1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

**«Построении нейронной сети для определения наличия атак в компьютерной сети»**

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. к.т.н. Платонов В.В.
4. Санкт-Петербург
5. 2023
6. Содержание

[Содержание 2](#_Toc138245088)

[Введение 3](#_Toc138245089)

[1 Набор данных 4](#_Toc138245090)

[2 Описание модели 11](#_Toc138245091)

[2.1 Функция активации 11](#_Toc138245092)

[2.2 Оптимизатор 12](#_Toc138245093)

[3 Метрики качества модели 13](#_Toc138245094)

[3.1 Простые оценки 13](#_Toc138245095)

[3.2 F-мера 14](#_Toc138245096)

[3.3 ROC-AUC 14](#_Toc138245097)

[4 Полученные результаты 17](#_Toc138245098)

[4.1 Метод главных компонент 17](#_Toc138245099)

[4.2 Построение нейронной сети и подбор параметров 18](#_Toc138245100)

[4.3 Оценка разработанной модели 20](#_Toc138245101)

[Заключение 22](#_Toc138245102)

[Список использованных источников 23](#_Toc138245103)

[Приложение 25](#_Toc138245104)

# Введение

На сегодняшний день ни одно предприятие не обходится без компьютерных технологий для обслуживания своей инфраструктуры. Даже малый бизнес нуждается в кассовом аппарате, терминале оплаты, а это уже сеть, хоть и небольшая. Если крупные предприятия могут позволить себе полноценные и дорогостоящие средства защиты, наполненные всевозможными межсетевыми экранами, средствами обнаружения и предотвращения вторжений, снифферами и тому подобными, то более мелкие предприятия не обладают ресурсами для реализации данной модели защиты. Встает вопрос о создании такой системы обнаружения вторжений, которая могла бы обнаруживать наличие атаки, а также потенциально относить её к некоторому классу, обладая при этом минимальным оптимальным числом параметров. При обнаружении атак в сетевых инфраструктурах могут использоваться различные наборы параметров, коих на сегодняшний очень много.

Актуальность работы: обнаружение атак в компьютерной сети.

Цель — обнаружение сетевых атак классификатором на базе нейронных сетей.

Задачи, выделяемые в данной работе:

1. Проанализировать набор данных, выбрать главные компоненте для обучения НС.
2. Разработать программный прототип персептрона и подобрать для него параметры, дающие лучший результат при обучении.
3. Обучить НС и проанализировать полученные результаты.

# Набор данных

Был использован набор данных CSE-CIC-IDS2018 [11]. В нем используется понятие профилей для систематического создания наборов данных, которые будут содержать подробные описания вторжений и абстрактные модели распространения для приложений, протоколов или сетевых объектов более низкого уровня. Эти профили могут использоваться для генерации событий в сети. Благодаря абстрактному характеру сгенерированных профилей их можно применять к широкому спектру сетевых протоколов с различной топологией. Выделяется два основных класса профилей:

**B-профили**: инкапсулируют поведение объектов пользователей с использованием различных методов машинного обучения и статистического анализа.

**M-Профили**: попытка описать сценарий атаки недвусмысленным образом.  Для атак рассматривается шесть различных сценариев:

