МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Факультет Информационных систем и технологий

Кафедра Безопасности информационных систем

Допустить к защите

Заведующий кафедрой
_____ Бородянский Ю.М.
(подпись) *(Ф.И.О.)*
«____» _____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка нейросети для прогнозирования угроз телекоммуникационных систем
(тема ВКР)

Вид выпускной квалификационной работы
бакалаврская работа
(бакалаврская работа, дипломная работа, дипломный проект, магистерская диссертация)

Направление/специальность подготовки
09.03.02 Информационные системы и технологии
(код и наименование направления/специальности)

Направленность (профиль)
Информационные системы и технологии
(наименование)

Квалификация Бакалавр
(наименование квалификации в соответствии с ФГОС ВО / ГОС ВПО)

Студент:
Гаипов Н.В., ИСТ-722 _____
(Ф.И.О., № группы) *(подпись)*

Научный руководитель:
д.т.н., профессор Буйневич М.В. _____
(учёная степень, учёное звание, Ф.И.О.) *(подпись)*

Санкт-Петербург
2021

Оборотная сторона титульного листа

_____

работа написана мною самостоятельно

_____

работа не содержит неправомерных заимствований

_____

работа может быть размещена в электронно-библиотечной системе университета

_____

_____ _____ _____

(дата) *(подпись)* *(ФИО студента)*

Текст ВКР размещен в электронно-библиотечной системе университета

Руководитель отдела комплектования библиотеки _____

(Ф.И.О.)

_____ _____

(дата) *(подпись)*

Коэффициент оригинальности ВКР _____ %.

Проверил: ___д.т.н., профессор Буйневич Михаил Викторович_____

(Должность, Ф.И.О.)

_____ _____

(дата) *(подпись)*

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,

СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Факультет ____ИСиТ_____ Кафедра_____БИС_____
Направление (специальность) 09.03.02 Информационные системы и технологии
_____
(код и наименование)

Утверждаю:
и.о. Зав. кафедрой БИС к.т.н., доцент
_____Бородянский Ю.М._____
(Ф.И.О., подпись)
«_____»_____20 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы (ВКР)

1. Студент____Гаипов Никита Вячеславович_ № группы _ИСТ-722_____
(фамилия, имя, отчество)

2. Руководитель Буйневич Михаил Викторович, д.т.н. профессор кафедры БИС
(фамилия, имя, отчество, должность, уч. степень и звание)
_____

3. Квалификация_____бакалавр_____
(наименование в соответствии с ФГОС ВО/ ГОС ВПО)

4. Вид работы _____бакалаврская работа_____
(бакалаврская работа, дипломный проект, дипломная работа, магистерская диссертация)

5. Тема ВКР Разработка нейросети для прогнозирования угроз телекоммуникационных систем_____
_____
_____
_____
утверждена приказом ректора университета от «___»_____ 20__ г. № ___

6. Исходные данные (технические требования): набор данных трафика сети предприятия; библиотеки для машинного обучения и построения нейросетей, программное обеспечение для написания программ на языке Python, методы прогнозирования угроз в информационных системах, методы и алгоритмы машинного обучения._____
_____

7. Содержание работы (анализ состояния проблемы, проведение исследований, разработка, расчеты параметров, экономическое обоснование и др.)

Введение

1. Исследование существующих методов прогнозирования угроз

2. Концепция метода решения задачи

3. Разработка архитектуры системы прогнозирования

4. Моделирование процесса прогнозирования угроз на основе сетевого трафика информационной системы с помощью системы прогнозирования

Заключение

8. Вид отчетных материалов, представляемых в ГЭК (пояснительная записка, перечень, графического материала, отчет о НИР, технический проект, образцы и
др.):

1. Пояснительная записка

2. Презентация

3. Два CD-диска с электронными материалами

_____
_____
_____
_____
_____
_____

9. Консультанты по ВКР с указанием относящихся к ним разделов

Раздел	Консультант	Подпись дата	
		Задание выдал	Задание принял
1.			
2.			
3.			
4.			

Дата выдачи задания «_____» _____20 г.

Дата представления ВКР к защите «_____» _____20 г.

*Руководитель ВКР*_____Буйневич М.В._____
(подпись)

Студент _____Гаипов Н.В._____
(подпись)

КАЛЕНДАРНЫЙ ПЛАН

№ п/п	Наименование этапов выпускной квалификационной работы (ВКР)	Срок выполнения этапов ВКР	Примечание
1.	Постановка цели выполнения ВКР и задач.	21.04.2021	
2.	Работа с теоретическим материалом.	21.04.2021-04.06.2021	
3.	Сбор информации, необходимой для написания работы.		
4.	Систематизация и обработка материалов ВКР.		
5.	Анализ полученных в работе результатов, обобщение.		
6.	Подготовка отчетных материалов, представляемых в государственную экзаменационную комиссию, доклада к защите и презентации.	05.06.2021-04.07.2021	
7.	Подготовка к защите ВКР, включая подготовку к процедуре защиты и процедуру защиты.		

Студент _____Гаипов Н.В._____
(подпись)

Руководитель ВКР _____БуйневичМ.В._____
(подпись)

РЕФЕРАТ

Тема выпускной квалификационной работы: «Разработка нейросети для прогнозирования угроз телекоммуникационных систем».

Работа содержит:

96 страниц, 35 рисунков, 4 таблицы и список литературы из 20 наименований.

Перечень ключевых слов: ПРОГНОЗИРОВАНИЕ УГРОЗ, НЕЙРОСЕТИ, РЕКУРРЕНТНЫЕ НЕЙРОСЕТИ, МОДЕЛЬ ОЦЕНКИ ВОЗНИКНОВЕНИЯ УГРОЗ, ПРИЧИННО-СЛЕДСТВЕННАЯ СВЯЗЬ, МАШИННОЕ ОБУЧЕНИЕ.

Цель дипломной работы – разработка модели прогнозирования угроз в телекоммуникационных системах на основе алгоритмов машинного обучения, которая позволяет автоматизировать и повысить точность процесса оценивания угроз за счет анализа временных зависимостей между данными.

В результате дипломной работы были рассмотрены теоретические основы реализации угроз в телекоммуникационных системах, составлена и предложена модель для их прогнозирования, учитывающая возникновение новых источников угроз на основе предыдущих, также предложена реализация модели прогнозирования угроз на базе нейронной сети и алгоритмов машинного обучения для систем обнаружения вторжений IPS/IDS, произведена оценка работы системы на наборе данных KDD-99, включающего в себя данные сетевого трафика, содержащего различные виды угроз, по итогу разработанная система отвечает всем предъявленным в работе требованиям по оценке качества прогнозирования.

ABSTRACT

Theme of final qualifying work: «Development of the neural network for predicting threats in telecommunication systems».

The work contains:

96 pages, 35 illustrations, 4 tables and bibliography of 20 titles.

List of keywords: THREAT PREDICTION, NEURAL NETWORKS, RECURRENT NEURAL NETWORKS, THREAT ASSESSMENT MODEL, CAUSATION, MACHINE LEARNING.

The purpose of the work is to develop a model for predicting threats in telecommunication systems based on machine learning algorithms, which allows automating and improving the accuracy of the threat assessment process by analyzing time dependencies between data.

As a result of the work, the theoretical foundations of the implementation of threats in telecommunication systems were considered, a model for their prediction was compiled and proposed, taking into account the emergence of new sources of threats based on the previous ones, and the implementation of a threat prediction model based on a neural network and machine learning algorithms for IPS/IDS, the system performance was assessed on the KDD-99 dataset, which includes network traffic data containing various types of threats, as a result, the developed system meets all the quality assessment requirements presented in the work.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир сложно представить без информационного взаимодействия, затрагивающего как отдельных членов общества, так и крупные организации, однако помимо очевидных получаемых преимуществ данного процесса у него есть и ряд существенных недостатков, связанных с безопасностью. Информация имеет определенную ценность, а ее потеря или компрометация через различные угрозы приводит к определенным убыткам со стороны системы, в которой данная угроза была реализована. Данная проблема в особенности актуальна для систем, где защищаемая информация циркулирует среди работников низких чинов и систем с высоким информационным оборотом, например, телекоммуникационных систем. Для предотвращения ущерба необходимо своевременно выявлять и устранять угрозы. Однако, учитывая широкий спектр угроз и стоимость оборудования невозможно принять защитные меры по отношению ко всем угрозам, а значит необходима система, предоставляющая карту возможных угроз с определенными характеристиками для оценки рисков. Это позволит уменьшить вероятность возникновения угроз на ранних этапах.

Существуют различные способы противодействия угрозам на различных этапах – до самой атаки, во время ее проведения и после. Однако такие способы берут конкретное состояние системы и время, хотя угрозы могут реализовываться как долгий и комплексный процесс. Так, забытая на секретарском столе карточка доступа к защищаемому объекту сама по себе является уязвимостью, однако пока ее не найдет злоумышленник не будет угрозы неправомерного доступа и в дальнейшем не будет угрозы компрометации данных. Получается если рассматривать конечную угрозу компрометации данных как отдельный этап противодействия угрозам, то из-за отсутствия учета причинно-следственной связи в возникновении угрозы качество оценки потенциальных угроз значительно падает. Следовательно, востребованным является недопущение реализации угрозы уже на стадии

возникновения уязвимости, либо корректирование вероятностных характеристик конечной угрозы в зависимости от ее предпосылок.

Таким образом, основное противоречие предметной области заключается в следующем: с одной стороны, необходимо повысить точность вероятностных характеристик потенциальных угроз, поскольку с развитием информационных и сетевых технологий атаки становятся все сложнее и более растянутыми как процесс, с другой стороны становится сложнее анализировать возникновение угроз с помощью существующих моделей, методик и алгоритмов прогнозирования и, следовательно, они не обладают достаточной эффективностью работы, поскольку дают недостаточно корректную вероятностную оценку угрозам.

Решением данного противоречия может являться применение высокоэффективных технологий для сферы информационной безопасности, а также в сочетании существующих и новых способов анализа и прогнозирования угроз, подкрепленных технологиями машинного обучения и анализа данных, таких как нейросети. Этим обуславливается актуальность темы дипломной работы.

Предлагаемая система прогнозирования угроз будет основываться на модели оценки угроз, несколько отличающейся от распространенных классических моделей, в которых основной целью является расчет вероятностных характеристик угроз именно в конкретном состоянии телекоммуникационной системы, не учитывая предпосылки в будущем.

Для примера использования разрабатываемой системы будет представлена ее имплементация с системами обнаружения и предотвращения вторжений IPS/IDS (Intrusion Prevention System/Intrusion Detection System).

Таким образом объектом исследования являются модели оценки и прогнозирования угроз, а также факторы, влияющие на возникновение угроз.

Предмет исследования - модель прогнозирования угроз, построенная на базе искусственной нейронной сети, позволяющая более качественно оценивать возникновение угроз в отличие от классической модели.

1 Исследование существующих методов прогнозирования угроз

1.1 Модель прогнозирования угроз

Классическая модель построена в предположении, что в любой телекоммуникационной системе (далее ТКС) существуют такие элементы безопасности, как: угроза ТКС, источник угроз, уязвимость ТКС и защитные меры. Взаимосвязь между элементами с точки зрения механизма реализации угроз показана на рисунке 1.

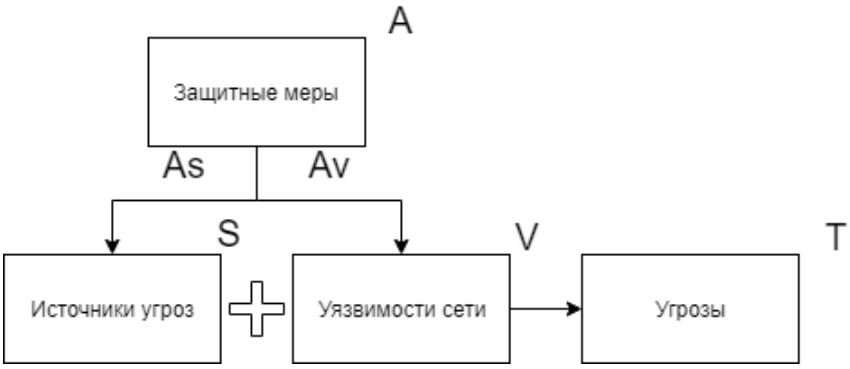

Рисунок 1 – Классическая модель реализации угроз

Возникновение угроз в соответствии с моделью на рисунке 1 можно описать следующей логической формулой:

$$(S \,\&\, !\, A_s) \,\&\, (V \,\&\, !\, A_v) \rightarrow T, \qquad (1)$$

Где символ «!» означает отрицание (то есть любое множество, не пересекающееся с данным), символ «*&*» - логическую операцию «И» над множествами (то есть пересечение двух множеств), а «->» означает соответствие элементов множеств (то есть наличие элементов одного множества приводит к возникновению элементов другого), S – множество источников угроз, V – множество уязвимостей сети, T – множество возникающих угроз, A – множество принимаемых защитных мер, As – подмножество защитных мер по отношению к источникам угроз, Av – подмножество защитных мер по отношению к уязвимостям сети.

1.2 Недостатки классической модели возникновения угроз

Хотя данная модель и является часто используемой, она обладает рядом следующих недостатков. Модель использует грубые и не однотипные формы источников и уязвимостей. С точки зрения этой модели источники угроз в сочетании с уязвимостями приводят к возникновению угроз, однако наличие лишь одного элемента предпосылки к угрозе означает отсутствие возникновения угрозы, хотя наличие уязвимости потенциально способствует реализации угрозы (это лишь вопрос времени, когда злоумышленник обнаружит уязвимость). Такой подход не учитывает также возникновение новых источников угроз в результате реализации угрозы (например, угроза компрометации данных доступа в ТКС может привести к новому источнику – несанкционированному доступу персонала ТКС).

Для ликвидации недостатков классической модели предлагается перейти к новой модели, основанной на причинно-следственной связи между источником и угрозой (см. рисунок 2). Это позволит разработать и принять Меры еще до момента первой реализации угрозы.

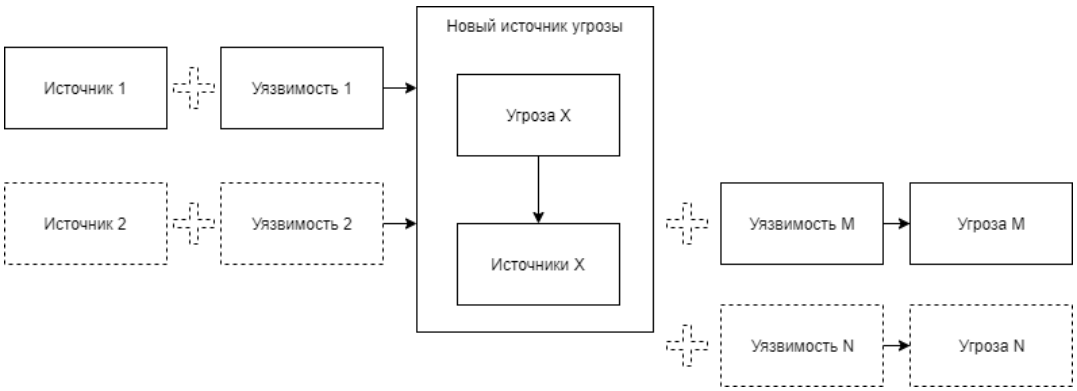

Рисунок 2 – Причинно-следственная связь источников и угроз

В общем случае одна угроза может иметь несколько источников, использующих различные уязвимости и имеющих разное влияние на ее реализацию. Аналогично один источник может приводить к различным угрозам. Такие случаи показаны на рисунке 2 пунктирной линией.

Применение закона причинности к классической модели приводит к альтернативному подходу в рассмотрении процесса возникновения угроз, что позволяет говорить и о новой модели. Согласно этому подходу, набор предпосылок приводит к реализации угроз, каждая из которых, в свою очередь, означает возникновение новой предпосылки. Будем считать угрозу и порождаемую ею предпосылку одним логическим элементом – новым источником предпосылок (далее НИП). Таким образом, логическая связь между предпосылками и угрозами описывается ориентированным графом, узлами которого являются НИП, а ребрами – следственные связи между предпосылками и угрозами соответствующих НИП. Такое представление используется в новой модели.

1.3 Анализ методик и алгоритмов прогнозирования угроз ТКС

Различные методы прогнозирования кибератак были разработаны в прошлом, но с ограниченным успехом. Прогнозирование, основанное на сетевых уязвимостях, может быть эффективным, но требует современных знаний о сети, что является сложной задачей. Эти методы предвидят будущие атаки, выявляя вредоносные действия в сетях. Тем не менее, наличие ложных или скрытых атак также может привести к ошибкам и, следовательно, их выявление не идеально. Чтобы обнаружить сложные атаки, скрытые в подавляющих данных, требуются одинаково продвинутые методы анализа и прогнозирования.

