / 0.33	0.82 / 0.75	0.27 / 0.12	0.55 / 0.44
8	AB	1.00 / 1.00	1.00 / 0.57	1.00 / 0.25	1.00 / 1.00	1.00 / 0.63
9	GB	1.00 / 1.00	1.00 / 0.57	1.00 / 0.25	1.00 / 1.00	1.00 / 0.63

По данным таблицы 1 видно, что модели классификаторов, обученных только на признаках, выделенных для событий мыши, недообучились, так как на новых данных они делают предсказания случайным образом. Это объясняется тем, что признаки, которые могли характеризовать тестовые данные, не были взяты во внимание, что говорит о недостаточности использования признаков, выделенных для событий мыши, или иными словами, при выполнении сценариев, из которых были отобраны тестовые данные, мышь использовалась

очень редко по сравнению с клавиатурой, то есть зависимость поведения и динамики использования мыши очень слаба. Именно по этой причине оценки моделей классификаторов на тестовых данных очень низкие.

В ряде следующих моделей классификаторы обучаются на комбинированном наборе признаков, которые были выделены как для событий мыши, так и для событий клавиатуры.

В таблице 2 представлены результаты оценок моделей, обученных на признаках, использованных для предыдущих моделей, и признаках, рассчитанных для специальных клавиш.

Таблица 2 — Оценка моделей классификаторов (события мыши в сочетании со специальными клавишами)

№	Алгоритм	Precision Non-Insider (train / test)	Precision Insider (train / test)	Recall Non-Insider (test / test)	Recall Insider (test / test)	Accuracy (train / test)
1	LR	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00
2	k-NN	0.91 / 1.00	0.91 / 1.00	0.91 / 1.00	0.91 / 1.00	0.91 / 1.00
3	NB	0.50 / 0.50	0.00 / 0.00	1.00 / 1.00	0.00 / 0.00	0.50 / 0.50
4	SVM	1.00 / 0.78	1.00 / 0.86	1.00 / 0.88	0.64 / 0.75	1.00 / 0.81
5	DT	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	0.82 / 0.75	1.00 / 0.88
6	RF	1.00 / 0.89	1.00 / 1.00	1.00 / 1.00	1.00 / 0.88	1.00 / 0.94
7	MLP	1.00 / 0.89	1.00 / 1.00	1.00 / 1.00	1.00 / 0.88	0.55 / 0.94
8	AB	1.00 / 0.89	1.00 / 1.00	1.00 / 1.00	1.00 / 0.88	1.00 / 0.94
9	GB	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	0.82 / 0.75	1.00 / 0.88

По данным, представленным в таблице 2, видно, что большинство моделей, обученных на новом наборе признаков, существенно улучшили результаты предсказаний как для обучаемой, так и для тестовой выборки. Данные улучшения могут свидетельствовать о том, что признаки, рассчитанные для событий клавиатуры намного более информативны, чем признаки,

выделенные для событий мыши. Исключением явилась модель на основе алгоритма NB, результаты которой остались неизменны, что объясняется недостаточным объёмом признаков для успешного обучения классификатора с использованием данного алгоритма.

В качестве последнего шага, были построены и оценены модели классификаторов на основе предыдущего набора признаков и дополнительных признаков, выделенных для диграфов и триграфов. Результаты оценок представлены в таблице 3.

Таблица 3 — Оценка моделей классификаторов (события мыши в сочетании со специальными клавишами, диграфами и триграфами)

№	Алгоритм	Precision Non-Insider (train / test)	Precision Insider (train / test)	Recall Non-Insider (test / test)	Recall Insider (test / test)	Accuracy (train / test)
1	LR	1.00 / 0.73	1.00 / 1.00	1.00 / 1.00	1.00 / 0.62	1.00 / 0.81
2	k-NN	1.00 / 0.73	1.00 / 1.00	1.00 / 1.00	1.00 / 0.62	1.00 / 0.81
3	NB	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	1.00 / 0.75	1.00 / 0.88
4	SVM	1.00 / 0.67	1.00 / 1.00	1.00 / 1.00	1.00 / 0.50	1.00 / 0.75
5	DT	1.00 / 0.50	1.00 / 0.00	1.00 / 1.00	1.00 / 0.00	1.00 / 0.50
6	RF	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	1.00 / 0.75	1.00 / 0.88
7	MLP	1.00 / 0.67	1.00 / 1.00	1.00 / 1.00	1.00 / 0.50	1.00 / 0.75
8	AB	1.00 / 0.89	1.00 / 1.00	1.00 / 1.00	1.00 / 0.88	1.00 / 0.94
9	GB	1.00 / 0.80	1.00 / 1.00	1.00 / 1.00	0.82 / 0.75	1.00 / 0.88