1. **Проникновение в сеть изнутри.** В этом сценарии жертве отправляется вредоносный файл по электронной почте и используется уязвимость приложения. После успешной эксплуатации на компьютере жертвы будет запущен бэкдор, а затем компьютер используется для сканирования внутренней сети на наличие других уязвимых блоков.
2. **Отказ в обслуживании по протоколу HTTP.** В этом сценарии используются средства Slowloris и LOIC в качестве основных инструментов, которые. Slowloris начинает с установления TCP-соединения с удаленным сервером. Инструмент поддерживает соединение открытым, регулярно отправляя на сервер действительные, неполные HTTP-запросы, чтобы предотвратить закрытие сокетов. Поскольку любой веб-сервер имеет ограниченную способность обслуживать соединения, это будет только вопросом времени, когда все сокеты будут израсходованы и никакое другое соединение не сможет быть установлено.
3. **Атаки на веб-приложения.** В этом сценарии используется уязвимое веб-приложение (DVWA), которое разработано, чтобы помочь специалистам по безопасности проверить свои навыки. На первом этапе проверяем веб-сайт с помощью сканера уязвимостей веб-приложения, а затем проводим различные типы веб-атак на уязвимый веб-сайт, включая внедрение SQL, командное внедрение и неограниченную загрузку файлов.
4. **Атаки методом полного перебора.** Такие атаки очень распространены в сетях, поскольку они, как правило, взламывают учетные записи со слабыми комбинациями имени пользователя и пароля. Окончательный сценарий был разработан с целью получения учетной записи SSH и MySQL путем проведения атаки методом перебора по словарю против главного сервера.
5. Есть некоторые атаки, основанные на некоторых известных уязвимостях, которые могут проводиться в течение определенного периода времени (это чрезвычайные уязвимости, которые иногда затрагивают миллионы серверов или жертв, и обычно требуются месяцы, чтобы исправить все уязвимые компьютеры по всему миру), одна из самых известных за последние годы — **Heartbleed**. Одним из самых известных инструментов для использования Heartbleed является Heartleech. Он может сканировать системы, уязвимые для ошибки, и затем может быть использован для их эксплуатации и извлечения данных. Для использования уязвимости был скомпилирован OpenSSL версии 1.0.1f, которая является уязвимой. Затем мы используется Heartleech для извлечения памяти сервера.
6. **Ботнет.** В этом наборе данных используется Zeus, который представляет собой вредоносный пакет "Троянский конь", работающий в версиях Microsoft Windows. Хотя он может использоваться для выполнения многих задач, его часто используют для кражи банковской информации путем регистрации нажатий клавиш в браузере пользователя и захвата формы. Он также используется для установки программы-вымогателя Crypto-Locker. Zeus распространяется в основном посредством бесплатных загрузок и фишинговых схем.

В таблице представлены все параметры, описанные в датасете.