Графы атак – это один из инструментов, который исследователи использовали для прогнозирования киберугроз, подробнее о данном методе рассказано в работе [1]. Графы атак показывают большинство, если не все способы, которыми хакер может использовать уязвимости, чтобы проникнуть в сеть компьютерной системы, и эти данные можно проанализировать, чтобы увидеть, в чем заключаются слабые места системы. Альтернативой использованию графа атак является использование динамической байесовской сети (далее DBN), тип статистической модели, которая оценивает вероятности

во времени, выявляя любые паттерны, присутствующие в атаках. Байесовские сети обычно используются для прогнозирования конечной цели атаки, например, будет ли злоумышленник взломать учетную запись и пароль. Например, в системах обнаружения вторжений байесовские сети решают такие проблемы как высокая вероятность ложных срабатываний и частое вмешательство человека в систему. Благодаря способности самонастройки алгоритмов своей работы, при использовании нескольких DBN, система обнаружения вторжений способна увеличить время автономной работы без вмешательства специалиста. Однако у такой системы есть проблемы с требуемой вычислительной мощностью, сложностью реализации и переобучением [2].

Еще одна методика прогнозирования угроз – это оценка возможностей (capability), удобных случаев (opportunity) и намерений злоумышленника (COI) [3]. Этот метод широко используется в военных и разведывательных сообществах для оценки угроз. Capability позволяет предсказать, какие сервисы может предпочесть злоумышленник, основываясь на том, что он или она успешно использовали ранее. Opportunity выясняет, есть ли у злоумышленника инсайдерская информация о сети, и какие виды защиты она имеет. Намерением является изучение мотивации атакующего и социального влияния. К сожалению, прогнозирование сетевых атак с помощью анализа COI происходит на ранних стадиях их возникновения и неэффективно для атак, которые постоянно меняют стратегию, либо происходят за короткий промежуток времени.

Вышеуказанные системы прогнозирования киберугроз показали, как перспективы, так и ограничения из-за ошибок в обнаружении вторжений, неполной информации о сети и слабом представлении атак. Методы запутывания – это методы, которые используются для уклонения от обнаружения путем преднамеренно трудного понимания вредоносного кода, например, вставка шума в вредоносное ПО для уклонения от обнаружения системой идентификации вторжений. Борьба с крупномасштабными,

скоординированными атаками требует достижений на различных фронтах, включая обнаружение вторжений, корреляцию предупреждений (сбор событий, генерируемых компьютерными системами), характеристику атак, прогнозирование атак и кластеризацию узлов.

При оценке безопасности и рисков сетей также необходимо принимать во внимание поведение хакеров, что может оказаться сложной задачей из-за огромного количества потенциальных известных и неизвестных уязвимостей в сети и выбора, который злоумышленник может сделать, чтобы проникнуть в сеть.

Для обнаружения аномалий в сети предприятия популярными являются системы контроля трафика и доступа. Многие из них включают в себя модуль прогнозирования, иногда основывающийся на алгоритмах машинного обучения. Существуют следующие коммерческие продукты, которые представляют из себя либо сервисы, либо крупные системы контроля, которые необходимо разворачивать на всем предприятии: AIMS.ai, Cisco SteathWatch, PacketFense, CoScale самые известные из них. Рассмотрим их поподробнее.

AIMS.ai – система для анализа различных информационных зависимостей в предприятии и предоставляющая предупреждения о возникновении аномалий в различных потоках данных. Система предоставляет программного агента, который выполняет перечисленный выше функционал используя алгоритмы машинного обучения. Система предлагает глубокий мониторинг различных процессов предприятия и обширный набор графического вывода результатов, а также прогнозирование аномалий на самых ранних стадиях их возникновения.

Другие системы и сервисы не предлагают прогнозирование аномалий, однако имеют схожий функционал мониторинга и идентификации угроз. Также все вышеперечисленные системы связывает то, что они являются очень затратными как с точки зрения приобретения, так и имплементации в предприятие и зачастую кампании, предоставляющие эти продукты, также предлагают консультацию и услуги по обучению работы с их системами.

Единственный продукт с открытым кодом и различными модификациями от сообщества это PacketFense. По итогу, как можно заметить, среди методов и систем прогнозирования угроз отсутствуют инструменты, позволяющие аналитику информационной безопасности с легкостью и эффективно проанализировать потенциальные аномалии на предприятии, так как все автоматизированные продукты являются сложно имплементируемыми, а методы ручного анализа требуют слишком много усилий от специалиста и их эффективность зависит лишь от его навыков. Это подтверждает необходимость метода прогнозирования, способного быть достаточно автоматизированным и в тоже время не являться крупной и затратной системой.

1.4 Выявление преимуществ применения машинного обучения в области прогнозирования угроз ТКС

Важнейшими приложениями в области компьютерной безопасности, где механизмы анализа больших данных получают в настоящее время значительное распространение, являются системы мониторинга компьютерной безопасности, в том числе, системы обнаружения и предотвращения вторжений (IPS/IDS).

В современном мире стремительный технологический прогресс побудил каждую организацию принять внедрение информационных и коммуникационных технологий (далее ИКТ). Следовательно, создается среда, в которой каждое действие, направляемое через эту систему, делает организацию уязвимой, соответственно безопасность системы ИКТ поставлена под угрозу. Следовательно, это призыв к многоуровневой системе обнаружения и защиты, которая может справиться с действительно новыми атаками на систему, а также в состоянии автономно адаптироваться к новым данным. Есть несколько систем, которые могут быть использованы для экранирования системы ИКТ от уязвимостей, а именно системы обнаружения аномалий и IDS. Система обнаружения вторжений (IDS) стала существенным

слоем во всех современных ИКТ из-за стремления к обеспечению кибербезопасности. Причиной является в том числе неопределенность в поиске типов атак и увеличения сложности кибератак, соответственно ухудшается продуктивность работы систем IDS и требуется интеграция глубоких нейронных сетей (DNN).

Недостатком систем обнаружения аномалий является сложность, вовлеченная в процесс определения правил. Каждый анализируемый протокол должен быть определен, реализован и проверен на точность. Еще одна проблема, связанная с обнаружением аномалий, состоит в том, что вредная деятельность, которая подпадает под обычный шаблон использования, не распознается. Поэтому существует потребность в IDS, которая может адаптироваться к недавним новым атакам и может быть обучена, а также развернута с использованием наборов данных нерегулярного распределения. Системы обнаружения вторжений (IDS) представляют собой спектр технологий кибербезопасности, изначально разработанных для обнаружения уязвимостей и используемых против целевого хоста. Единственное использование IDS это обнаруживать угрозы. Поэтому он расположен вне полосы инфраструктуры сети и обрабатывает коммуникационный проход между отправителем и получателем данных не в реальном времени. Вместо этого системы IDS часто используют TAP или SPAN порты для анализа копии потока встроенного трафика и стараются предсказать атаку на основе ранее обученного алгоритма, следовательно, делают необходимость вмешательства человека тривиальной. В области кибербезопасности алгоритмы машинного обучения сыграли важную роль. Особенно из-за невероятной производительности и потенциала сетей глубокого обучения.

В последнее время в задачах из самых разных областей, которые были признаны неразрешимыми в прошлом, надежность применения искусственного интеллекта (ИИ) увеличилась. Глубокое обучение - это не что иное, как раздел машинного обучения, которое имитирует функции человеческого мозга, отсюда и название искусственной нейронной

сети. Концепция глубокого обучения состоит в создании иерархических представлений, которые включают в себя создание простых блоков для решения проблем высокого уровня. Отсюда рождается концепт, в котором глубокие нейронные сети и IDS объединены вместе для компенсации недостатков на различных уровнях. Кроме того, так как системы IDS являются независимыми от пропускной способности сети, простые атаки, такие как DoS, которые в первую очередь направлены на ухудшение пропускной способности сети для получения доступа к хосту, не могут ухудшить производительность системы IDS, следовательно, на этот уровень безопасности нельзя вмешаться с легкостью.

1.5 Требования к модели прогнозирования угроз ТКС

1.5.1 Требования к структуре и функционированию системы

Система «Нейросеть для прогнозирования угроз в ТКС» должна быть создана как интегрированная информационная система, состоящая из следующих подсистем:

– Подсистема сбора данных;

– Подсистема хранения данных;

– Подсистема обучения нейросети;

– Подсистема классификации угроз;

– Подсистема прогнозирования угроз;

– Подсистема ведения отчетов.

Система должна разрабатываться, как система открытого типа, что решает следующие задачи:

– обеспечение переобучения системы на новом датасете;

– обеспечение интегрирования системы под различные виды угроз;

– обеспечение переносимости программного обеспечения;

– функциональной интеграции задач, решаемых ранее раздельно.

1.5.2 Требования к надежности

Надежность функционирования системы «Нейросеть для прогнозирования угроз в ТКС» должна обеспечиваться следующими способами:

– надежностью приобретаемых технических средств;

– резервированием оборудования;

– соблюдением условий эксплуатации оборудования в соответствии с техническими условиями и проведением своевременных профилактических работ;

– применением технологии ведения информационной базы, исключающей ее утрату или искажение;

– надежностью разрабатываемого программного обеспечения;

– компетентностью специалиста, который будет ответственен за обучение нейросети;

– периодическое переобучение нейросети на новом наборе данных во избежание проблемы «запоминания зависимостей».

Разрабатываемое программное обеспечение должно позволять:

– сохранение потенциальных угроз, полученных после классификации;

– корректное преобразование входных данных для работы нейросети.

В качестве критерия надежности системы «Нейросеть для прогнозирования угроз в ТКС» и ее компонентов определяется следующие меры качества: TP (True Positive) – количество экземпляров данных, определенных как угрозы, и являющиеся таковыми; FP (False Positive) – количество экземпляров данных, определенных как угрозы, но не являющиеся таковыми; TN (True Negative) – количество экземпляров данных, не определенных как угрозы, но являющиеся таковыми; FN (False Positive) – количество экземпляров данных, не определенных как угрозы и не являющиеся таковыми. Классическим синонимом FP служат ошибки I-го рода, а FN – ошибки II-го рода. Качество прогнозирования угроз системой может

быть оценено с помощью других, более понятных человеку мер: полноты, точности, аккуратности, ошибки, F-меры.

Полнота (r) характеризует способность системы выявлять угрозы, не учитывая при этом количество неверных срабатываний. Мера полноты может быть вычислена, как доля верно определенных угроз среди всех экземпляров данных: $r = TP/\ TP+FN$.

Точность (p) характеризует способность системы выявлять только угрозы, не «захватывая» при этом легитимный трафик. Мера точности может быть вычислена, как доля верно определенных угроз среди всех определенных экземпляров данных: $p = TP/\ TP+FP$.

Аккуратность (a) характеризует способность системы делать верные решения относительно определения угроз. Мера аккуратности может быть вычислена, как доля верно определенных угроз и не угроз среди всех пользовательских сессий: $a = TP+TN/\ TP+FP+TN+FN$.

Ошибка (e) характеризует способность системы делать неверные решения относительно определения угроз. Мера ошибки может быть вычислена, как доля неверно определенных угроз и не угроз среди всех экземпляров данных: $e = FP+FN/\ TP+FP+TN+FN$.

F-мера (f), как правило, применяется для совместной оценки системы с позиции полноты и точности. F-мера может быть вычислена, как отношение удвоенного произведения полноты и точности системы к их сумме: f= $2\times p\times r$ $p+r$. С помощью указанных мер разрабатываемая система прогнозирования угроз в ТКС может быть сравнена как с ближайшими аналогами, так и с ее собственными модификациями.

2 Концепция метода решения задачи

2.1 Необходимые средства разработки

2.1.1 Обоснование выбора набора данных

Исследование ID в сетевой безопасности существует с рождения компьютерной архитектуры. Использование техник машинного обучения (далее ML) для решения задач системы IDS стало распространенным в последнее время, но данные для обучения под рукой ограничены и в основном используются только в целях разметки. Наборы данных DARPA, являются одними из наиболее полных наборов данных, доступных публично. Данные teddump, предложенные в ходе оценки DARPA ID 1998 года, являются полностью очищенными и использованы для KDDCup - 1999 г. на 5-й Международной конференции по знаниям обнаружения и интеллектуального анализа данных. Работа должна была организовать записи соединений, которые уже предварительно обработаны либо в трафик, который является нормальным, либо в одни из следующих категорий атак: «DoS», «Probing», «R2L» и «U2R».

Есть ряд интересных публикаций, где сравниваются результаты использования учебных и тестовых наборов данных в различных моделях. В одной из таких работ [4] генетический алгоритм и деревья решений были использованы для автоматической генерации правил для интеллектуальной системы, что улучшило эффективность системы IDS. Также было предложено интегрированное использование нейронных сетей в IDS. Например, в работе [5] было рассмотрено применение рекуррентных нейронных сетей (RNNs) и произведено сравнение производительности архитектуры нейронной сети для обнаружение статистической аномалии в наборах данных в четырех разных сценариях.

Хотя наборы данных KDDCup-'99' имеют различные проблемы в работе [6] утверждается, что они все еще являются эффективным эталоном набора

данных, который является общедоступным и удобным для сравнения различных методы обнаружения.

Программа DARPA по оценке личности в 1998 году была под руководством и подготовкой Lincoln Labs из MIT. Ее главной целью являлся анализ и проведение исследований в области идентификации. Был подготовлен стандартизированный набор данных, который включал различные типы вторжений, имитирующие полевую среду и был выпущен в общий доступ. В конкурсе «Обнаружение вторжений» KDD 1999 года набор данных был хорошо очищенной версией датасета от Lincoln Labs. Основным недостатком было то, что они потерпели неудачу при проверке своих данных для симуляции трафика реальной сети. Независимо от этих критических замечаний, набор данных KDDCup-'99' использовался многими как эффективный набор данных для бенчмаркинга алгоритмов IDS в последующие года. Несмотря на критику в направлении создания набора данных, в ходе подробного анализа содержания, выявили неоднородности и смоделировали аномалии в форме данных сетевого трафика, которые позволили дать обучающимся моделям лучшее понимание процесса идентификации угроз. Причины, по которым классификаторы машинного обучения имеют ограниченные возможности в идентификации атак, которые принадлежат к категориям R2L, U2R в наборах данных KDDCup-'99 ' были обсуждены в работе Сабнани, Махешкумар и Гурзель Серпен «Почему алгоритмы машинного обучения не могут обнаружить угрозу в наборе данных обнаружения вторжений KDD». Они пришли к выводу, что невозможно получить приемлемую частоту обнаружения с использованием классического машинного обучения. Также заявлена возможность получения высокой частоты обнаружения в большинстве задач, если производить очищение и расширение набора данных путем объединения наборов тестовых данных и данных для обучения. Тем не менее, объективно эффективного подхода обнаружено не было.

2.1.2 Описание использованного набора данных

Начиная с года выпуска набора данных KDD-99, он наиболее широко используется для оценки IDS. Этот набор данных сгруппирован для почти 4900000 сессий, которые включают в себя 41 признак. Атаки были классифицированы как указано ниже:

• Атака отказа в обслуживании (DoS): вторжение, когда злоумышленник стремится сделать хоста недоступным для функционирования, заполняя целевую машину огромными количествами запросов и, следовательно, перегружая хоста.

• User-to-Root-Attack (U2R): маневр со стороны злоумышленника, когда он, входя в систему как обычный пользователь эксплуатирует уязвимости системы, чтобы получить доступ супер-пользователя.

• Remote-to-Local-Attack (R2L): вторжение, в котором злоумышленник может отправлять пакеты данных к цели, не имея учетной записи пользователя на этой машине, а также пытается использовать уязвимость для получения локального доступа, скрываясь в качестве существующего пользователя целевой машины.

• Зондовая атака: тип атаки, в которой злоумышленник пытается собрать информацию об устройстве сети и конечная цель этого состоит в том, чтобы пройти через брандмауэр и получить root-доступ.

Набор KDDCup-'99' подразделяется на следующие три группы.

- Основные признаки: к этой группе относятся атрибуты, полученные из соединений TCP / IP. Большинство из этих признаков приводит к неявной задержке обнаружения угрозы.

- Особенности трафика: к этой группе относятся признаки, рассчитанные по окну времени. Они могут быть разделены на 2 группы:

Признаки «Тот же хост»: соединения, имеющие одинаковый идентификатор конечного хоста и попадающие в эту категорию каждые 2 секунды. Служат для расчета статистики поведения протокола.

Признаки «Та же услуга»: соединения, которые поступают от идентичного сервиса и попадающие в эту категорию каждые 2 секунды.

- Признаки контента: в основном зондирующие атаки и DoS атаки имеют по крайней мере какую-то частотную последовательность шаблонов вторжений в отличие от атак R2L и U2R. Это по той причине, что они включают в себя несколько соединений к одному набору хостов за короткий промежуток времени, пока другие 2 вторжения интегрированы в разделенные пакеты данных, в которые обычно входит только одно соединение. Для обнаружения этих типов атак, нам нужны некоторые уникальные признаки, с помощью которых мы сможем искать какое-то нерегулярное поведение. Они называются признаками контента.

2.1.3 Обоснование выбора языка программирования и необходимых библиотек

Python — высокоуровневый язык программирования, отличающийся высокой производительностью и читаемостью кода. Данный язык программирования является одним из самых быстродействующих, это значит, что при должной реализации на языке Python можно писать код компактно и быстро решать довольно трудные задачи. Практически все библиотеки и фреймворки для новых и быстроразвивающихся отраслей информационных технологий пишутся для этого языка. Нейросети не являются исключением и поэтому весь необходимый для реализации проекта инструментарий написан под python, а значит и использоваться в дальнейшем будет именно этот язык программирования.

Для анализа и интерпретации данных, а также для написания нейросети и использования ML-алгоритмов нам понадобится определенный инструментарий и библиотеки.