Для большинства моделей результаты ухудшились, но ненамного. Для модели на основе DT существенное ухудшение связано со склонностью данного алгоритма к сильному переобучению. Однако для модели на основе RF, использующей в своей основе ансамбль деревьев принятия решений, результат ухудшился несильно, при этом меньшая точность по сравнению с соответствующей моделью из таблицы 2 связана с увеличением числа признаков

при том же объёме данных, на которых происходило обучение. Аналогичная причина небольшого уменьшения точностей и для остальных моделей. Однако стоит отметить, что переобучению подверглась только модель на основе DT, так как все остальные модели успешно прошли валидацию на 21% данных. Существенное улучшение точности было сделано моделью на основе NB, что объясняется достаточным количеством признаков, необходимых для успешной сходимости данного алгоритма.

Высокие показатели точностей моделей на основе двух последних групп признаков объясняются несколькими причинами:

- данные были собраны у людей из одной возрастной категории с приблизительно одинаковыми умением пользоваться компьютером и показателем скорости печати на клавиатуре. При этом в процессе сбора данных использовались одни и те же клавиатура и мышь. Поэтому в данных не наблюдается сильного разброса, что уменьшило количество потенциальных кластеров, а это во многом способствует облегчению процесса обучения, более точной генерализации, и следовательно улучшению классификации;
- в процессе накопления данных участники были предупреждены о том, чтобы чётко следовать инструкциям сценариев, что привело к отсутствию лишних данных и соответственно к отсутствию выбросов;
- ограничения по времени, которые были наложены на сценарии с участием внутреннего нарушителя явно повлияли на скорости нажатия клавиш. Это привело к тому, что большинство временных признаков, рассчитанных в сценариях с участием внутреннего нарушителя имеют значения медиан меньшие значений медиан соответствующих временных показателей, рассчитанных для сценариев с нормальным поведением. Данная закономерность отображена на рисунках 2,3.