Таблица — Описание всех параметров

| № | Имя | Описание |
| --- | --- | --- |
| 1 | Dest\_port | Идентификатор пользователя |
| 2 | protocol | Хост отправителя |
| 3 | fl\_dur | Продолжительность потока |
| 4 | tot\_fw\_pk | Общее количество пакетов в прямом направлении |
| 5 | tot\_bw\_pk | Общее количество пакетов в обратном направлении |
| 6 | tot\_l\_fw\_pkt | Общий размер пакета в прямом направлении |
| 7 | fw\_pkt\_l\_max | Максимальный размер пакета в прямом направлении |
| 8 | fw\_pkt\_l\_min | Минимальный размер пакета в прямом направлении |
| 9 | fw\_pkt\_l\_avg | Средний размер пакета в прямом направлении |
| 10 | fw\_pkt\_l\_std | Размер стандартного отклонения пакета в прямом направлении |
| 11 | Bw\_pkt\_l\_max | Максимальный размер пакета в обратном направлении |
| 12 | Bw\_pkt\_l\_min | Минимальный размер пакета в обратном направлении |
| 13 | Bw\_pkt\_l\_avg | Средний размер пакета в обратном направлении |
| 14 | Bw\_pkt\_l\_std | Размер стандартного отклонения пакета в обратном направлении |
| 15 | fl\_byt\_s | скорость потока в байтах, то есть количество пакетов, передаваемых в секунду |
| 16 | fl\_pkt\_s | скорость потока пакетов, то есть количество пакетов, передаваемых в секунду |
| 17 | fl\_iat\_avg | Среднее время между двумя потоками |
| 18 | fl\_iat\_std | Стандартное отклонение времени двух потоков |
| 19 | fl\_iat\_max | Максимальное время между двумя потоками |
| 20 | fl\_iat\_min | Минимальное время между двумя потоками |
| 21 | fw\_iat\_tot | Общее время между двумя пакетами, отправленными в прямом направлении |
| 22 | fw\_iat\_avg | Среднее время между двумя пакетами, отправленными в прямом направлении |
| 23 | fw\_iat\_std | Время стандартного отклонения между двумя пакетами, отправленными в прямом направлении |
| 24 | fw\_iat\_max | Максимальное время между двумя пакетами, отправленными в прямом направлении |
| 25 | fw\_iat\_min | Минимальное время между двумя пакетами, отправленными в прямом направлении |
| 26 | bw\_iat\_tot | Общее время между двумя пакетами, отправленными в обратном направлении |
| 27 | bw\_iat\_avg | Среднее время между двумя пакетами, отправленными в обратном направлении |
| 28 | bw\_iat\_std | Стандартное отклонение времени между двумя пакетами, отправленными в обратном направлении |
| 29 | bw\_iat\_max | Максимальное время между двумя пакетами, отправленными в обратном направлении |
| 30 | bw\_iat\_min | Минимальное время между двумя пакетами, отправленными в обратном направлении |
| 31 | fw\_psh\_flag | Количество раз, когда флаг PSH был установлен в пакетах, перемещающихся в прямом направлении (0 для UDP) |
| 32 | bw\_psh\_flag | Количество раз, когда флаг PSH был установлен в пакетах, перемещающихся в обратном направлении (0 для UDP) |
| 33 | fw\_urg\_flag | Количество раз, когда флаг URG был установлен в пакетах, перемещающихся в прямом направлении (0 для UDP) |
| 34 | bw\_urg\_flag | Количество раз, когда флаг URG был установлен в пакетах, перемещающихся в обратном направлении (0 для UDP) |
| 35 | fw\_hdr\_len | Общее количество байт, используемых для заголовков в прямом направлении |
| 36 | bw\_hdr\_len | Общее количество байт, используемых для заголовков в прямом направлении |
| 37 | fw\_pkt\_s | Количество пересылаемых пакетов в секунду |
| 38 | bw\_pkt\_s | Количество обратных пакетов в секунду |
| 39 | pkt\_len\_min | Минимальная длина потока |
| 40 | pkt\_len\_max | Максимальная длина потока |
| 41 | pkt\_len\_avg | Средняя длина потока |
| 42 | pkt\_len\_std | Длина стандартного отклонения потока |
| 43 | pkt\_len\_va | Минимальное время между прибытием пакета |
| 44 | fin\_cnt | Количество пакетов с FIN |
| 45 | syn\_cnt | Количество пакетов с SYN |
| 46 | rst\_cnt | Количество пакетов с RST |
| 47 | pst\_cnt | Количество пакетов с PUSH |
| 48 | ack\_cnt | Количество пакетов с подтверждением |
| 49 | urg\_cnt | Количество пакетов с URG |
| 50 | cwe\_cnt | Количество пакетов с CWE |
| 51 | ece\_cnt | Количество пакетов с ECE |
| 52 | down\_up\_ratio | Соотношение загрузки и выгрузки |
| 53 | pkt\_size\_avg | Средний размер пакета |
| 54 | fw\_seg\_avg | Средний размер, наблюдаемый в прямом направлении |
| 55 | bw\_seg\_avg | Средний размер, наблюдаемый в обратном направлении |
| 56 | fw\_byt\_blk\_avg | Среднее количество байтов с массовой скоростью в прямом направлении |
| 57 | fw\_pkt\_blk\_avg | Среднее количество пакетов с массовой скоростью в прямом направлении |
| 58 | fw\_blk\_rate\_avg | Среднее количество массовых ставок в прямом направлении |
| 59 | bw\_byt\_blk\_avg | Среднее количество байтов с массовой скоростью в обратном направлении |
| 60 | bw\_pkt\_blk\_avg | Среднее количество пакетов с массовой скоростью в обратном направлении |
| 61 | bw\_blk\_rate\_avg | Среднее количество массовых ставок в обратном направлении |
| 62 | subfl\_fw\_pk | Среднее количество пакетов в подпотоке в прямом направлении |
| 63 | subfl\_fw\_byt | Среднее количество байтов в подпотоке в прямом направлении |
| 64 | subfl\_bw\_pkt | Среднее количество пакетов в подпотоке в обратном направлении |
| 65 | subfl\_bw\_byt | Среднее количество байтов в подпотоке в обратном направлении |
| 66 | fw\_win\_byt | Количество байтов, отправленных в начальном окне в прямом направлении |
| 67 | bw\_win\_byt | количество байтов, отправленных в начальном окне в обратном направлении |
| 68 | Fw\_act\_pkt | Количество пакетов, содержащих не менее 1 байта полезной нагрузки TCP-данных в прямом направлении |
| 69 | fw\_seg\_min | Минимальный размер сегмента, наблюдаемый в прямом направлении |
| 70 | atv\_avg | Среднее время, в течение которого поток был активен до перехода в режим ожидания |
| 71 | atv\_std | Стандартное отклонение времени, когда поток был активен до перехода в режим ожидания |
| 72 | atv\_max | Максимальное время, в течение которого поток был активен до перехода в режим ожидания |
| 73 | atv\_min | Минимальное время, в течение которого поток был активен до перехода в режим ожидания |
| 74 | idl\_avg | Среднее время простоя потока до того, как он стал активным |
| 75 | idl\_std | Стандартное отклонение времени простоя потока до того, как он стал активным |
| 76 | idl\_max | Максимальное время простоя потока до того, как он стал активным |
| 77 | idl\_min | Минимальное время, в течение которого поток простаивал, прежде чем стать активным |
| 78 | Label | Значение 1 или 0 |