Pandas — это пакет функций для языка Python с открытым исходным кодом. Наиболее широко данный пакет используется в аналитике данных и задачах машинного обучения. Pandas является надстройкой другого пакета – Numpy, который поддерживает обработку многомерных массивов. Пакет

Pandas позволяет с легкостью решить долгие, повторяющиеся задачи, связанные с обработкой данных, такие как очистка, нормализация и визуализация данных, статистический анализ, объединение и изменение размерностей массивов данных. В общем, с пакетом Pandas, можно сделать все, что заставляет ведущих мировых специалистов в области обработки данных голосовать за Pandas, как за лучший общедоступный инструмент для анализа и обработки данных.

Pandas предоставляет две основные структуры, куда помещаются данные, это классы Series и Dataframe. Первый представляет собой массив фиксированного типа с лишь одной размерностью. Второй класс предоставляет двухмерную структуру данных, которая описывается строками и столбцами. Класс DataFrame наиболее часто используемая структура, так как она позволяет представить наборы данных в виде признакового описания отдельных объектов, которые размещаются в строки, а столбцы соответственно представляют различные признаки этих объектов.

Seaborn — библиотека для визуализации данных, созданная на базе matplotlib. Она предоставляет высокоуровневый интерфейс для вывода привлекательных и информативных статистических графиков. Seaborn помогает изучить данные для их большего понимания. Функции построения графиков данной библиотеки работают с фреймами данных и массивами, содержащими целые наборы данных, и внутренне выполняют необходимое семантическое отображение для большей информативности на выведенных графиках.

Scikit-learn – это библиотека для машинного обучения на языке программирования Python с открытым исходным кодом. С помощью нее можно реализовать различные алгоритмы классификации, регрессии и кластеризации, в том числе алгоритмы SVM, случайного леса, k-ближайших соседей и DBSCAN, которые построены на взаимодействии библиотек NumPy и SciPy с Python.

Достоинствами данной библиотеки являются:

- Простые и эффективные инструменты для data mining и data analysis;

- Удобный доступ к необходимым компонентам;

- Построен на NumPy, SciPy и Matplotlib;

Внедрение моделей машинного обучения менее устрашающий и сложный процесс чем может показаться на первый взгляд, облегчение разработки таких сложных архитектур предоставляет фреймворк TensorFlow, разработанный компанией Google. TensorFlow упрощает процесс сбора данных, составление архитектуры и обучение моделей, их обслуживание и тестирование.

Keras - это API глубокого обучения, написанный на Python, работающий поверх платформы машинного обучения TensorFlow . Он был разработан с упором на возможность быстрого экспериментирования. Возможность как можно быстрее перейти от идеи к результату - ключ к хорошему исследованию.

С помощью вышеперечисленного инструментария можно не только построить необходимую модель прогнозирования, но еще и удобно выводить необходимые графики и информацию в ходе обучения и оценки модели. Также удобство фреймворка Keras позволяет специалисту информационной безопасности с легкостью экспериментировать над моделью прогнозирования угроз и тестировать свои собственные варианты архитектуры.

2.2 Применение алгоритмов машинного обучения к модели прогнозирования угроз ТКС

Предложенная в разделе 1 модель достаточно хорошо коррелирует с распространенной моделью искусственной нейронной сети с точки зрения предназначения, структуры, логики элементов и связей. Также, зачастую при прогнозировании и оценки угроз применяется ручной анализ собранных данных или состояния системы, что плохо сказывается на результатах. Нейросеть же способна повысить точность вычислений, работать, учитывая гораздо большее количество признаков, а также находить скрытые

зависимости в данных, которые найти с помощью ручного анализа практически невозможно. Кроме того, нейросети довольно легко и эффективно внедряются в другие системы обеспечения кибербезопасности, например, в систему IPS/IDS, как было описано в предыдущем разделе.

Реализация новой модели с точки зрения нейронной сети подразумевает 2 задачи:

- Задача классификации (регрессия);

- Установка причинно-следственной связи между НИП.

2.2.1 Задача классификации

Мы установили требования для нейросети, а именно, что она должна решать 2 поставленные задачи, однако в силу архитектуры и свойств нейросетей они являются однозадачными, а значит для реализации модели прогнозирования угроз понадобится по нейросети на каждую задачу. Но для решения задачи классификации, то есть на основе входных признаков угрозы определить ее вероятность, нам важна интерпретация результата.

Интерпретация результата - Раздел Data Science, позволяющий понять причины выбора MachineLearning-моделью того или иного решения. Существует два основных направления исследований:

- Изучение модели как «черного ящика». В алгоритм загружаются примеры, после чего он их анализирует, сравнивая признаки и выводы алгоритма, делая выводы о приоритете каких-либо из них. В случае с нейросетями обычно применяют именно черный ящик;

- Изучение свойств самой модели. Тут в основном изучаются признаки, на основе которых обучается модель, для определения степени их важности. Чаще всего применяется к алгоритмам, основанным на методе решающих деревьев.

Например, при прогнозировании угроз ТКС признаки объектов – это данные сетевых подключений, программного обеспечения, показатели камер, и так далее, а ответы – это ответы на вопрос, какова вероятность возникновения угрозы. Естественно, заказчика интересует не только прогноз

самой угрозы, но и интерпретация результата, т. е. причины ее появления для их последующего устранения. Это может быть долгое отсутствие технического обслуживания компьютеров, ненастроенные сетевые фильтры, или недостаточный контроль охраняемой зоны. Потому в рамках проекта прогноза угрозы в ТКС должна быть не просто создана ML-модель, но и проделана работа по её интерпретации, т. е. по выявлению факторов, влияющих на возникновение угрозы. На рисунке 3 показана зависимость интерпретируемости результата модели от выбора алгоритма машинного обучения.

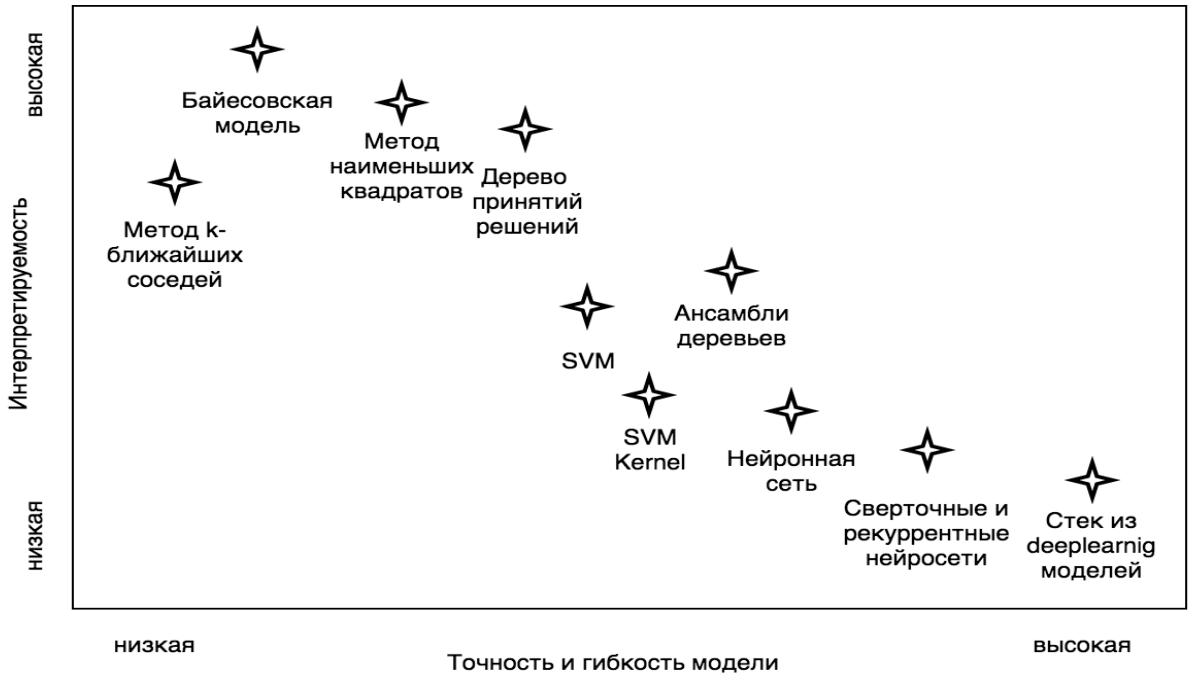

Рисунок 3 - Зависимость гибкости алгоритма машинного обучения и интерпретируемости полученной модели

По итогу для решения задачи классификации предлагается протестировать можно ли добиться необходимого результата, используя дерево решений или же придется использовать нейросети на обоих этапах проектирования системы. Важно отметить, что в процессе внедрения системы в эксплуатацию, необходим специалист в области машинного обучения, который будет на основе заранее собранных предприятием данных подбирать

алгоритм, подробности о рекомендациях по эксплуатации и внедрению системы представлены в разделе 3.4.

2.2.2 Дерево принятий решений в решении задачи классификации угроз

Дерево принятий решений является алгоритмом машинного обучения. Самое общее и не строгое определение машинного обучения звучит так: говорят, что компьютерная программа обучается при решении какой-то задачи из класса T, если ее производительность, согласно метрике P, улучшается при накоплении опыта E.

Далее в разных сценариях под T, P, и E подразумеваются совершенно разные вещи. В нашем случае задача T это задача бинарной классификации – прогнозирование бинарного признака объекта (0 или 1) на основании прочих его признаков.

Под опытом E понимаются данные, и в зависимости от того размечены они или нет принимают решение о методе обучения модели, с учителем или без него. В задачах обучения без учителя имеется выборка, состоящая из объектов, описываемых набором признаков. В задачах обучения с учителем вдобавок к этому для каждого объекта некоторой выборки, называемой обучающей, известен целевой признак.

Наконец, третья абстракция в определении машинного обучения – это метрика оценки производительности алгоритма P. Для примера – самая простая и часто используемая метрика для оценки производительности модели - это аккуратность (accuracy), которая отражает количество правильных ответов к общему числу ответов.

Дерево решений как алгоритм машинного обучения - объединение логических правил вида "Значение признака a меньше x & значение признака b меньше y, следовательно, пример относится к классу 1". Для того, чтобы понять какие коэффициенты x, y и признаки a, b брать для сравнения необходимо ввести метрику – для объяснения работы дерева решений подойдет энтропия.

Энтропия Шеннона определяется для системы с N возможными состояниями следующим образом:

$$S = -\sum_{i=1}^{N} pi \ \log_2 pi, \qquad (2)$$

где pi – вероятности нахождения системы в i-ом состоянии. Это очень важное понятие, используемое в различных областях, таких как физика, информатика и других. Опуская некоторые предпосылки введения данного понятие, энтропия по сути характеризует степень хаоса в системе. То есть чем выше энтропия, тем менее упорядочены элементы в выборке. Этот параметр будет основным при принятии решений «Деревом».

Поскольку энтропия – по сути степень неопределенности в системе, уменьшение энтропии называют приростом информации. Формально прирост информации (information gain, IG) при разбиении выборки по признаку Q определяется как:

$$IG(Q) = \ S_0 - \ \sum_{i=1}^{q} \frac{N_i}{N} S_i, \qquad (3)$$

где q – число групп после разбиения, Ni – число элементов выборки, у которых признак Q имеет i-ое значение.

В основе популярных алгоритмов построения дерева решений, таких как ID3 и C4.5, лежит принцип жадной максимизации прироста информации. Его суть заключается в том, что на каждом шаге выборка разделяется таким образом, что элементы становятся максимально упорядоченными и энтропия постепенно минимизируется. В конце концов, когда она становится приблизительно равна нулю, алгоритм заканчивает свою работу и «Дерево» считается обученным. Однако с остановкой алгоритма не все так просто, во избежание переобучения модели, «Дерево» необходимо заранее «отсекать» или уменьшать количество «листьев», созданных разделением выборки.

При прогнозировании количественного признака идея построения дерева остается та же, но меняется критерий качества на дисперсию вокруг среднего:

$$D = \frac{1}{l}\sum_{i=1}^{l}(y_i - \frac{1}{l}\sum_{i=1}^{l} y_i)^2, \quad (4)$$

где ℓ – число объектов в листе, yi – значения целевого признака. Попросту говоря, минимизируя дисперсию вокруг среднего, мы ищем признаки, разбивающие выборку таким образом, что значения целевого признака при каждом разделении примерно равны.

2.2.3 Визуализация дерева решений

Неформально, задача классификации – построить какую-то "хорошую" границу, разделяющую 2 класса (красные точки от желтых). Обучение модели в этом случае сводится к тому, как выбрать хорошую разделяющую границу. Возможно, прямая будет слишком простой границей, а какая-то сложная кривая, огибающая каждую красную точку – будет слишком сложной и будем много ошибаться на новых примерах из того же распределения, из которого пришла обучающая выборка, например, на рисунке 4 (n-мерный случай – гиперплоскость) ситуация, где классификатор обладает высокой точностью, четко разделяя область двух классов.

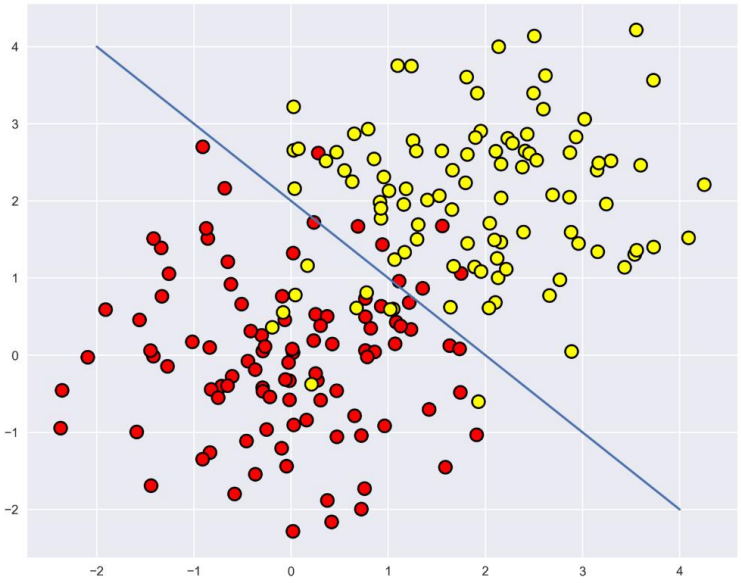

Рисунок 4 – Пример визуализации алгоритма классификации

Алгоритм классификации дерева решений и внешний вид дерева представлены на рисунках 5 и 6.

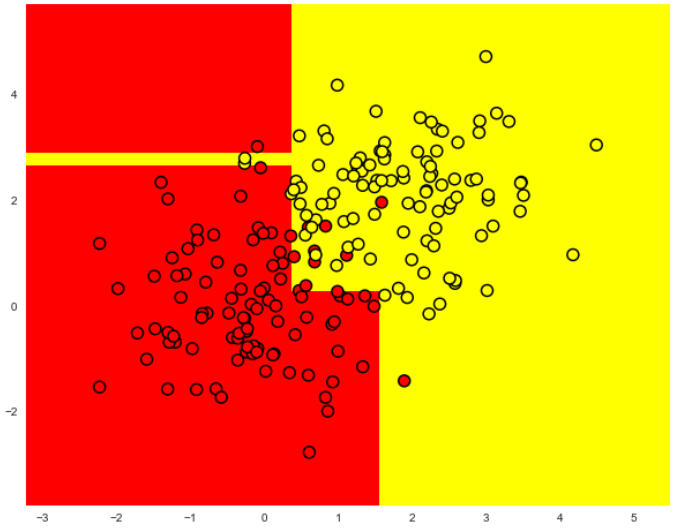

Рисунок 5 – Визуализация алгоритма классификации дерева

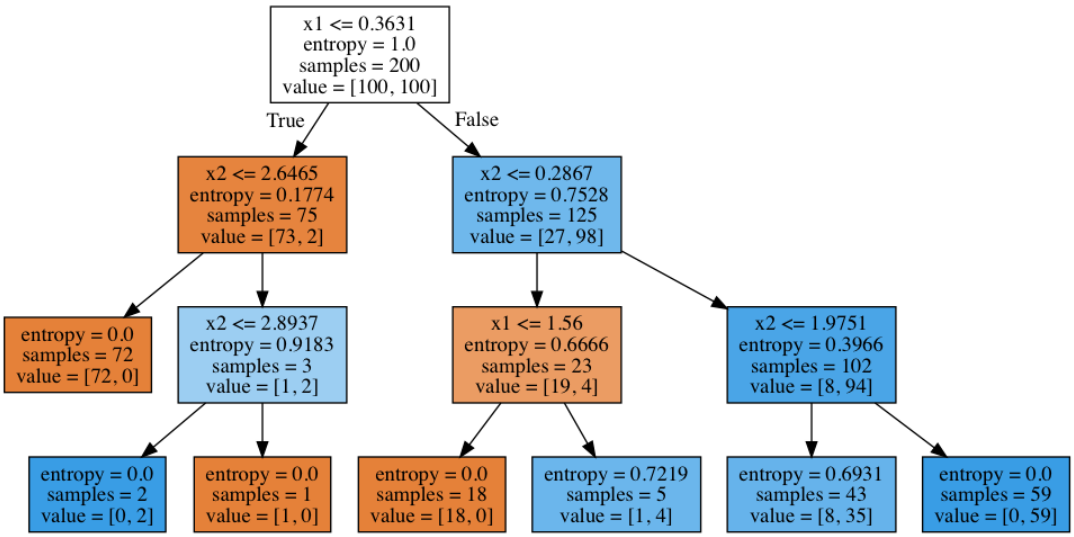

Рисунок 6 – Визуализация дерева решений

При такой визуализации чем больше объектов одного класса, тем цвет вершины ближе к темно-оранжевому и, наоборот, чем больше объектов второго класса, тем ближе цвет к темно-синему. В начале объектов одного класса поровну, поэтому корневая вершина дерева – белого цвета.