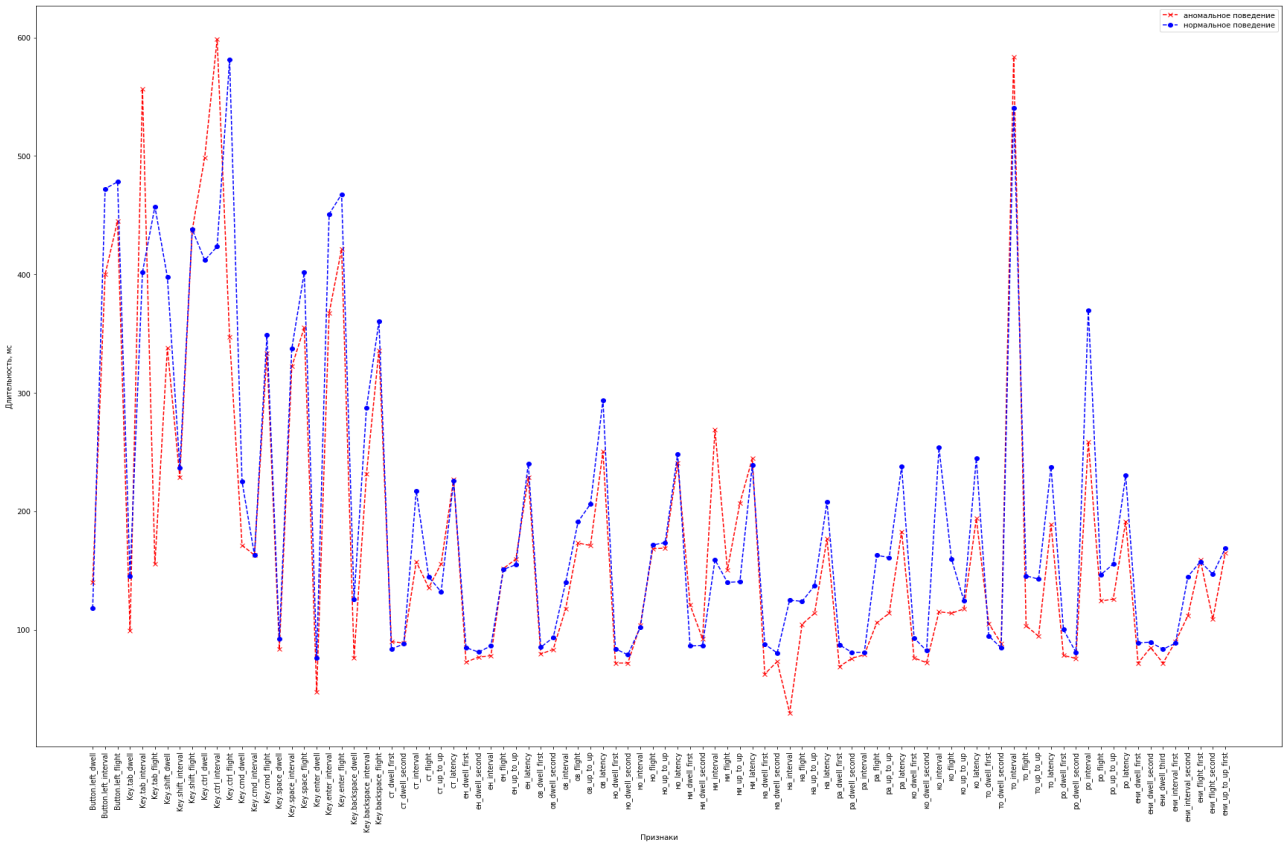

Рисунок 2 — Относительное расположение первое половины временных признаков аномальной и нормальной категорий