# Описание модели

В качестве модели машинного обучения был выбран многослойный персептрон [12, 13, 14].

## Функция активации

В качестве функции активации была выбрана ReLu (rectified linear unit)

ReLu возвращает значение *х*, если *х* положительно, и 0 в противном случае. Её график приведен ниже на рисунке.

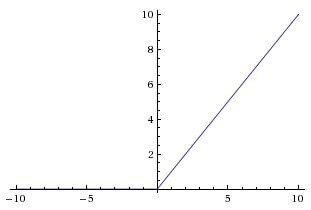


Рисунок — График функции активации

ReLu нелинейна по своей природе, её комбинация также нелинейна. Такая функция является хорошим аппроксиматором, так как любая функция может быть аппроксимирована комбинацией ReLu.

Одним из её достоинств является разреженность активации. Представим большую нейронную сеть с множеством нейронов. Использование сигмоиды или гиперболического тангенса будет влечь за собой активацию всех нейронов. Это означает, что почти все активации должны быть обработаны для описания выхода сети. Другими словами, это слишком затратно по времени вычислений.

Представим сеть со случайно инициализированными весами, в которой примерно 50% активаций равны 0 из-за характеристик ReLu. В такой сети включается меньшее количество нейронов (разреженная активация), а сама сеть становится легче.

Из-за того, что часть ReLu представляет из себя горизонтальную линию (для отрицательных значений *х*), градиент на этой части равен 0. Из-за равенства нулю градиента, веса не будут корректироваться во время спуска. Это означает, что пребывающие в таком состоянии нейроны не будут реагировать на изменения в ошибке/входных данных (просто потому, что градиент равен нулю, ничего не будет меняться). Такое явление называется проблемой умирающего ReLu (Dying ReLu problem). Из-за этой проблемы некоторые нейроны просто выключатся и не будут отвечать, делая значительную часть нейросети пассивной. Существуют вариации ReLu, которые помогают эту проблему избежать. Однако в данном примере все входные данные больше или равны 0, поэтому такой проблемы не возникнет.

ReLu менее требовательна к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции. Поэтому имеет смысл использовать ReLu при создании глубоких нейронных сетей [5].

## Оптимизатор

Оптимизаторы — важный компонент архитектуры нейронных сетей. Они играют важную роль в процессе тренировки нейронных сетей, помогая им делать всё более точные прогнозы. Оптимизаторы определяют оптимальный набор параметров модели, таких как вес и смещение, чтобы при решении конкретной задачи модель выдавала наилучшие результаты. Самой распространённой техникой оптимизации, используемой большинством нейронных сетей, является алгоритм градиентного спуска.