2.3 Глубокие нейронные сети

Как было определено ранее, альтернативным вариантом для реализации классификатора являются глубокие нейронные сети. Как показывает практика, они выдают лучшие результаты по классификации чем алгоритмы машинного обучения, однако требуют дополнительных мощностей для обучения.

2.3.1 Установка причинно-следственной связи между НИП

Как говорилось ранее, новая модель прогнозирования угроз имеет схожую логику элементов, что и нейронные сети. Перед тем, как перейти к взаимосвязи НИП-ов между собой (в чем нам поможет именно нейросеть) введем несколько понятий и покажем, что из себя представляет нейросеть.

Искусственная нейросеть является математической моделью, построенной по принципу организации и функционирования биологических нейронных сетей [7], [8]. Она состоит из соединенных и взаимодействующих простых процессоров – искусственных нейронов. Каждый нейрон способен получать сигналы от одних нейронов, обрабатывать их, возбуждаться и

посылать новый сигнал другим. Начальные Предпосылки предлагаемой модели с точки зрения нейросети являются входным возбуждением, а конечные Угрозы – ее выходной реакцией. НИП с точки зрения нейросети подобны нейронам, реагирующим на возбуждения (Предпосылки) и передающим возбуждения далее (создавая Угрозы). Для подтверждения этой аналогии на рисунке 7 приведена модель нейрона.

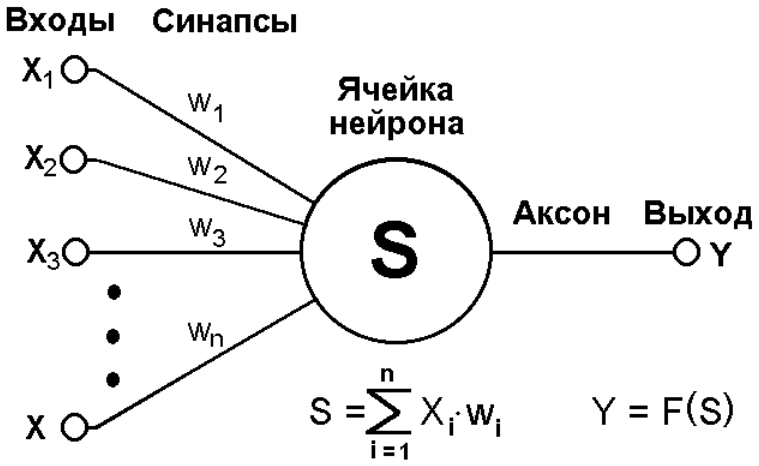

Рисунок 7 – модель нейрона

Согласно рисунку 7 нейрон представляет собой узел сети, имеющий множество входов со значениями {x1, x2, x3… xn}, множество внутренних весов {w1, w2, w3… wn} и выход со значением Y, вычисляемым с помощью функции F(S) по комбинации взвешенных значений xn и wn. Функция F(S) отражает чувствительность нейрона к возбуждению и в новой модели описывает механизм возникновения НИП. Будучи соединенными в сложную систему, нейроны вместе способны выполнять задачи по прогнозированию порождаемых НИП, что в контексте новой модели соответствует выявлению картины конечных Угроз по имеющимся начальным Предпосылкам (смотрите рисунок 8).

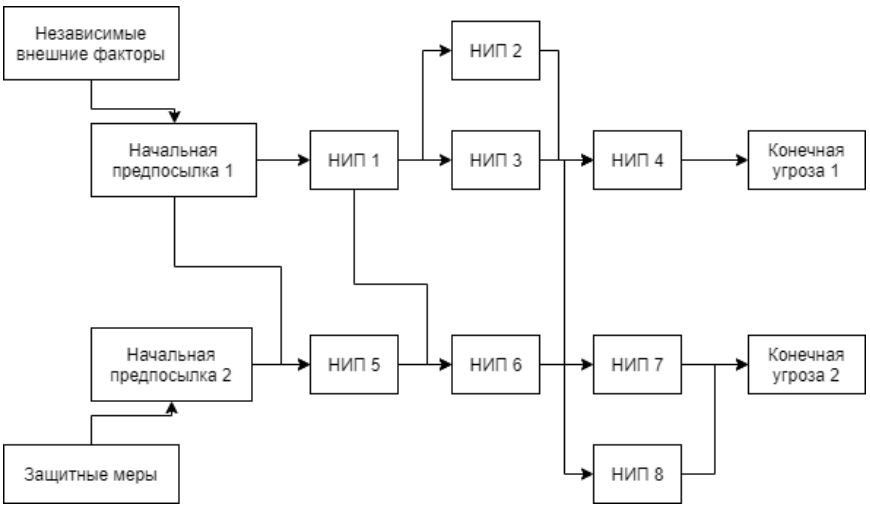

Рисунок 8 – Модель прогнозирования угроз

2.3.2 Применение рекуррентной нейросети

RNN - слой нейросети, обрабатывающий последовательности целиком и имеющий элемент “памяти” для установки зависимостей в этой последовательности.

Идея рекуррентных нейросетей (RNN) заключается в последовательном использовании информации [9]. В традиционных нейронных сетях подразумевается, что все входы и выходы независимы. Но для нашей задачи это не подходит. Если вы хотите предсказать следующую угрозу в причинно-следственной цепочке, нужно учитывать предшествующие ей угрозы и предпосылки. RNN называются рекуррентными, потому что они выполняют одну и ту же задачу для каждого элемента последовательности, причем выход зависит от предыдущих вычислений. Еще одна интерпретация RNN: это сети, у которых есть «память», которая учитывает предшествующую информацию. Теоретически RNN могут использовать информацию в произвольно длинных последовательностях, но на практике они ограничены лишь несколькими шагами.

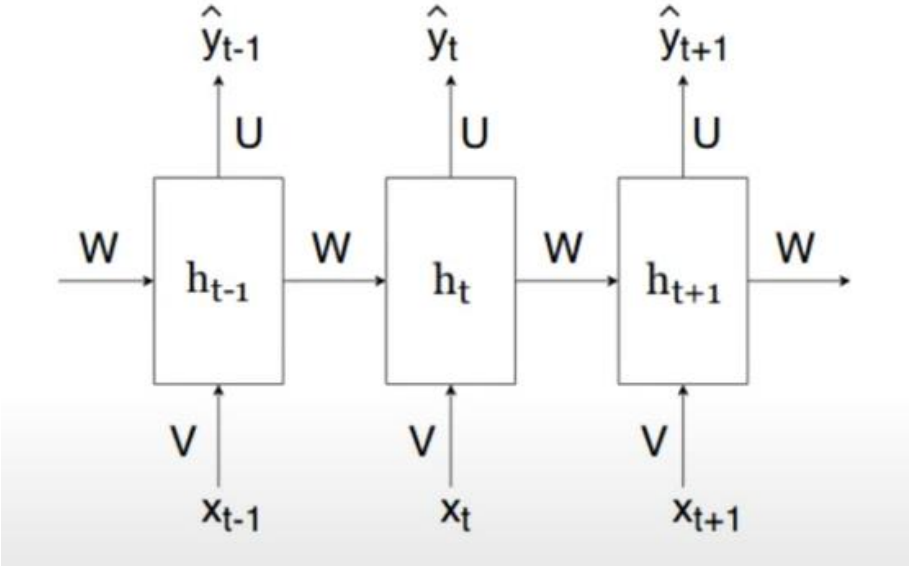

Рисунок 9 – Рекуррентная нейросеть и ее развертка

Как показано на рисунке 9 модель RNN это по сути последовательность ячеек, где в каждой ячейке выполняется одна и та же операция, ячейки пронумерованы. Операции в ячейках описываются ht - скрытым состоянием в момент t, на основе этого h затем делается классификация ý. Формула для расчета:

$$h_t = f(Vx_t + Wh_{t-1} + b) \quad (5)$$

$$\acute{y}_t = g(Uh_t + \grave{b}) \quad (6)$$

где f - функция активации, g - предположим Softmax

В RNN обычно используют гиперболический тангенс. RELU не стоит использовать, так как есть некоторые проблемы с обработкой последовательности, например в RNN мы хотим, чтобы функция имела “нулевой момент ожидания”, а функция RELU сдвигает распределение и если это будет происходить на каждом шаге t, то распределение может уйти куда то в бесконечность на определенном моменте ожидания.

Целью обучения нейросети RNN также является уменьшение функции потерь - минимизация суммарных потерь. Функция потерь считается на

каждом шаге t, то есть для каждого t существует полученная ý которая сравнивается с истинной y. Минимальное значение функции потерь от аргументов ý и y и есть результат обучения. Основные параметры функции:

- матрица V переводит экземпляры x в пространство размерности скрытого состояния.

- матрица W - матрица перехода h_{t-1} к размерности h_t.

- свободный коэффициент b - вектор сдвига.

У RNN есть некоторые проблемы, возникающие при подсчете градиента. Это его взрыв и затухание. Когда отношение производной h_t к производной h_{t-1} стремится к бесконечности – это взрыв градиента, когда стремится к нулю – это затухание. Для борьбы с взрывом градиента применяется метод gradient clipping. Метод выглядит довольно примитивно, так как вся его суть заключается в том, что мы просто вводим какой-то параметр, выше которого значение градиента не может быть, то есть мы вручную устанавливаем диапазон норм градиента, однако метод является рабочим и постоянно применяющимся при оптимизации RNN нейросетей. Для решения проблемы с затуханием градиента существуют дополнительные модификации RNN архитектуры, в частности к модели прогнозирования угроз был применен один из таких методов – LSTM сеть.

2.3.3 LSTM сеть

LSTM - дополнительный путь течения информации (состояние сети C) для борьбы с затуханием градиента [10]. Если градиент функции потерь равен нулю, соответственно нету зависимости функции от параметров, соответственно с помощью C_t к L добавляется дополнительная информация, которая зависит от этих параметров, тем самым меняя параметры U, W, V будет изменятся и результат функции потерь.

Как работают эти дополнительные ячейки показано на рисунке 10:

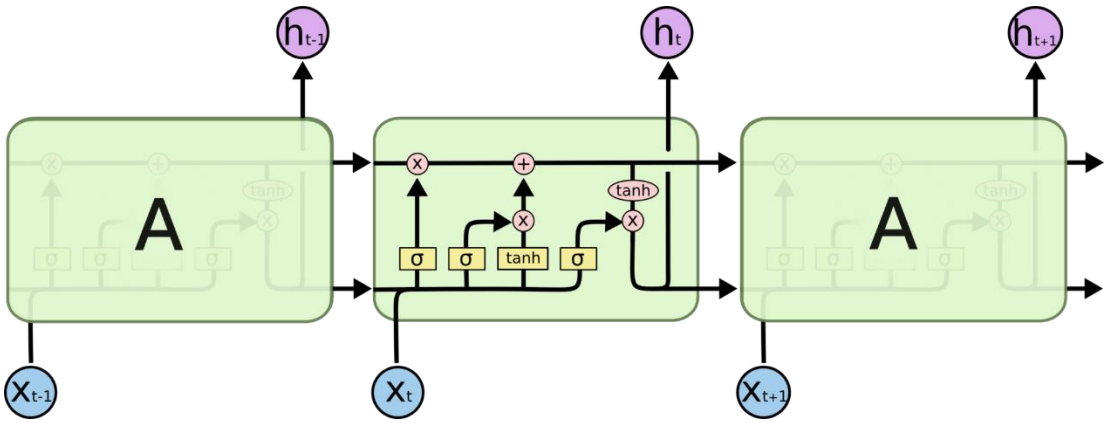

Рисунок 10 – LSTM ячейка

Состояние сети обновляется в каждой клетке (очень аккуратно, с возможностью восстановления). Для начала необходимо учитывать “старую информацию”, то есть C_{t-1}, для этого мы обрабатываем вход с помощью сигмоиды, получая значение от 0 до 1, и уже C_{t-1} умножаем на это значение, тем самым уменьшая C_{t-1}, определяя ее вклад в значение следующего шага (часть старой информации тем самым забывается - это называется Forget gate).

Далее формируется новая информация - этот элемент называется Input gate. Полученные данные на входе ячейки преобразуются с помощью гиперболического тангенса, тем самым генерируя некий сигнал, далее этот сигнал умножается на сигмоиду, полученную на предыдущем шаге. Сгенерированный сигнал в итоге складывается с результатом из Forget gate, тем самым получаем C_t, соединив то что было раньше с тем что получили сейчас.

Последний этап - Output gate. Мы хотим уже из C_t получить h_t, то есть полученное новое состояние сети из LSTM ячейки скрещивается с h_{t-1} и формирует h_t. Происходит это следующим образом: мы все ту же сигмоиду домножаем на гиперболический тангенс уже от C_t, тем самым определяя h_t.

Как и любой архитектуре нейросети в RNN и в частности LSTM можно добавлять слои для получения более абстрактного представления на каждом шаге. Тут первый слой остается без изменений, а в последующих мы подаем на вход уже не x_t, а h_t предыдущего слоя. В определении yt можно брать не

только h_{t2}, а можем брать с каждого слоя выход, конкатенировать их и уже определять yt. Помимо слоев, можно также менять направление.

Во время обучения используется метод обратного распространения ошибки во времени, он подразумевает, что мы разворачиваем каждое скрытое состояние в определенный момент времени, тем самым представляя ее как сеть прямого распространения. Собственно, выполняется вычисление скрытых состояний и выходов развернутой сети, а также подсчитываются градиенты функций активации. Впоследствии, после подсчета функции потерь, корректируются веса нейросети. Сложность данных вычислений пропорциональна длине входной последовательности данных.

Модель на рисунке 9 имеет выходы на каждом шаге, но, в зависимости от задачи, они могут не понадобиться. Например, при определении эмоциональной окраски предложения, целесообразно заботиться только о конечном результате, а не о окраске после каждого слова. Аналогично, нам может не потребоваться ввод данных на каждом шаге. Главной особенностью является свободная модификация ячеек скрытого состояния в RNN, что позволяет добиться гибкости в модели и получать результат в зависимости от разных потребностей.

Глубокие нейронные сети (DNN) - это искусственные нейронные сети (ANN) с многослойной архитектурой, состоящей из нескольких слоев ввода-вывода. Они могут моделировать сложные нелинейные отношения и могут генерировать вычислительные модели, где объект выражен через слоистую композицию примитивы. Ниже мы примерно рассмотрим простые DNN и применение функции ReLU (rectified linear unit) и почему она предпочтительнее других функций активации.

Хотя традиционные алгоритмы машинного обучения являются линейными, глубокие нейронные сети укладываются в возрастающую иерархию сложности, а также абстракции. Каждый слой применяет нелинейное преобразование над его входом и на выходе создает матрицу значений. Проще говоря, исходные параметры принимаются входным слоем и

передаются на первый скрытый слой. Эти скрытые слои выполняют математические расчеты на входах. Одна из проблем в создании нейронных сетей это установить оптимальное количество скрытых слоев и количество нейронов для каждого слоя. Каждый нейрон имеет функцию активации, которая используется для приведения в нужной числовой форме выхода из нейрона. «Глубокое» в глубоком обучении означает наличие более одного скрытого слоя. Пока выход не достигнет приемлемого уровня точности, итерации обучения продолжаются.

2.3.4 Выбор функции потерь и оптимизатора

Выбор функции потерь зависит от специфики задачи. Нейросети изучают сопоставление входов и выходов с учетом загруженных экземпляров, поэтому и функция потерь должна соответствовать постановке конкретной задачи для последующего моделирования, обусловленной исходной выборкой. Самая стандартная функция для бинарной классификации, а именно с ней связано прогнозирование и идентификация угроз, это кросс-энтропия. Кросс-энтропия вычисляет оценку, которая суммирует среднюю разницу между фактическим и прогнозируемым распределением вероятности для отнесения к классу 1. Эта оценка минимизируется в ходе обучения и по итогу в идеальном случае равна 0.

Так как операция по минимизации функции потерь является затратной, к модели обычно применяют различного вида оптимизаторы. Главная их цель – ускорение обучения, за счет небольших изменений гипер-параметров модели, например, весов и скорости обучения. В связке с кросс-энтропией рекомендуется использовать оптимизатор Adam, так как у него высокая вычислительная эффективность, простая реализация и небольшие требования к памяти. Основная суть работы оптимизатора, с математической точки зрения – это то что алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент, а два его параметра beta управляют скоростью затухания этих градиентов. Таким образом у модели появляется

более ясное представление как изменять свои гипер-параметры и обучение становится более эффективным.

2.3.5 Выбор функции активации

Зачастую нейросети используют сигмоидальную или тангенциальную функции активации в скрытых слоях [11]. Однако ReLu в большинстве задач является более эффективным в способности ускорить весь процесс обучения по сравнению с другими функциями активации.