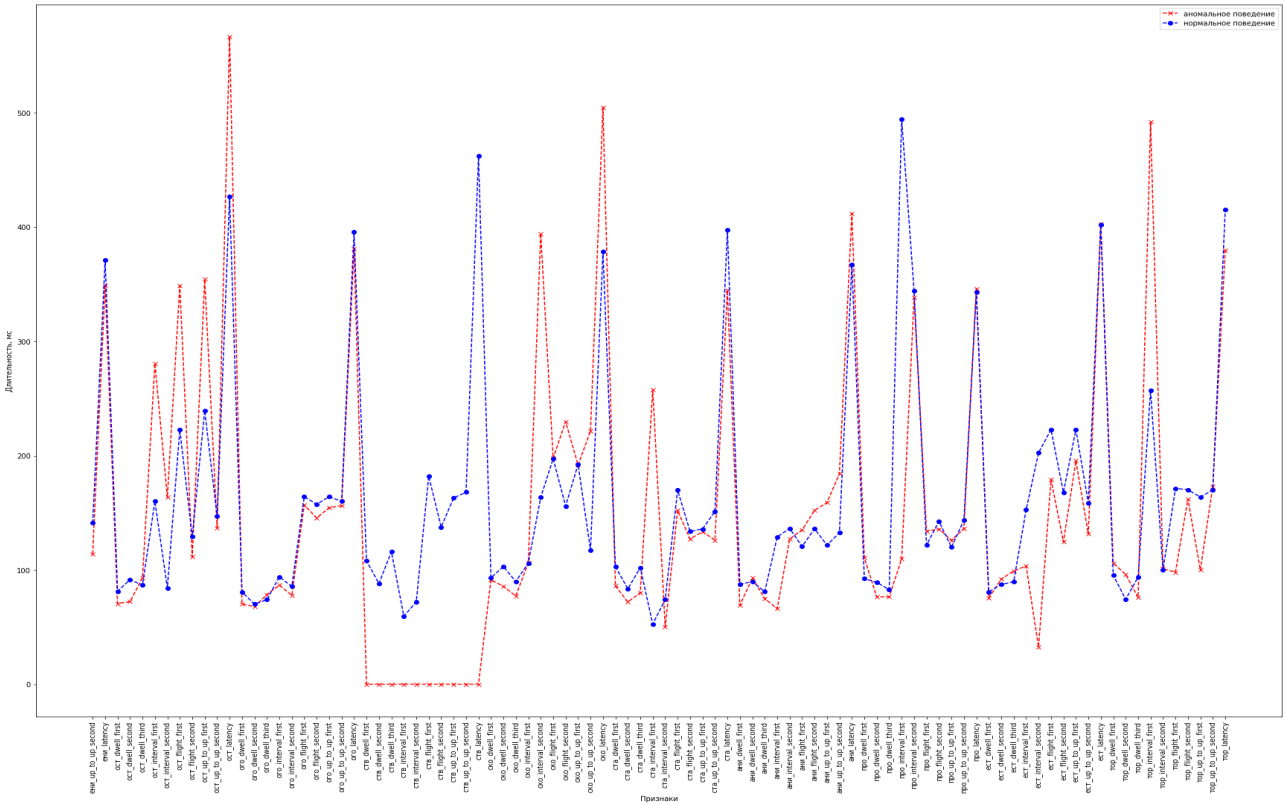

Рисунок 3 — Относительное расположение второй половины временных признаков аномальной и нормальной категорий

На графиках, представленных выше видно, что не для всех временных признаков из категории аномального поведения значения длительностей меньше значений соответствующих признаков из категории нормального поведения. Однако данные несоответствия не являются выбросами, так как доля таких несоответствий в среднем для групп признаков (Таблица В.1) составила 45%. Это показано в таблице 4.

Таблица 4 — Отношение соответствующих групп признаков аномальной и нормальной категорий

№	Группа признаков	Количество несоответствий, шт	Всего сравнений, шт	Доля несоответствий, %
1	dwell	6	8	75
2	interval	13	18	72
3	flight	16	18	89
4	up_to_up	7	10	30
5	dwell_first	15	20	25
6	dwell_second	15	20	25
7	dwell_third	7	10	30
8	interval_first	4	10	60
9	interval_second	6	10	40
10	flight_first	5	10	50
11	flight_second	7	10	30
12	up_to_up_first	6	10	40
13	up_to_up_second	6	10	40
14	latency	14	20	30

Достаточно сложно сравнить алгоритмы при полученных показателях точности, однако возможно объяснить почему тот или иной алгоритм сработал хуже или наоборот лучше другого.

Алгоритм DT выдал максимальную точность и не переобучился, хотя основным его недостатком является склонность к переобучению. Это возможно объяснить только тем, что признаки достаточно информативны, а данные категорий очень хорошо генерализируются, то есть их можно описать функцией низкого порядка, что позволяет просто разделить категории в пространстве и добиться таких показателей. Результат алгоритм RF в этом случае очевиден, так как его основная идея заключается в том, чтобы использовать ансамбль деревьев принятия решений и усреднить результаты по всем деревьям, а так как DT выдала максимально возможную точность, то и RF справился с классификацией без никаких проблем. Аналогичное объяснение у результатов алгоритмов на основе бустинга AB и GB, так как они также используют ансамбли деревьев принятия решений.

k-NN, SVM и MLP отработали немного хуже, так как для этих алгоритмов недостаточно того объёма данных, который использовался в исследовании. Однако переобучения не было ни в одном из трёх алгоритмов, что опять же объясняется закономерностью распределения данных.

Логистическая регрессия показала максимально возможную точность и это можно объяснить только тем, что распределение данных описывается очень простой функцией. При этом переобучения так же не произошло.

ЗАКЛЮЧЕНИЕ

В данной работе был проведён анализ исследований в области обнаружения угрозы внутреннего нарушителя с использованием различных биометрических показателей, а также продемонстрирована возможность обнаружения угрозы 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, выполняя 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 и построены модели классификаторов на основе 9 различных алгоритмов машинного обучения с учителем. Почти все модели, обученные на полном наборе признаков, показали максимально возможную точность за исключением некоторых, которые выдали точность в 88%.