Adam — adaptive moment estimation сочетает в себе и идею накопления движения и идею более слабого обновления весов для типичных признаков. Adam отличается тем, что мы накапливаем не Δθ, а значения градиента. Авторы алгоритма предложили оценивать среднюю нецентрированную дисперсию для того, чтобы знать, как часто градиент изменяется.

Важное отличие состоит в начальной калибровке mt и vt: если задать нулевое начальное значение, то они будут долго накапливаться, особенно при большом окне накопления, а какие-то изначальные значения — это ещё два гиперпараметра, что усложняет дальнейшие вычисления. Поэтому mt и vt искусственно увеличиваются на первых шагах.

Здесь следует внимательно посмотреть на то, как быстро синхронизировались значения обновлений на первых зубцах графиков с прямоугольниками и на гладкость кривой обновлений на графике с синусом — её мы получили «бесплатно». При рекомендуемом параметре β1 резкие всплески градиента не вызывают мгновенного отклика в накопленном значении, поэтому хорошо настроенному Adam не нужен gradient clipping.

Авторы Adam предлагают в качестве значений по умолчанию *β1 = 0,9; β2 = 0,999; ϵ = 10-8* и утверждают, что алгоритм выступает лучше или примерно так же, как и все предыдущие алгоритмы на широком наборе датасетов за счёт начальной калибровки [6].

# Метрики качества модели

## Простые оценки

Accuracy — показывает долю правильных классификаций. Несмотря на очевидность и простоту, является одной из самых малоинформативных оценок классификаторов.

У этой метрики есть одна особенность, которую необходимо учитывать. Она присваивает всем классам одинаковый вес, что может быть не корректно в случае, если их распределение в обучающей выборке сильно смещено в сторону какого-то одного или нескольких классов. В этом случае у классификатора есть больше информации по этим классам и соответственно в рамках этих классов он будет принимать более точные решения.

Один выход из этой ситуации заключается в том, чтобы обучать классификатор на специально подготовленном, сбалансированном датасете. Минус этого решения в том, что тогда у классификатора нет информации об относительной частоте представителей того или иного класса.

Recall — (полнота, sensitivity, TPR (true positive rate)) показывает отношение верно классифицированных объектов класса к общему числу элементов этого класса.

Precision — (точность) показывает долю верно классифицированных объектов среди всех объектов, которые к этому классу отнес классификатор.

## F-мера

Чем выше точность и полнота, тем лучше. Но в реальной жизни максимальная точность и полнота не достижимы одновременно и приходится искать некий баланс. Поэтому, хотелось бы иметь некую метрику, которая объединяла бы в себе информацию о точности и полноте алгоритма. Именно такой метрикой является F-мера.

F-мера представляет собой гармоническое среднее между точностью и полнотой. Гармоническое среднее, в отличии арифметического стремится к нулю, когда хотя бы одно из значений стремится к нулю.

Данная формула придает одинаковый вес точности и полноте, поэтому F-мера будет падать одинаково при уменьшении и точности, и полноты [2].

## ROC-AUC

Кривая ROC получается путем вычисления и построения графика соотношения истинного положительного результата и ложноположительного результата для одного классификатора при различных пороговых значениях. Кривые ROC помогают визуализировать, как эти варианты влияют на производительность классификатора.

Истинный положительный показатель (он же Recall) может быть представлен как:

где TP — количество истинно положительных результатов, а FN — количество ложноотрицательных результатов. Истинный положительный показатель является мерой вероятности того, что фактический положительный экземпляр будет классифицирован как положительный.

Частота ложноположительных результатов может быть записана так:

где FP — количество ложноположительных результатов, а TN — количество истинных отрицательных результатов. Частота ложноположительных результатов, по сути, является показателем того, как часто будет возникать “ложная тревога” — или как часто фактический негативный случай будет классифицироваться как положительный.