Сигмоида выражается следующей формулой:

$$\sigma(x) = 1 / (1 + e - x) \quad (7)$$

Эта функция преобразует входное вещественно число в число, соответствующее интервалу от 0 до 1. Основной особенностью сигмоиды является то, что она преобразует большие по модулю отрицательные числа в число, приближенное к нулю, а большие положительные числа приводит к единице. Соответственно сигмоидой удобно интерпретируется насыщенность нейрона, это свойство в последнее время используют на выходных слоях для определения классификации. Однако из-за этого же свойства сигмоидой и перестали пользоваться в скрытых слоях, на это есть 3 причины:

- Насыщение данной функции активации приводит к затуханию градиента в скрытых слоях. Это происходит за счет того, что при насыщении на участках 0 или 1, градиент в этих местах становится близок к нулю, а учитывая, что во время метода обратного распространения ошибок локальный градиент на этих участках умножается на общий, то получится что обнулится и общий градиент, и сигнал не будет проходить через сеть.

- Второй проблемой насыщения является то, что при больших исходных весах, нейроны сразу перейдут в насыщение и нейросеть будет плохо обучаться.

- Третья проблема возникает из-за того, что выход сигмоиды не центрирован относительно нуля. Поскольку градиентный спуск чувствителен

к входным данным, данное свойство негативно влияет на его динамику, поскольку все последующие слои будут нести значения не центрированные относительно нуля. В результате из-за скачков между положительными и отрицательными значениями градиента веса обновляются соответственно зигзагообразной динамике, что оказывает негативное влияние на обучение.

Гиперболический тангенс схож с сигмоидой, однако его отличие заключается в интервале, к которому приводится число, он находится в диапазоне от -1 до 1. Подобно сигмоиде, гиперболический тангенс может насыщаться. Но в данном случае решена проблема с центрированием.

В последние годы большую популярность приобрела функция активации под названием «выпрямитель». Нейроны с данной функцией активации имеют название ReLU (rectified linear unit). Принцип работы ReLU заключается в реализации порогового перехода в нуле. Преимущества данной функции активации заключается в быстроте вычисления, по сравнению с сигмоидой и гиперболическим тангенсом, также при ее использовании повышается скорость сходимости стохастического градиентного спуска из-за линейной природы вычислений и отсутствия насыщения. Единственная проблема, связанная с применением ReLU заключается в выведении из строя нейронов при слишком высокой скорости обучения, однако обычно этот гиперпараметр не выставляется в настолько большие значения. Собственно, учитывая востребованность ReLU в применении к моделям машинного обучения она и будет выбрана в качестве функции активации в скрытых слоях модели прогнозирования угроз ТКС.

2.3.6 Батч-нормализация и ее применение к модели

Как говорилось ранее, цель обучения нейросети – это минимизация функции потерь, также называемой целевой функцией. Ее минимизация происходит за счет подсчета градиента по ее основным параметрам, тем самым мы получаем направление движения функции и корректируя соответственно параметры модели мы постепенно, шаг за шагом приближаемся к необходимому нам результату, который будет отражаться в

точности модели и правильно подобранных весах (которые можно потом сохранить и уже использовать модель для прогнозирования). Рассмотрим работу градиентного спуска поподробнее. Предположим имеется целевая функция, представленная на рисунке 11.

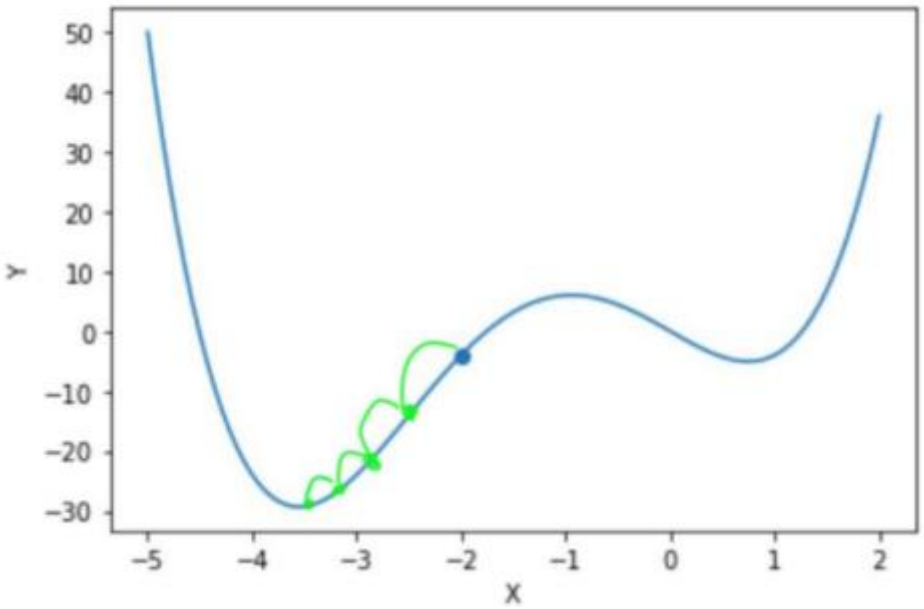

Рисунок 11 – Пример работы градиентного спуска

По результатам итерации модели мы получили некоторое значение x (оно отмечено на графике синей точкой). Получив это некоторое случайное значение, мы начинаем движение от него к точке минимума, каждый шаг алгоритма вычисляя производную функции в точке, в которой сейчас находимся и сдвигая эту точку в направлении минимума. Вычислять производную на каждом шаге нужно, потому что мы в какой-то момент можем "перепрыгнуть" через точку минимума, тогда производная поменяет знак и мы поймем, что нужно двигаться обратно. Остается два вопроса: на какую величину Δx уменьшать и когда останавливать алгоритм.

Давайте попробуем выбрать Δx. Это нетривиально, ведь мы не видим графика функции и не знаем, насколько далеко точка минимума. А численное значение производной нам показывает только факт убывания функции и скорость убывания (угол наклона графика), но не расстояние до точки минимума. Если мы зафиксируем некоторую величину Δx и на каждом шаге алгоритма будем сдвигать x на одинаковую величину Δx в направлении

минимума, то это может занять очень много времени (например, когда точка минимума в точке x=1, а мы стоим в точке x=1001 и Δx=1, нам потребуется 1000 шагов алгоритма, чтобы дойти до минимума). Плюс, если мы стоим в точке x=1.5, то, сдвинувшись на Δx=1 по направлению к точке минимума мы попадем в точку x=0.5, далее опять сдвинувшись на Δx=1 по направлению к точке минимума мы попадем в точку x=1.5 и будем так ходить туда-сюда, не приближаясь к точке минимума x=1 ближе.

Давайте еще раз внимательно посмотрим на график функции (выше) и увидим, что чем ближе точка к минимуму, тем плавнее график. Более формально, что чем ближе точка к минимуму, тем меньше скорость убывания/возрастания функции, а значит, тем меньше модуль значения производной.

Мы можем это использовать. Давайте зафиксируем некоторое маленькое число εε и каждый шаг алгоритма будем двигать xx по направлению к минимуму на шаг ε·|F′(x)|. Тогда пока наша точка будет далеко от минимума, |F′(x)| будет довольно большим, и мы будем двигаться большими шагами, а по мере приближения к точке минимума |F′(x)| будет уменьшаться, и наш шаг тоже будет уменьшаться. Так мы менее вероятно перескочим через точку минимума и подойдем к ней ближе в итоге.

Ответ на второй вопрос «когда останавливать алгоритм поиска минимума?» может быть разный. Действуя таким образом, ровно в точку минимума мы почти никогда не попадем, придется остановиться в какой-то точке возле минимума. Но для применений этого алгоритма нахождение точки около минимума вполне хватает. Чаще всего алгоритм останавливается после прохождения определенного количества шагов и/или достижения определенной близости к точке минимума. Близость, опять же, можно измерять значением производной в точке. Чем ближе, тем модуль производной меньше.

Итак, теперь мы можем сформулировать итоговый алгоритм:

1) Берем некоторую точку x функции F, фиксируем ε (например, ε=0.001).

2) Вычисляем производную F′(x).

3) Изменяем x: x=x−εF′(x).

4) Повторяем 3, пока не пройдет определенное количество шагов и/или мы не станем достаточно близко к точке минимума.

Этот алгоритм называется алгоритм градиентного спуска. "Градиентного" - потому что мы "спускаемся" к точке минимума, вычисляя производную (градиент) функции на каждом шаге.

Теперь, когда мы узнали, что такое градиентный спуск перейдем к его видам. Градиентный спуск бывает пакетным и стохастическим [12]. Формула пакетного градиентного спуска выглядит следующим образом:

$$\Theta_j = \Theta_j + a(y^i - \acute{y}^i)x_j^i \quad (7)$$

Формула стохастического:

$$\Theta_j = \Theta_j + a\sum_{i=0}^{n}(y^i - \acute{y}^i)x_j^i \quad (8)$$

Где a – это шаг метода, y – вывод модели, ý - требуемый результат, Θ_j – параметры модели (матрица весов), i – количество входных экземпляров модели.

Хоть формулы и схожи, но как видно пакетный вычисляет за один шаг градиент от всего набора данных, тогда как стохастический вычисляет для каждого экземпляра отдельно. Этот момент определяет большую разницу между ними. Пакетный спуск и вправду двигается по направлению наискорейшего спуска, используя весь набор в построении целевой функции, а стохастический не может вычислить правильный градиент всей выборки, так как он использует каждый ее элемент по отдельности. То есть если пакетный

спуск пройдет плавно к точке минимума, то стохастический идет не ровно по линии антиградиента, а отклоняясь в случайную сторону на каждом шаге. Однако несмотря на такую особенность стохастический градиентный спуск сходится почти всегда и на больших объемах данных работает намного быстрее. Но главным минусом все-таки является стохастическая природа такого вида спуска, что в результате приводит к ситуациям, где метод не может остановится у минимума и дергается по функции потерь из стороны в сторону.

Из-за этого момента нельзя применять стохастический вариант в «чистом» виде и на практике используют скрещенный вариант – мини-пакетный градиентный спуск (mini-batch). Собственно, в этом методе вместо одного экземпляра выборки мы берем несколько (обычно от 32-128), его мы и будем использовать при построении модели прогнозирования угроз.

Теперь перейдем к тому, что же такое батч-нормализация [13]. Во время обучения нейросетей, даже если мы нормализуем сигнал на входе, пройдя через несколько слоев, он все равно исказиться как по математическому ожиданию, так и по дисперсии, что в итоге способствует серьезным несоответствиям в градиентах на различных слоях системы. Для решения такой проблемы обычно используют ресурсоемкие регуляризаторы, однако они очень сильно замедляют процесс обучения, поэтому в 2015 году, для решения данной проблемы был предложен метод от компании Google, имеющий название батч-нормализация. Его цель – ускорить процесс обучения. Справляется он с этой задачей за счет того, что нормализует входные данные таким образом, что по итогу мы получаем нулевое математическое ожидание и дисперсию на выходе из каждого слоя. Выполняется нормализация перед входом в каждый слой и для каждого батча экземпляров.

Формулы процесса вычисления математического ожидания и дисперсии:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \quad (9)$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad (10)$$

Где B – номер батча (пакета), m – количество экземпляров в пакете, x – конкретный экземпляр.

С помощью этих характеристик мы преобразуем функцию активации следующим образом:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 - \varepsilon}} \quad (11)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (12)$$

Где y – функция активации, ε – параметр, защищающий от деления на ноль, γ и β параметры, которые система должна обучить. Данные параметры заверяют нас, что система не потеряла способности к обобщению и батч-нормализацию полезно применять к входным данным. Помимо ускорения обучения нейросети, данный метод является отличным регуляризатором, помогая не так осмотрительно выбирать гипер-параметры модели и других регуляризаторов, такие как скорость обучения, вероятность дропаута и мощность L2-регуляризатора. Эта особенность является следствием отсутствия детерминированности в наборе данных при прохождении через слои.

2.3.7 Проблема переобучения и методы ее решения

Переобучение (overfitting) — одна из проблем глубоких нейронных сетей, состоящая в том, что модель хорошо распознает только примеры из обучающей выборки, адаптируясь к обучающим примерам, вместо того чтобы

учиться классифицировать примеры, не участвовавшие в обучении (теряя способность к обобщению).

С точки зрения нейросети, проблема выглядит следующим образом: во время обучения методом обратного распространения ошибки, возникает так называемая совместная адаптация во время обновления весов - это когда веса нейрона корректируются с учетом деятельности других нейронов, чтобы минимизировать функцию потерь. По итогу веса нейрона меняются, исправляя при этом ошибки других нейронов и возникают дополнительные “лишние зависимости”, которые приводят к запоминанию обучающего датасета. По итогу происходит запоминание обучающего множества без распознавания скрытых процессов, которые это множество образовали. Графически данное утверждение можно интерпретировать так: на рисунке 12 имеется обучающее множество отмеченное синими точками, желтая линия - график многочлена третьей степени, являющейся близкой к истинной зависимости в данных, фиолетовая - график многочлена 14-й степени, собственно видно, что график многочлена 14-й степени более подходит под обучающее множество, однако имеет сильное расхождение с необходимой зависимостью, что впоследствии, при реализации модели вне обучающего множества, приведет к огромным ошибкам и неточностям.

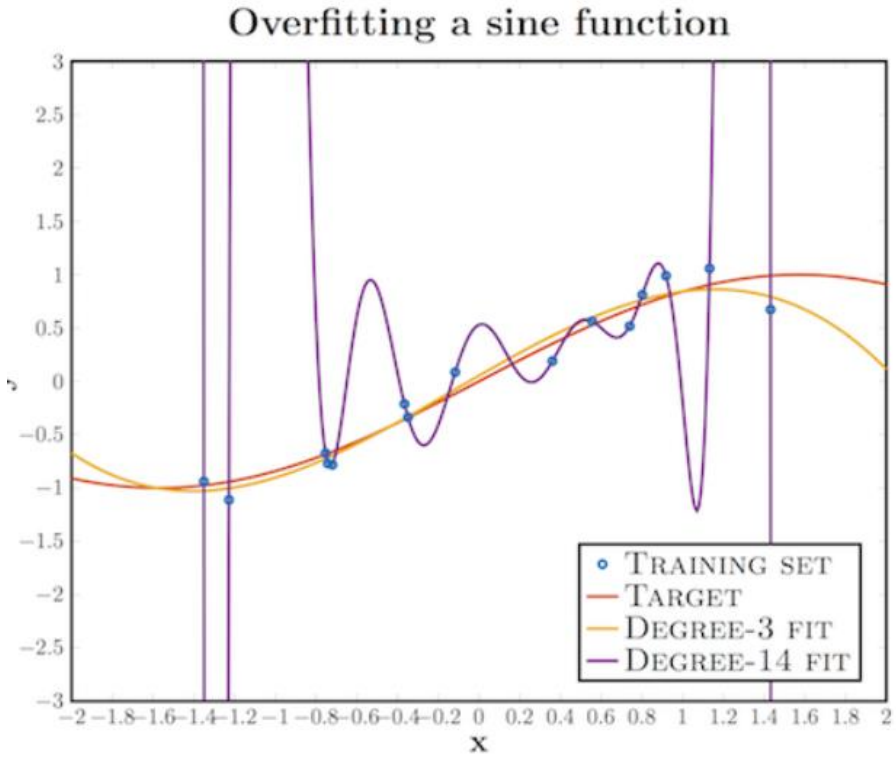

Рисунок 12 – График примера переобучения

Наиболее эффективным решением проблемы переобучения является метод исключения (Dropout) [14].

Сети для обучения получаются с помощью исключения из сети (dropping out) нейронов с вероятностью rate, таким образом, вероятность того, что нейрон останется в сети, составляет 1- rate. “Исключение” нейрона означает, что при любых входных данных или параметрах он возвращает 0.

Еще одним методом борьбы с переобучением является L2-регуляризация. Если в дропауте мы отключали нейроны, то в данном методе уменьшаются веса. L2-регуляризация также является очень популярным методом регуляризации, которая использует более прямолинейный подход, нежели дропаут. Обычно первопричиной проблемы переобучения в общем виде является излишняя сложность модели (большое количество параметров) для конкретного исходного множества значений, вследствии чего модель в погоне за минимизацией затрат теряет способность к обучению. Задачей L2-регуляризации является понизить сложность модели, сохранив те самые параметры в исходном количестве, делается это посредством наложения

штрафов на наибольшие веса в соответствии с параметром “лямбда” - коэффициентов регуляризации, выражающий предпочтение минимизации нормы к минимизации потерь на обучающем множестве. То есть для каждого веса в целевой функции идет умножение на параметр “лямбда” разделенный на 2 (деление на 2 нужно, чтобы при взятии градиента сокращалась двойка в w^2, это упрощает вычисления в методе обратного распространения ошибки).

Подбор параметра “лямбда” очень важен, так как если взять коэффициент слишком малым эффект от регуляризации будет аналогично мал, а если же подобрать слишком большой, то обнулятся все веса в модели. Чтобы применить L2-регуляризацию в Keras, необходимо добавить параметр W_regularizer к каждому слою.