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, подтвердили сделанное предположение о том, что выявление стресса может сыграть ключевую роль при обнаружении угрозы внутреннего нарушителя.

В дальнейшем исследовании планируется собрать намного больше данных из разных возрастных категорий пользователей с различными способностями использования клавиатуры, что позволит использовать все события на клавиатуре и мыши, при выделении признаков, которые были опущены в данном исследовании. При этом планируется доработка сценариев с целью большего задействования латинского алфавита, что в разы увеличит количество признаков, из которых посредством алгоритмов можно будет выделить наиболее

информативные признаки. Также будут исследованы модели на основе алгоритмов обучения без учителя и алгоритмов обнаружения аномалий.

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. Damaševičiusa, R. Maskeliūnasa, M. Woźniak // The Workshop on Computational Intelligence in Ambient Assisted Living Systems (CIAALS 2018) — 2018.

16 **Yassir H.** Insider Threat Detection Based on Users' Mouse Movements and Keystrokes Behavior [Текст] / H. Yassir, 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.

18 **Insider Threat** [Электронный ресурс] — Режим доступа: https://www.observeit.com/insider-threat/ — Свободный.

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.

20 **Oladimeji T.O.** Review on Insider Threat Detection Techniques [Текст] / 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 [Текст] / 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 [Текст] / 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 [Текст] / 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 [Текст] / 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 [Текст] / 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 [Текст] / J. Zhang, Y. Chen, J. Ankang // Turkish Journal of Electrical Engineering & Computer Sciences — 2017.

27 **Choras M.** Machine Learning Techniques for Threat Modeling and Detection [Текст] / M. Choras, R. Kozik // Security and Resilience in Intelligent Data-Centric Systems and Communication Networks — 2018 — С. 179-192.

28 **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.

29 **Russian Letter Frequencies** [Электронный ресурс] — Режим доступа: http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/russian-letter-frequencies/ — Свободный.

Приложение А

Сценарии, используемые в процессе сбора данных

Сценарии, вызывающие стресс и описывающие действия внутреннего нарушителя. На каждый из нижеперечисленных сценариев выделяется по 4 минуты:

- 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;
- 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;

- 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 Вконтакте, где ссылка к сохранённым на ресурсе характеристикам будет поделена на несколько частей.

Сценарии, не вызывающие стресс и описывающие действия, которые мог бы выполнять сотрудник в течение рабочего дня. Ограничений по времени нет:

- 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;
- 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;

- 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.

Приложение Б

Логирование событий клавиатуры и мыши

```
from pynput import keyboard
from pynput 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 you 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 В

Временные и частотны признаки, используемые для нажатий клавиш или кнопок

Таблицы В.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
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 В.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

Приложение Г

Выделение частотных и временных признаков и 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.esc",
    "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","ра","ко","то","ро"] #
,"ан","ос","по","го","ер","од", "ре"]
ru_tri = ["ени","ост","ого","ств","ско","ста","ани","про","ест","тор"] #
,"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:
                    # сохраняем 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 Д

Предобработка признаков и модели классификаторов

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

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

# разделение датасета на обучаемую и валидационную выборки
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

# 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)

# метрики
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 = 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_test, y_pred))
    print(classification_report(Y_test, Y_pred))

# модели классификаторов
models = {
    'logistic regression': LogisticRegression(solver='lbfgs', random_state=0),
    'k-nearest neighbors': KNeighborsClassifier(n_neighbors=3),
    'naive bayes': BernoulliNB(),
    'support vector machines': SVC(kernel='linear', gamma='auto'),
    'random forest': RandomForestClassifier(max_depth=2, n_estimators = 200,
random_state=0),
    'decision tree': DecisionTreeClassifier(),
    'adaptive boosting': AdaBoostClassifier(DecisionTreeClassifier(),
algorithm="SAMME", n_estimators=200),
    'gradient boosting': GradientBoostingClassifier(n_estimators=200,
learning_rate=1.0, max_depth=1, random_state=0),
```

```
    'multi-layer perceptron': MLPClassifier(alpha=1, max_iter=1000),
}

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