На рисунке показано, как некоторые теоретические классификаторы будут отображаться на кривой ROC. Серая пунктирная линия представляет классификатор, который ничем не лучше случайного угадывания — он нанесен в виде диагональной линии. Фиолетовая линия представляет идеальный классификатор — с истинным положительным показателем 100% и ложноположительным показателем 0%. Почти все примеры из реального мира будут находиться где-то между этими двумя линиями — не идеальны, но обеспечивают большую прогностическую силу, чем случайное угадывание.

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 2 — Пример ROC кривой

Хотя полезно визуализировать кривую ROC классификатора, во многих случаях можно свести эту информацию к одному показателю — AUC, который обозначает площадь под кривой (ROC). Как правило, чем выше оценка AUC, тем лучше классификатор справляется с поставленной задачей.

Для классификатора, не обладающего способностью к прогнозированию (т.е. случайному угадыванию), соответствует AUC = 0,5, а для идеального классификатора AUC = 1,0. Большинство классификаторов будут находиться в диапазоне от 0,5 до 1,0.

Одним из преимуществ ROC-кривых является то, что они помогают нам найти порог классификации, соответствующий конкретной задаче.

Например, если бы требовалось оценить классификатор спама по электронной почте, лучше, чтобы частота ложноположительных результатов была очень, очень низкой. Нежелательно, чтобы кто-то потерял важное электронное письмо из-за спам-фильтра только потому, что алгоритм был слишком агрессивным. Вероятно, лучше даже пропустить через фильтр достаточное количество реальных спам-писем (истинно положительных), просто чтобы убедиться, что ни одно важное электронное письмо не было потеряно.

С другой стороны, если классификатор предсказывает, есть ли у кого-то неизлечимая болезнь, можно допустить большее количество ложноположительных результатов (неправильная диагностика болезни), просто чтобы убедиться, что не было пропущено ни одного истинно положительного результата (люди, у которых действительно есть болезнь).

Кроме того, кривые ROC и оценки AUC также позволяют сравнивать производительность разных классификаторов для одной и той же задачи [3].

# Полученные результаты

## Метод главных компонент

Метод главных компонент является одним из алгоритмов сокращения размерности обрабатываемых данных с наименьшей потерей информации в процессе машинного обучения. Суть метода заключается в следующем. Строится матрица, где каждая строка ― выборка, то есть числовая характеристика некоторой компоненты. Далее строится ковариационная матрица. Она является обобщением дисперсии на случай многомерных случайных величин и описывает разброс случайной величины так же, как и дисперсия. Диагональные элементы ковариационной матрицы показывают дисперсии по изначальному базису, а ее собственные значения ― по новому (по главным компонентам). Далее необходимо найти собственный вектор матрицы. При перемножении собственного вектора на матрицу ковариации будет получен новый вектор ― проекция, содержащий в себе значения главной компоненты

Из датасета были взяты записи и на основе них построены выборки размера 10000 для каждой компоненты. В языке программирования Python метод главных компонент уже реализован в библиотеке sklearn [8]. Необходимо задать, до какого числа компонент будет снижена размерность, и передать методу матрицу значений. В качестве возвращаемого значения будут получены собственные вектора с максимальными собственными числами.

Необходимо выделить столько главных компонент, чтобы они охватывали 99% информации. Формула для вычисления представлена ниже:

Для получения этих данных воспользуемся методом explained\_variance\_ratio\_ того же класса PCA библиотеки sklearn [9].