3 Разработка архитектуры системы прогнозирования

3.1 Очистка набора данных

Хоть и набор данных KDD-99 считается очищенным и подходящим для тестирования нейронных архитектур правильно будет все же проверить его на аномалии и целостность. Также вывод дополнительной инфографики может дать некоторое понимание о том, какие признаки важны, а какие может стоит не учитывать в модели. В свою очередь выбор метрики для нейросети зависит от распределения данных в исходной выборке и, учитывая вышеперечисленные факторы, процесс очистки набора данных является необходимым перед построением архитектуры нейросети, так что определим ключевые моменты из которых состоит процесс очистки набора данных. Основная задача очистки наборов данных – это избавиться от пропусков, дубликатов и различных выбросов в данных, при этом не потеряв важную информацию. Большинство алгоритмов машинного обучения вообще не работают с пропусками в данных, что по итогу влияет на некачественный результат модели. Собственно, перейдем к ключевым моментам процесса очистки. Основные этапы, которые проходятся в ходе очистки наборов данных связаны с поиском и устранением:

- Пропусков в данных;
- Выбросов (нетипичных данных);
- Дубликатов (неинформативных данных)
- Несогласованных данных (одних и тех же данных, представленных в разных форматах);

Для работы с данными удобнее всего использовать библиотеку Pandas в силу ее широкого функционала и простого использования, в качестве среды исполнения будем использовать Jupiter Notebook. Первым этапом является создание ноутбука с именем analitycs.ipynb. Далее необходимо импортировать все нужные нам библиотеки (см. рисунок 13).

```
#импорт библиотек
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib as mpl
```

Рисунок 13 – Импортирование библиотек

Как и говорилось, основной библиотекой для использования будет Pandas, также для некоторых математических операций нам понадобится Numpy, для вывода инфографики будем использовать популярные библиотеки Matlotlib и Seaborn. Далее установим стиль для графиков ggplot, фиксированный размер фигур (см. рисунок 14).

```
plt.style.use('ggplot')
matplotlib.rcParams['figure.figsize'] = (12,8)
```

Рисунок 14 – Стиль для графиков

Теперь в соответствии с рисунком 15, мы загружаем набор данных KDD-99, он состоит из двух частей: набора для обучения, который мы загружаем в массив traindata и набора для тестирования, который мы загружаем в массив testdata. Далее необходимо вывести размерность набора данных и тип данных в каждой из колонок, делается это с помощью функций dshape() и dtypes() соответственно. Из выведенной информации видно, что у нас только числовые признаки, значит обрабатывать категориальные смысла нет, а размерность набора данных составляет 494021 строки и 42 столбца (см. рисунок 30). Теперь, чтобы иметь представление о распределенности целевого признака (является угрозой или нет) применим функцию value_counts() к столбцу threat. Угроз в наборе данных KDD-99 на порядок больше, чем простых пользовательских соединений (см. рисунок 16), следовательно, при оценке модели, мы можем воспользоваться стандартной метрикой точности, к тому же, этот факт может говорить о том, что большинство угроз связаны между собой, а значит мы можем добиться определенного успеха при использовании рекуррентной нейросети.

```
#чтение датасета для обучения и тренировки
traindata = pd.read_csv('kddtrain.csv', header=None)
traindata.columns= ["threat", "duration","protocol_type","service","flag","src_bytes","dst_bytes","land"
                        ,"wrong_fragment","urgent","hot","num_failed_logins","logged_in",
                         "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations"
                        ,"num_shells", "num_access_files","num_outbound_cmds","is_host_login","is_guest_login"
                        ,"count", "srv_count","serror_rate", "srv_serror_rate","rerror_rate","srv_rerror_rate"
                        ,"same_srv_rate","diff_srv_rate", "srv_diff_host_rate","dst_host_count","dst_host_srv_count"
                        ,"dst_host_same_srv_rate", "dst_host_diff_srv_rate", "dst_host_same_src_port_rate","dst_host_
                        ,"dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate", "dst_host_srv_rerr
testdata = pd.read_csv('kddtest.csv', header=None)
#вывод размерности и типов данных в датасете
print(traindata.shape)
print(traindata.dtypes)
# вывод распределенности целевого признака
print(traindata['threat'].value_counts())
```

Рисунок 15 – Загрузка набора данных KDD-99

```
(494021, 42)
1    396743
0     97278
Name: threat, dtype: int64
```

Рисунок 16 – Вывод информации о размерности и распределении

Перейдем к очистке данных от пропусков. Существует много способов обнаружения пропусков в данных, для удобства визуализации я предлагаю воспользоваться тепловой картой. Реализуем ее с помощью функции heatmap() из библиотеки Seaborn. Карта, приведенная на рисунке 17, демонстрирует паттерн пропущенных значений в наборе данных. Желтым отмечены пропуски в данных, синим – их отсутствие, на горизонтальной оси располагаются отдельные признаки, на вертикальной количество строк. Как видно из карты, пропуски в данных отсутствуют.

```
B [16]: cols = traindata.columns[:]
        # определяем цвета
        # желтый - пропущенные данные, синий - не пропущенные
        colours = ['#ffff00', '#000099']
        sns.heatmap(traindata[cols].isnull(), cmap=sns.color_palette(colours))
```

```
Out[16]: <AxesSubplot:>
```

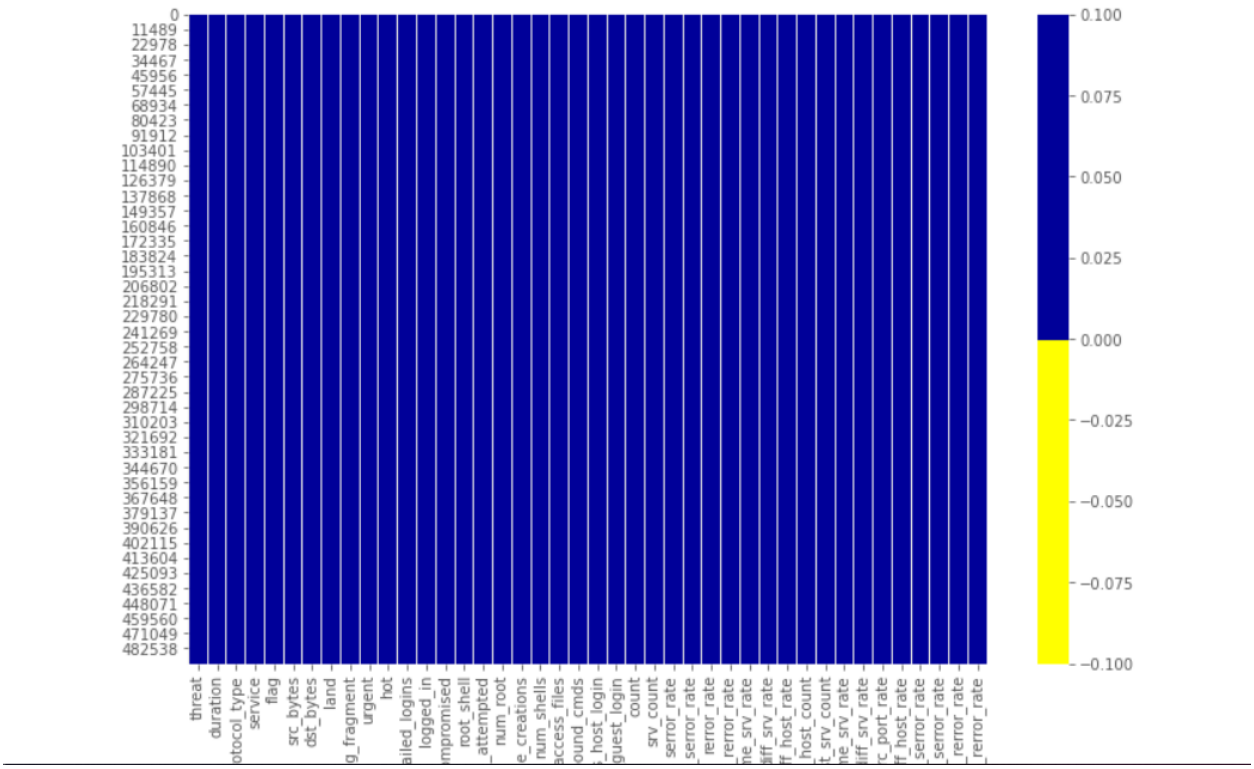

Рисунок 17 – Тепловая карта пропущенных значений

Теперь перейдем к анализу выбросов. Выбросами обычно считают данные, которые существенно отклоняются от других наблюдений и могут являться как ошибкой системы сбора, так и реальным отклонением. Чтобы понять, что с ними делать для начала их следует обнаружить. Для численных и категориальных признаков используют разные методы обнаружения выбросов, однако учитывая, что у нас имеются только числовые признаки воспользуемся описательной статистикой. С помощью данного метода можно увидеть как отличаются максимальные значения от общего распределения и уже для признаков, содержащих выбросы можно визуализировать выборку коробчатой диаграммой. В нашем случае сильные различия между максимальным значением и 75% квартилем были у следующих признаков: num_root, duration_num, num_file_creation. Построим для них коробчатые диаграммы (см. рисунки 18, 19, 20).

```
traindata.boxplot(column=['num_root'])
```

```
<AxesSubplot:>
```

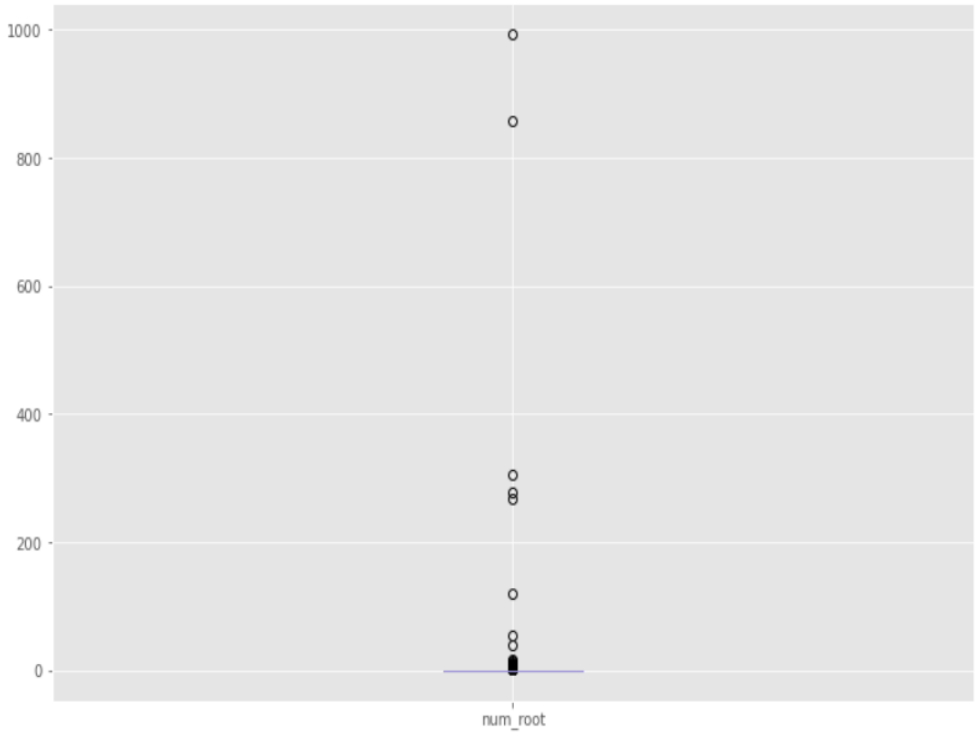

Рисунок 18 – Коробчатая диаграмма признака num_root

```
traindata.boxplot(column=['duration'])
```

```
<AxesSubplot:>
```

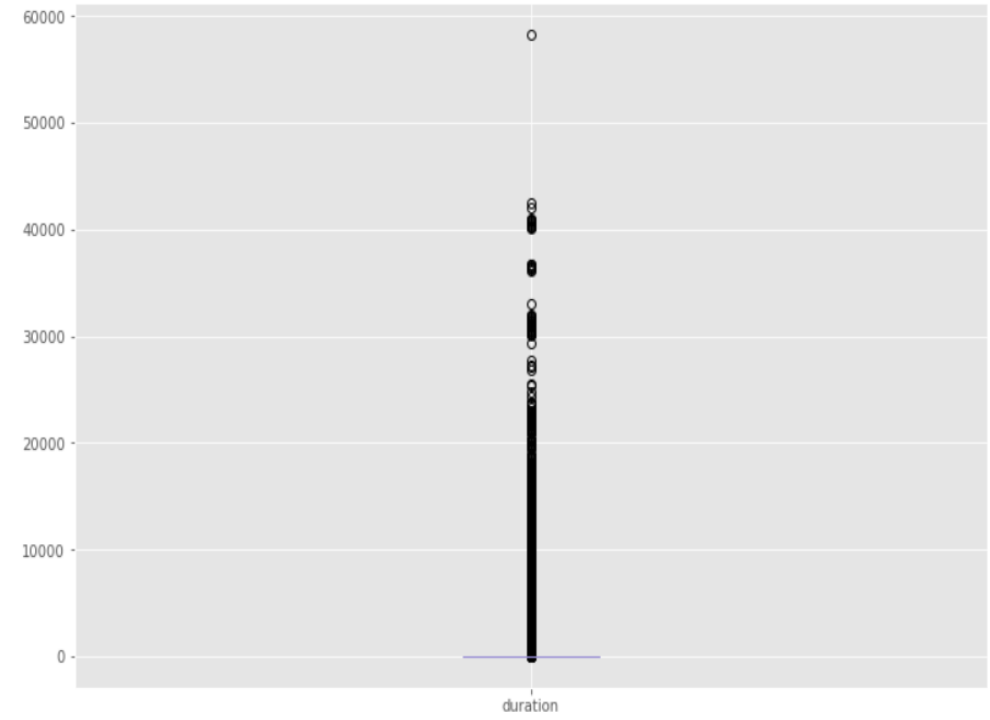

Рисунок 19 – Коробчатая диаграмма признака duration

```
traindata.boxplot(column=['num_file_creations'])
```

```
<AxesSubplot:>
```

Рисунок 20 – Коробчатая диаграмма признака num_file_creations

Распределения значений всех признаков, кроме num_root более-менее равномерно, однако у num_root слишком отдалены 2 верхних значения, они и являются выбросами. От записей с выбросами не всегда следует избавляться, учитывая специфику нашей задачи – поиск и прогнозирование угроз, то такие выбросы могут и являться факторами угрозы, поэтому удалять записи с выбросами не следует.

Заключительной частью очистки набора данных является избавление от неинформативных признаков. Вся информация, поступающая в модель должна нести какую-то информативную нагрузку, если такой ценности данные не могут предоставить – они считаются мусором и их удаляют. Неинформативные признаки бывают разными, разберемся с каждым поэтапно. Первым видом неинформативного признака считается признак, имеющий слишком много одинаковых значений. Переберем признаки с помощью следующего цикла (см. рисунок 21), который выводит список признаков, у которых более 95% строк имеют одинаковые значения.

```
num_rows = len(traindata.index)
low_info_cols = []

for col in traindata.columns:
    cnts = traindata[col].value_counts(dropna=False)
    top_pct = (cnts/num_rows).iloc[0]

    if top_pct > 0.95:
        low_info_cols.append(col)
        print('{0}: {1:.5f}%'.format(col, top_pct*100))
        print(cnts)
        print()
```

```
duration: 97.50011%
0        481671
1          2476
2           870
3           625
5           554
           ...
10136         1
669           1
11160         1
20116         1
5502          1
Name: duration, Length: 2495, dtype: int64
```

Рисунок 21 – Цикл перебора одинаковых значений

По итогу следует убрать признаки num_outbound_cmds и is_host_login, так как у них 100% строк с одинаковыми значениями. Избавляемся от этих колонок с помощью функции drop(). Это не все существующие методы очистки наборов данных, однако для нашей задачи этого достаточно, так как последующая обработка подразумевает удаление аномалий, однако учитывая что мы имеем дело с угрозами, такие аномалии все же могут быть необходимыми данными для обучения модели.

3.2 Определение параметров нейросети

Настройка гипер-параметров необходима для определения оптимального набора числовых коэффициентов, которые регулируют обучение нейросети и соответственно помогают добиться желаемого результата. Начинается построение нейросети обычно с применения уже используемых ранее простых архитектур, которые давали положительные результаты для данной задачи, а также установления параметров со значениями, которые опять же, помогали ранее в различных работах

добиваться неплохой результативности модели. Это помогает добиться определенной производительности, которая может являться неплохой отправной точкой для дальнейшей настройки модели, то есть изменения ее архитектуры и гипер-параметров для извлечения максимальной производительности. Сама по себе настройка гипер-параметров является отдельной областью с большим количеством возможностей для исследований. В данной дипломной работе предлагается поддерживать скорость обучения постоянным на уровне 0.01. Определение оптимального количества нейронов в слое было произведено экспериментальным путем, с помощью изменения их количества от 2 до 1024. После этого количество было увеличено до 1280, но это не дало заметного увеличения точности. Следовательно, количество нейронов было настроено на 1024. Размер батча был выбран 64 экземпляра.

Традиционно, манипулирование с количеством слоев нейросети влияет больше на результат, чем увеличение количества нейронов в слое. Таким образом, были использованы следующие топологии сети для изучения и определения оптимальной структуры сети для наших входных данных: DNN с 1,2,3,4,5 слоями.

Для всех вышеперечисленных топологий сети было запущено 100 эпох. Модель предлагаемой архитектуры DNN для всех случаев использования показана на рис. 1. Она состоит из входного слоя, от 1 до 5 скрытых слоев и выходного слоя. Входной слой состоит из 41 нейрона. Нейроны от входного слоя до скрытого и от скрытого до выходного полностью связаны. Для обучения сети DNN используется метод обратного распространения ошибки. Наконец, лучшая производительность была показана в DNN с тремя слоями. Подробные статистические результаты для различных структур сети представлены в таблице 2.