При помощи метода главных компонент были выбраны 5 основных параметров, которые содержат в себе 98% информации. Были выбраны следующие параметры (рисунок): продолжительность потока, среднее время между двумя пакетами, отправленными в обратном направлении, стандартное отклонение времени между двумя пакетами, отправленными в обратном направлении, минимальное время между двумя потоками, максимальное время простоя потока до того, как он стал активным. При чем параметр bw\_iat\_avg повторяется два раза, но имеет разные значения. В ходе поиска данной ошибки в сети Интернет было выяснено, что преобразование PCA не дает гарантии, что входные данные будут разными. Такая функция вносит наибольший вклад, поэтому получаются дублирующиеся компоненты [7]. Таким образом, данный дубликат игнорируется.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 — Главные компоненты

## Построение нейронной сети и подбор параметров

Перед построением модели машинного обучения необходимо подготовить входные данные. Была создана выборка размером 20000. Проверим, сколько лейблов содержат 0, а сколько 1: выборка оказалась несбалансированной. Исправим это методом upsampling, то есть, методом дублирования тех компонент, которых меньше, пока отношение не будет примерно одинаковым [10]. На рисунке видно отношение классов до и после преобразования.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рисунок 4 — Балансировка выборки данных

С помощью фреймворка scikit-learn 1.2.2 был реализован персептрон [4], а именно модель MLP. Реализованный персептрон имеет следующую структуру: n=5 входных нейронов, 2 скрытых слоя, 1 выходной нейрон, который выдает либо 0 (атаки нет), либо 1 (атака есть). Рассматриваются различные комбинации нейронов скрытого слоя от 0 до 2n+1 на каждом слое. На каждом шаге вычисляются ошибка, точность и AUC. Веса инициализируются библиотекой самостоятельно исходя из количества нейронов, либо могут быть указаны вручную при необходимости. На рисунке представлен график изменения значения ошибки при различном количестве нейронов в сети.

Оптимальный результат был получен при 11 нейронах на каждом слое. Однако стоит заметить, что значения могут меняться при каждом запуске алгоритма, поскольку в нем присутствует некоторый элемент случайности, а именно случайное разбиение всей выборки на обучающую и валидационную, а также случайное распределение весов.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 5 — Изменение значения ошибки при различном числе нейронов скрытого слоя

## Оценка разработанной модели

Рассмотрим более подробно метрики для каждого класса при 11 нейронах в каждом скрытом слое. На рисунке 10 представлена матрица несоответствия, precision, recall, F-мера для каждого класса в отдельности и в обоих в совокупности, а также посчитан параметр AUC. На рисунок 11 представлен график ROC-кривой.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 6 — Метрики качества разработанной модели

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Рисунок 7 — ROC

# Заключение

В ходе выполнения данной работы был реализован однослойный персептрон с 5 нейронами входного слоя, 11 нейронами на каждом из 2 скрытых слоев и 1 выходным нейроном. В качестве функции активации использовалась ReLU, в качестве функции ошибки — бинарная кросс-энтропия, в качестве оптимизатора — Адам.

При помощи метода главных компонент были выбраны 5 основных параметров, которые содержат в себе 98% информации. Были выбраны следующие параметры: продолжительность потока, среднее время между двумя пакетами, отправленными в обратном направлении, стандартное отклонение времени между двумя пакетами, отправленными в обратном направлении, минимальное время между двумя потоками, максимальное время простоя потока до того, как он стал активным.

Для улучшения качества модели она была обучена при различных комбинациях параметров. Оптимизатор Адам показал лучший результат по сравнению с алгоритмом SGD. Однако разница была не сильно значительной.

В итоге показатель AUC составил лишь 0,77. При проверке разработанной модели на другом наборе данных она показала результат свыше 0,95. Из чего можно сделать вид, что конкретно модель MLP может быть не такой эффективной в применении к данному набору данных.