3.3 Создание классификатора угроз ТКС

3.3.1 Создание модели на основе дерева решений

Код модели был реализован в среде разработки Jupyter Notebook. Jupyter Notebook – невероятно мощный инструмент для интерактивной разработки и представления проектов в области наук о данных. Среда позволяет создавать так называемые «блокноты», которые объединяют код и его вывод в единый документ, который совмещает визуализацию, повествовательный текст, математические уравнения и другие мультимедиа. Этот интуитивно понятный рабочий процесс способствует итеративной и быстрой разработке, что делает ноутбуки все более популярным выбором для представления в данных и их анализа. Для начала рассмотрим процесс создания классификатора угроз ТКС на основе дерева решений.

Первым этапом является создание блокнота с именем dt.ipynb. Сперва необходимо импортировать нужные библиотеки как показано на рисунке 22. Нам понадобится класс классификатора из Sklearn и набор метрик.

```
In [4]: import numpy as np
        import pandas as pd
        from sklearn.preprocessing import Normalizer
        from sklearn.tree import DecisionTreeClassifier
        from sklearn import metrics
```

Рисунок 22 – Импортирование библиотек в файле dt.ipynb

Далее необходимо загрузить в переменные наборы данных KDD для обучения и тестирования, которые упоминались ранее. Делается это также, как и в пункте очистки набора данных. Следующим этапом необходимо с помощью функции iloc() (функция предоставляет итератор) извлечь значения признаков и предсказываемое значение в соответствующие переменные X и Y. Далее, так как нейросети чувствительны к диапазону входных данных, происходит масштабирование значений к диапазону от 0 до 1, реализуется данное действие с помощью функции Normalizer() и transform(). Данный процесс представлен на рисунке 23.

```
X = traindata.iloc[:,1:40]
Y = traindata.iloc[:,0]
C = testdata.iloc[:,0]
T = testdata.iloc[:,1:40]

scaler = Normalizer().fit(X)
trainX = scaler.transform(X)

scaler = Normalizer().fit(T)
testT = scaler.transform(T)


traindata = np.array(trainX)
trainlabel = np.array(Y)

testdata = np.array(testT)
testlabel = np.array(C)
```

Рисунок 23 – Разделение датасета на X и Y

Когда тренировочный датасет и датасет для тестирования готовы, они загружаются в модель DecisionTreeClassifier(), после чего происходит обучение модели (см. рисунок 24), а также сохранение предсказанных результатов в отдельные текстовые файлы. В заключение происходит оценка эффективности модели с помощью таких метрик как точность (precision), аккуратность (accuracy), полнота (recall) и F-мера (f1).

```
In [47]: model = DecisionTreeClassifier()
         model.fit(traindata, trainlabel)
         print(model)

         expected = testlabel
         predicted = model.predict(testdata)
         proba = model.predict_proba(testdata)

         np.savetxt('classical/predictedlabelDT.txt', predicted, fmt='%01d')
         np.savetxt('classical/predictedprobaDT.txt', proba)

         y_train1 = expected
         y_pred = predicted
         accuracy = accuracy_score(y_train1, y_pred)
         recall = recall_score(y_train1, y_pred , average="binary")
         precision = precision_score(y_train1, y_pred , average="binary")
         f1 = f1_score(y_train1, y_pred, average="binary")


         print("-----------------------------------------------")
         print("accuracy")
         print("%.3f" %accuracy)
         print("precision")
         print("%.3f" %precision)
         print("recall")
         print("%.3f" %recall)
         print("f1score")
         print("%.3f" %f1)
```

Рисунок 24 – Обучение дерева решений

Подробные данные о полученных метриках приведены в таблице 2. Полный код модели представлен в приложении А.

3.3.2 Создание модели на основе трехслойной нейросети

Процесс загрузки и предобработки датасета происходит аналогично с деревом решений. После разделения на обучающую и тренировочную выборку, а также выделение предсказываемого признака, настраиваются гипер-параметры модели и устанавливается ее архитектура (см. рисунок 25). Количество входных параметров устанавливаем равным 39 (без учета целевого признака), вероятность дропаута 0.01, функции активации в скрытых слоях – ReLU, параметр лямбда L2-регуляризации был принят на значении 0.0001, так как многие работы, например, как в статье Баррета Зофа и Куока Ле “Neural Architecture Search with Reinforcement Learning” утверждают, что все модели имеют стандарты на параметры оптимизации и касательно L2-регуляризации значение 0.0001 является наиболее оптимальным. Инициализируется модель с помощью функции Sequential(), а далее к ней добавляются слои Dense(). В слоях инициализируются такие параметры, как функция активации, регуляризатор весов, дропаут и дополнительный слой батч-нормализации.

```
l2_lambda = 0.0001
model = Sequential()
model.add(Dense(512,input_dim=39, kernel_regularizer=l2(l2_lambda), activation='relu'))
model.add(Dropout(0.01))
model.add(BatchNormalization())
model.add(Dense(256, kernel_regularizer=l2(l2_lambda),activation='relu'))
model.add(Dropout(0.01))
model.add(BatchNormalization())
model.add(Dense(128, kernel_regularizer=l2(l2_lambda), activation='relu'))
model.add(Dropout(0.01))
model.add(BatchNormalization())
model.add(Dense(1, kernel_regularizer=l2(l2_lambda)))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))
```

Рисунок 25 – Установка архитектуры нейросети

После установки архитектуры можно начать обучение нейросети. Для удобства отслеживания данного процесса рекомендуется установить так

называемый checkpointer, позволяющий выводить статистику обучения на каждой эпохе, а также сохранять ее в файл. Для сохранения результатов в csv файл был использован CSVLogger. Параметры модели были выставлены стандартные для бинарной классификации: функция потерь – бинарная кроссэнтропия, оптимизатор – Adam, метрика – точность. Процесс обучения представлен на рисунке 26. На 43-й эпохе функция потерь перестала улучшаться и обучение можно считать завершенным.

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
checkpointer = callbacks.ModelCheckpoint(filepath="C:/Users/Shigatau/Documents/Диплом/Intrusion-Detection-Systems-m
csv_logger = CSVLogger('C:/Users/Shigatau/Documents/Диплом/Intrusion-Detection-Systems-master/dnn/kddresults/dnn3la
model.fit(X_train, y_train,  batch_size=batch_size, epochs=100, callbacks=[checkpointer,csv_logger])
model.save("C:/Users/Shigatau/Documents/Диплом/dnn/kddresults/dnn3layer/dnn3layer_model.hdf5")
```

```
7720/7720 [==============================] - 43s 6ms/step - loss: 0.0098 - accuracy: 0.9978

Epoch 00041: loss improved from 0.01027 to 0.01024, saving model to C:/Users/Shigatau/Documents/Диплом/Intrusion-De
stems-master/dnn/kddresults/dnn3layer\check.hdf5
Epoch 42/100
7720/7720 [==============================] - 43s 6ms/step - loss: 0.0099 - accuracy: 0.9978

Epoch 00042: loss improved from 0.01024 to 0.01018, saving model to C:/Users/Shigatau/Documents/Диплом/Intrusion-De
stems-master/dnn/kddresults/dnn3layer\check.hdf5
Epoch 43/100
7720/7720 [==============================] - 43s 6ms/step - loss: 0.0099 - accuracy: 0.9979

Epoch 00043: loss did not improve from 0.01018
Epoch 44/100
7720/7720 [==============================] - 45s 6ms/step - loss: 0.0106 - accuracy: 0.9977

Epoch 00044: loss did not improve from 0.01018
Epoch 45/100
7720/7720 [==============================] - 44s 6ms/step - loss: 0.0107 - accuracy: 0.9976
```

Рисунок 26 – Процесс обучения нейросети

Точность на обучающей выборке по итогу обучения достигает 0.9979, что является хорошим результатом. Полный код модели представлен в приложении Б. Теперь сохраненную модель необходимо протестировать, этот процесс представлен в четвертом разделе. После составления архитектуры классификатора можно приступить к созданию полноценной модели прогнозирования, добавив к нейросети LSTM-слои, отвечающие за причинно-следственную связь между НИП.

3.4 Совмещение классификатора с LSTM нейросетью

Первый этапом в построении LSTM нейросети, как и в других моделях является загрузка и обработка набора данных. Процесс схож с предыдущими моделями, однако есть отличие в том, что для LSTM нейросети необходимо преобразовать временной ряд из подключений пользователей в приемлемый для обучения формат. Для этого воспользуемся методом скользящего окна, чтобы сформировать временные промежутки в данных. Для этого, после стандартной процедуры загрузки данных, были сформированы numpy массивы, для удобных преобразований над их размерностью, а все значения приведены к числовому типу float32, так как LSTM сети чувствительны к типу данных. Также был изменен нормализатор на MinMaxScaler, для того, чтобы не изменить бинарные признаки и сохранить их формат значений 0 или 1. LSTM архитектура является крайне гибкой и широко модифицируемой, соответственно подходов к решению задачи прогнозирования может быть крайне много. Варианты варьируются уже на стадии того, как мы будем разделять наш набор данных на значения X и Y. В рамках демонстрации примера предлагается с помощью метода скользящего окна представить данные в следующем формате: каждая запись будет представлена как набор из 40 признаков временного шага t-1 + 1 признак в настоящем времени t, который будет являться бинарным признаком угрозы. Собственно, из этого формата, можно заметить, что в LSTM необходимым гиперпараметром является временной шаг t, называемый timesteps и его инициализация необходима как на стадии решейпинга набора данных, так и на стадии инициализации слоя keras.LSTM. Разберем, что он из себя представляет поподробнее.

Важным гиперпараметром в LSTM является количество временных шагов, которые используются моделью для прогнозирования периода времени. По сути, этот параметр устанавливает рамки для периода времени, как в методе скользящего окна. Для начала, важно понимать, что метка времени в контексте датасета соответствует экземпляру данных с меткой времени. Если обратиться к рисунку 27, то на нем видно, что синий квадрат

обозначает скользящее окно, разделенное на две части с помощью красной линии по центру. Это разделение обуславливает следующие три момента:

- множество точек посередине красной линии обозначает настоящее время t

- множество точек справа от линии обозначает будущее время t + timesteps

- и множество точек слева от линии соответствует прошлому времени t - timesteps

И в каждом таком скользящем окне, если мы предположим, что количество временных шагов равно 10, то LSTM научится на 10 шагах и попытается предсказать следующие 10 шагов в будущем, после чего окно переместится на 10 временных шагов вперед и процесс возобновится.

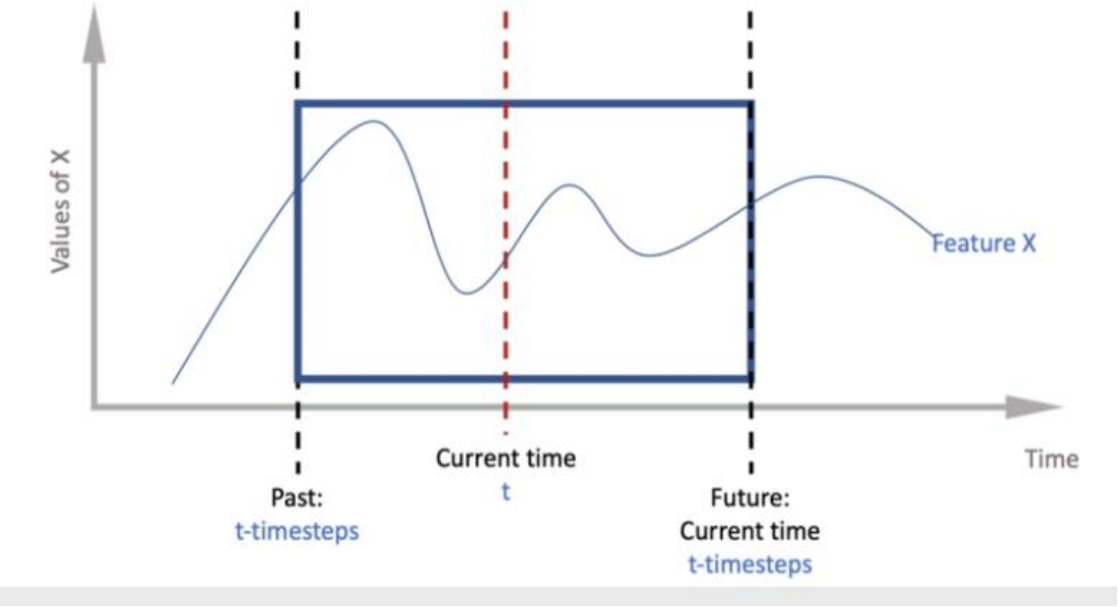

Рисунок 27 – Демонстрация окна времени

Когда запускается окно, другими словами, когда вы перемещаете окно в следующую фазу прогноза, вы перемещаете центр окна вправо на величину, равную одному временному шагу, например, 10. В keras.LSTM параметр timesteps изначально установлен в значение 1. Применив созданную функцию для преобразования набора данных в формат для обучения получился набора

данных из 41-го столбца, пример первых 3 строк приведен на рисунке 28. Функция для преобразования набора данных приведена на рисунке 29.

·3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var8(t-1)	var9(t-1)	var10(t-1)	...	var32(t-1)	var33(t-1)	var34(t-1)	var35(t-1)	var36(t-1)	var37(t-1)	var38(t-1)	var39(t-1)	var40(t-1)	var1(t)
0.0	0.0	0.0	2.610418e-07	0.001057	0.0	0.0	0.0	...	0.035294	1.0	0.0	0.11	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	3.446905e-07	0.000094	0.0	0.0	0.0	...	0.074510	1.0	0.0	0.05	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	3.389216e-07	0.000259	0.0	0.0	0.0	...	0.113725	1.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0	0.0

Рисунок 28 – Набор данных после применения метода скользящего окна

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Рисунок 29 - Функция для преобразования данных в формат для обучения

Далее необходимо выделить значения, по которым мы будем предсказывать, то есть значения X и значения, которые мы будем предсказывать, то есть значения Y. Соответственно набор trainX составят 40 признаков шага t-1, а trainY будет состоять из одного признака угрозы на временном шаге t. Далее с помощью функции reshape() был изменен формат массивов на ожидаемый LSTM слоем 3D формат – [samples, timesteps, features]. Получившиеся размерности массивов trainX, trainY, tesX, textY соответственно равны (494020, 1, 40), (494020, 1), (311028, 1, 40), (311028, 1).

Теперь можно определить архитектуру LSTM и обучить модель. При проектировании LSTM слоя были протестированы архитектуры, содержащие

64, 128 и 256 узлов. Показатели точности на обучающей выборке и последняя эпоха, на которой значение функции потерь больше не улучшалось приведены для сравнения в таблице 1. Слой LSTM проектировался с учетом регуляризаторов, таких как дропаут, L2-регуляризатор и рекуррентный дропаут. Значения были установлены такие же, как и в модели классификатора, аналогично функция потерь осталась без изменений и использованный оптимизатор все также Adam. Слои Dense, отвечающие за модель классификатора были добавлены после LSTM ячеек, так как бинарная классификация является выходным слоем и соответственно слой преобразования признаков для классификации должен быть предпоследним. Батч-нормализация производилась, начиная с первого слоя классификатора.

Таблица 1 – Тестирование LSTM архитектур

Количество узлов	Последняя эпоха обучения	Точность
64	12	0.9909
128	13	0.9990
256	16	0.9934

Как видно из таблицы, наибольшая точность достигается при 128 узлах и обучение проходит крайне быстро, уже на 13-й эпохе, соответственно использование такой архитектуры слоя предпочтительнее. Также важным моментом является подбор размера батча. В построенной модели был использован размер батча, равный 64, однако важно учитывать один момент во время построения LSTM архитектур, заключается он в том, что в Keras внутреннее состояние LSTM сбрасывается в конце каждого батча, соответственно в рамках задачи, где временной шаг определен единицей это ни на что не влияет, однако если взять временной шаг больше чем размер пакета, то внутреннее состояние LSTM ячейки сбросится раньше времени и временная зависимость между экземплярами данных пропадет.

Процесс обучения был запущен совместно с процессом валидации с тренировочными данными, путем помещения в качестве аргумента функции fit() тренировочных наборов данных. Эти действия позволяет сразу сравнить,

как модель работает на тренировочных данных и данных для обучения, и проследить имеет ли место быть «переобучение» модели. После обучения выводится описательная характеристика процесса обучения на каждой эпохе (см. рисунок 30), а также график минимизации функции потерь в ходе обучения и валидации тестовой выборки (см. рисунок 31). Хронология процесса обучения сохраняется в файл check.hdf5, архитектура модели сохраняется в файл LSTM_model.hdf5, результаты на каждой эпохе сохраняются в файл training_set_dnnanalysis.csv.