# Список использованных источников

1. Оценка качества в задачах классификации [Электронный ресурс]. URL: <https://neerc.ifmo.ru/wiki/index.php?title=Оценка_качества_в_задачах_классификации>. – (дата обращения: 18.03.2023).
2. Оценка классификатора (точность, полнота, F-мера) [Электронный ресурс]. URL: <https://www.bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html>. – (дата обращения: 18.03.2023).
3. Understanding the ROC Curve and AUC [Электронный ресурс]. URL: <https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb> . – (дата обращения: 07.05.2023).
4. sklearn.neural\_network.MLPClassifier [Электронный ресурс]. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html> . – (дата обращения: 06.05.2023).
5. Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, tahn [Электронный ресурс]. URL: <https://neurohive.io/ru/osnovy-data-science/activation-functions/>. – (дата обращения: 17.01.2023).
6. Методы оптимизации нейронных сетей [Электронный ресурс]. URL: <https://habr.com/ru/articles/318970/>. – (дата обращения: 21.05.2023).
7. [Why PCA output some components duplicately?](https://stackoverflow.com/questions/67769996/why-pca-output-some-components-duplicately) [Электронный ресурс]. URL: <https://stackoverflow.com/questions/67769996/why-pca-output-some-components-duplicately>. – (дата обращения: 26.06.2021).
8. Как работает метод главных компонент (PCA) на простом примере [Электронный ресурс]. URL: <https://habr.com/ru/post/304214/> . – (дата обращения: 12.06.2022).
9. [sklearn.decomposition](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition).PCA [Электронный ресурс]. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA.get_feature_names_out>. – (дата обращения: 26.06.2022).
10. Как победить несбалансированность датасета: метод upsampling data [Электронный ресурс]. URL: <https://habr.com/ru/articles/568266/>. – (дата обращения: 21.05.2023).
11. Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, «[Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization](https://www.semanticscholar.org/paper/Toward-Generating-a-New-Intrusion-Detection-Dataset-Sharafaldin-Lashkari/a27089efabc5f4abd5ddf2be2a409bff41f31199)», 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018
12. Д. Рутковская, М. Пилиньский, Л. Рутковский Нейронные сети, генетические алгоритмы и нечеткие системы //Горячая линия – Телеком. – 2006. – №. 103. – С. 18-25.
13. Фрэнк Розенблатт [Принципы нейродинамики: перцептроны и теория механизмов мозга](http://www.raai.org/library/books/other/ros_principles2.djvu) //М.: «Мир», 1965. – С. 43.
14. Ф. Вассерман Нейрокомпьютерная техника: Теория и практика. // М.: «Мир», 1992.

# Приложение

import pandas as pd  
from math import sqrt  
from sklearn.neural\_network import MLPClassifier  
from sklearn.preprocessing import Normalizer  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV  
from sklearn.metrics import \*  
import matplotlib.pyplot as plt  
import warnings  
from sklearn.exceptions import DataConversionWarning  
warnings.filterwarnings(action='ignore', category=DataConversionWarning)  
  
def main():  
 # how much neurons we need?  
 Q1 = Q2 = 12  
  
 hidden = [11, 11]  
 loss = []  
 my\_dict = {'loss': 1, 'layer1': 0, 'layer2': 0}  
 X = pd.read\_csv('nir.csv', sep=',', usecols=[1, 2, 3, 4, 5])  
 Y = pd.read\_csv('nir.csv', sep=',', usecols=[6])  
 X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, train\_size=0.5, random\_state=42)  
 # test\_data = pd.read\_csv('validate.csv', sep=',')  
 X\_train = Normalizer().fit\_transform(X\_train)  
 X\_test = Normalizer().fit\_transform(X\_test)  
 # create the model  
  
 print("start training...")  
 clf = MLPClassifier(hidden\_layer\_sizes=hidden, activation="relu", solver='adam',  
 max\_iter=1000, shuffle=True)  
 # print("neurons in hidden layer ", i, " ", j)  
 clf.fit(X\_train, Y\_train)  
 print("start validating...")  
 y\_pred = clf.predict(X\_test)  
 loss\_i = zero\_one\_loss(Y\_test, y\_pred)  
  
 print("loss = ", loss\_i)  
 print("mean test accuracy = ", clf.score(X\_test, Y\_test)) # Return the mean accuracy on the given test data and labels.  
 print("confusion matrix:\n", confusion\_matrix(Y\_test, y\_pred))  
 print(classification\_report(Y\_test, y\_pred))  
  
 #roc-auc  
 print("AUC = ", roc\_auc\_score(Y\_test, y\_pred))  
 RocCurveDisplay.from\_predictions(Y\_test, y\_pred)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()