```
Epoch 96/100
7720/7720 - 30s - loss: 0.0059 - accuracy: 0.9990 - val_loss: 0.3790 - val_accuracy: 0.9452

Epoch 00096: loss did not improve from 0.00586
Epoch 97/100
7720/7720 - 31s - loss: 0.0060 - accuracy: 0.9990 - val_loss: 0.7229 - val_accuracy: 0.8094

Epoch 00097: loss did not improve from 0.00586
Epoch 98/100
7720/7720 - 31s - loss: 0.0060 - accuracy: 0.9990 - val_loss: 0.3723 - val_accuracy: 0.9432

Epoch 00098: loss did not improve from 0.00586
Epoch 99/100
7720/7720 - 30s - loss: 0.0060 - accuracy: 0.9990 - val_loss: 0.3613 - val_accuracy: 0.9452

Epoch 00099: loss did not improve from 0.00586
Epoch 100/100
7720/7720 - 30s - loss: 0.0060 - accuracy: 0.9990 - val_loss: 0.4730 - val_accuracy: 0.9180
```

Рисунок 30 – Последние 4 эпохи обучения

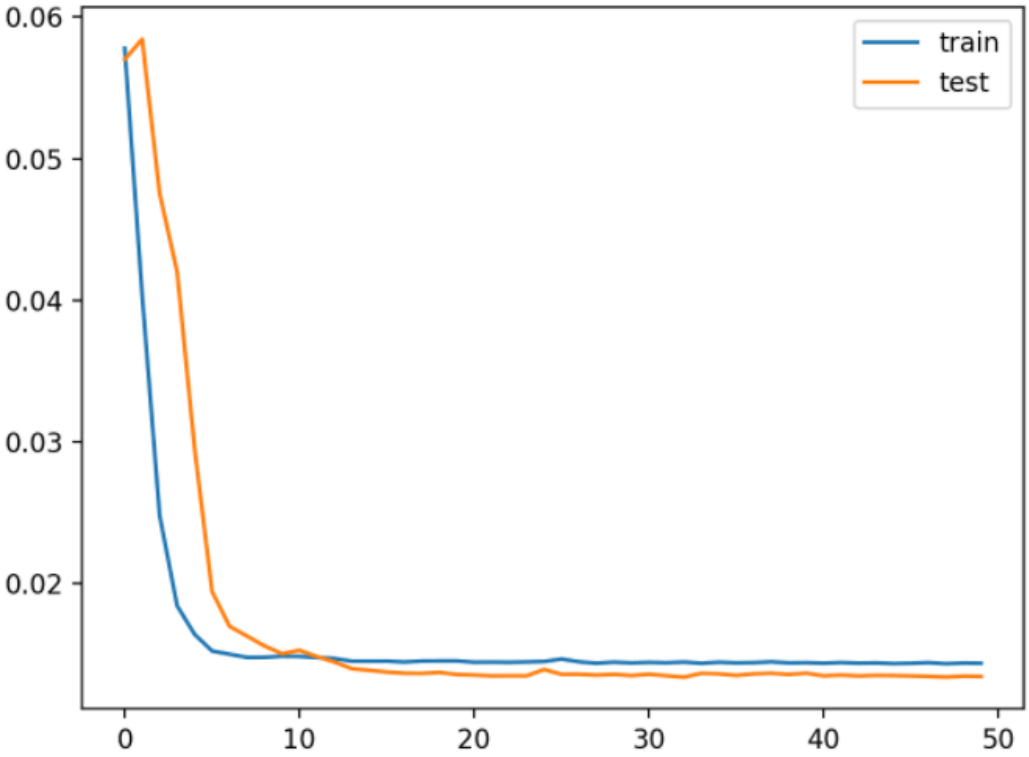

Рисунок 31 – График минимизации функции потерь

Судя по графику минимизации функции потерь, можно увидеть, что функция потерь на тренировочной выборке падает к минимуму быстрее чем функция на тестовой выборке. Это может отражать то, что модель переобучается на тренировочных данных, однако в данном случае разница небольшая, а значит и предпринимать меры не нужно, однако на других данных предрасположенность к переобучению может быть выше и следует учитывать это при использовании предложенного метода прогнозирования. Полный код модели представлен в приложении В.

4 Моделирование процесса прогнозирования угроз на основе сетевого трафика информационной системы с помощью системы прогнозирования

4.1 Прогнозирование угроз с помощью составленной модели

В предыдущем разделе были составлены архитектуры моделей классификатора угроз и прогнозирования угроз, а также обучены. Все модели показывают высокую производительность, поэтому можно приступить к тестированию. Во время тестирования обученные модели функционируют в режиме, в котором они будут введены в эксплуатацию, соответственно классификатор угроз будет выдавать является ли подключение опасным, а модель прогнозирования предсказывает содержит ли следующее подключение угрозы или нет.

Для начала тестирования необходимо загрузить веса и архитектуру модели из ранее обученной нейросети, а также установить необходимые для анализа результатов метрики: точность (precision), аккуратность (accuracy), полнота (recall) и F-мера (f1). После реализации метода model.predict_classes() происходит прогнозирование новых значений моделями. Результаты тестирования классификатора приведены на рисунке 32 и сохранены в файле dnn3predicted.txt.

```
for file in os.listdir("C:/Users/Никита/Documents/Диплом/dnn/kddresults/dnn3layer/"):
  model.load_weights("C:/Users/Никита/Documents/Диплом/dnn/kddresults/dnn3layer/"+file)
  y_train1 = y_test
  y_pred = model.predict_classes(X_test)
  accuracy = accuracy_score(y_train1, y_pred)
  recall = recall_score(y_train1, y_pred , average="binary")
  precision = precision_score(y_train1, y_pred , average="binary")
  f1 = f1_score(y_train1, y_pred, average="binary")
  print("----------------------------------------------")
  print("accuracy")
  print("%.3f" %accuracy)
  print("recall")
  print("%.3f" %recall)
  print("precision")
  print("%.3f" %precision)
  print("f1score")
  print("%.3f" %f1)
  score.append(accuracy)
  name.append(file)
```

```
----------------------------------------------
accuracy
0.929
recall
0.913
precision
0.999
f1score
0.954
```

Рисунок 32 – Тестирование классификатора угроз

Далее можно приступить к прогнозированию новых значений с помощью обученной модели LSTM. Ее процесс тестирования несколько отличается от классификатора. После реализации метода model.predict(), получаем на выходе набор спрогнозированных угроз. Однако перед тем, как приступить к оценке производительности модели с помощью метрик, необходимо вернуть размерность массива testX и исходный формат данных, так как ранее был использован нормализатор значений. Когда все данные были приведены к исходному формату, модель производит подсчет метрик точности (precision), аккуратности (accuracy), полноты (recall) и F-меры (f1). Результаты представлены на рисунке 33.

```
----------------------------------------------
accuracy
0.918
recall
0.972
precision
0.929
f1score
0.950
```

Рисунок 33 – Результаты вычисления метрик итоговой модели

Результат прогнозирования сохраняется в файл LSTMpredicted.txt. Полученные метрики свидетельствуют о высокой точности модели и успешности эксперимента. Как и говорилось ранее, можно протестировать другие способы прогнозирования временных рядов, однако в качестве примера реализации причинно-следственной связи между НИП, такой архитектуры модели достаточно.

4.2 Анализ полученных результатов

Анализ результатов проводится по все тем же метрика: точность (precision), аккуратность (accuracy), полнота (recall) и F-мера (f1). Результаты моделей приведены в таблице 2.

Таблица 2 – Результаты тестируемых моделей

Алгоритм	точность	аккуратность	полнота	F-мера
DNN-1	0,998	0,929	0,915	0,954
DNN-2	0,998	0,929	0,914	0,954
DNN-3	0,999	0,929	0,913	0,954
DNN-4	0,999	0,929	0,913	0,954
DNN-5	0,998	0,927	0,911	0,953
Дерево решений	0,999	0,930	0,914	0,955
LSTM	0,929	0,918	0,972	0,950

При сравнении архитектур DNN, оптимальным является архитектура с 3-мя слотами, так как она имеет самые высокие результаты по метрикам и имеет меньшие вычислительные затраты по сравнению с четырехслойной DNN. По итогу сравнения результатов дерева решений и DNN-3 видно, что для данных наборов данных результаты лучше у дерева решений, а учитывая, что этот алгоритм имеет большую интерпретацию результата, то очевидно дальнейшее использование модели на основе дерева решений. Однако совмещать выход дерева решений и LSTM сеть намного сложнее и требует дополнительных усилий по реализации, в то время как полносвязные слои Dense() легко добавляются к модели LSTM без особой потери в производительности, а при более сложных зависимостях в данных модель

полносвязной нейросети должна выдавать лучшие результаты нежели дерево решений. Соответственно при использовании метода прогнозирования или идентификации угроз экспертом информационной безопасности он должен учитывать, что в ситуациях, где ему необходима интерпретация результатов лучше использовать дерево решений, при соответствующих навыках программирования, а если не хватает производительности модели, либо интерпретация результатов не так важна, то выгоднее и легче использовать модель полносвязной нейросети.

Анализируя результаты LSTM сети видно, что ее метрики меньше, чем у классификаторов, однако это нормальная ситуация и обосновывается она тем, что задача прогнозирования намного сложнее классификации, так как необходимо предполагать какие данные поступят в будущем, а не просто выдвигать решение на основе уже поступивших данных. В рамках своей задачи, результаты LSTM нейросети довольно велики, однако как и говорилось ранее, в рамках демонстрации работы метода прогнозирования была выбрана упрощенная архитектура, предсказывающая на временном шаге равном единице, однако учитывая сложность реальных угроз и их специфику в различных областях, может потребоваться улучшение архитектуры модели или преобразование метода прогнозирования (имеется ввиду на основе каких данных делается прогноз и обучаем ли мы модель синтезировать новые данные или просто следить за динамикой угроз в обучающей выборке). Таким образом был представлен рабочий метод по прогнозированию угроз, который при должном умении способен улучшить качество оценки аудита информационной безопасности и улучшить представление специалиста информационной безопасности о возможных угрозах в системе или предприятии.

4.3 Сравнение работы полученной системы прогнозирования угроз с другими системами

Понятие эффективности системы, модели или процесса является достаточно многогранным, имеющим различные терминологии и способы вычисления. Самым известным методом вычисления эффективности считается совокупность трех показателей: своевременность, обоснованность, ресурсопотребление. Рассмотрим, что подразумевается под каждым из этих понятий.

Под своевременностью понимается способность системы обеспечивать решение задачи – обнаружение угроз в установленный промежуток времени.

Обоснованность означает меру выполнения задачи системы, а именно – долю обнаруженных угроз по сравнению с их реальным наличием в сети, уже приводились метрики точности, аккуратности и полноты, которые достаточно выражают обоснованность. Можно выразить формальное представление данному качеству и другими мерами, однако данные три являются наиболее востребованными и повсеместно используемыми.

Ресурсопотребление характеризует программные и аппаратные средства, необходимые системе обнаружения угроз для решения своей задачи, а также их характеристики. Для лаконичности применения, удобнее вместо последнего иногда использовать противоположный показатель – ресурсоемкость, определяя таким образом общее повышение эффективности как повышение всех или ряда ее показателей. Особенность такого определения заключается в том, что повышение одного из параметров (без дополнительных мер) неизменно ведет к понижению другого. Так, обнаружение большего числа угроз (повышение обоснованности) может быть получено за счет снижения скорости обнаружения (снижение своевременности) и/или затрачивания большего количества ресурсов (повышение ресурсопотребления). Следовательно, повышение любого из параметров при условии сохранения остальных будет однозначно означать общее повышение эффективности системы.

Для сравнения показателя обоснованности разработанной системы было произведено сравнение с альтернативной системой по обнаружению угроз PacketFense.

Данный продукт является Open Source проектом, а его решение в области контроля доступа в сеть является востребованным в сфере информационной безопасности. Основное назначение данного продукта заключается в анализе сетевого трафика на наличие угроз информационной безопасности, в результате обрабатывая трафик на выявление различных аномалий в нем.

Основными возможностями продукта являются следующие:

- обнаружение угроз в режиме реального времени;
- реагирование на инциденты и расследования;
- сегментацию сети;
- производительность сети и планирование пропускной способности;
- возможности для обеспечения соответствия регулятивным требованиям.

PacketFense основан на алгоритмах машинного обучения, что в свою очередь делает его ближайшим аналогом моделей, представленных в данной дипломной работе. Однако данная система рассматривает идентификацию угроз в настоящем времени и в ней отсутствует элемент прогнозирования, так как продукт основан на классической системе возникновения угроз. Тем не менее, модель классификатора, спроектированного в третьем разделе, может быть использована для идентификации угроз и решает ту же задачу, что и PacketFense, следовательно, можно сравнить эффективность предложенной модели с уже существующим продуктом. Однако следует учесть, что такие программы контроля сетей, как PacketFense развертываются на всем предприятии и их сложно использовать для мелких и одноразовых задач, как например анализ набора данных на аномалии. Чтобы реализовать данную задачу необходимо внедрять целую систему контроля на всю сеть предприятия, что затрачивает слишком много времени и ресурсов и требует

генерации подключений пользователей, аналогичных набору данных. В то время как метод, предложенный в рамках дипломной работы является инструментом для решения подобных задач, что в некоторых ситуациях может быть необходимо при составлении аудита информационной безопасности. Сетевая архитектура PacketFense приведена на рисунке 34.

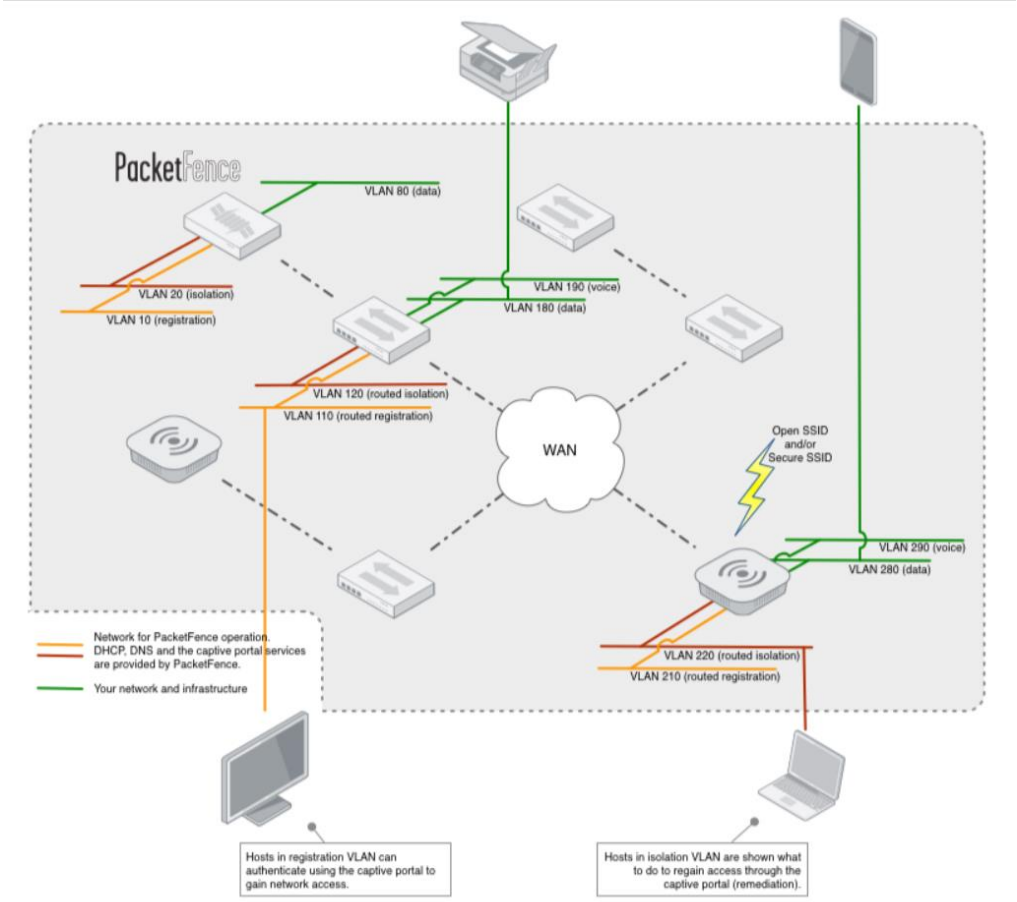

Рисунок 34 – Сетевая архитектура PacketFense

Для сравнения обоснованности разработанной системы и ее ближайшего аналога необходимо вычислить и сравнить введенные меры качества для каждой из систем. Для этого программа PacketFense была запущена на том же наборе данных KDD для анализа на угрозы. Результаты приведены в таблице 3.

Таблица 3 – Результаты сравнения обоснованности систем

Название системы	точность	аккуратность	полнота	F-мера
DNN-3	0,999	0,929	0,913	0,954
PacketFense	0,989	0,980	0,920	0,910

Сравнительный анализ для разработанной системы и PacketFense позволяет утверждать, что первая обладает большими показателями точности и f-меры, но разница в показателях не особо велика, а значит обоснованными можно считать обе системы, но как говорилось ранее, DNN-3 удобнее использовать при исследованиях, постановке экспериментов над данными и решении узконаправленных задач прогнозирования угроз.

Далее проведем оценку параметра своевременности системы. Для этого необходимо вычислить и сравнить время обнаружения угроз разработанной системы и PacketFense. Для этого сравним время выдачи результата системы после обработки тестового набора данных. Среднее время идентификации угроз за 5 попыток у разрабатываемой системы составляет 19 секунд, а у PacketFense 24 секунды. Опять же, данное различие не значительно, поэтому перейдем к анализу ресурсопотребления.

Оценивая ресурсопотребление, было принято решение сравнить занимаемое пространство программой на жестком диске, средняя нагрузка на CPU и память в режиме прогнозирования угроз.

Таблица 4 – Результаты сравнения ресурсопотребления систем

Название системы	Объем занимаемого пространства	Средняя нагрузка на CPU, %	Средняя загрузка памяти, %
DNN-3	200 КБ	40	30
PacketFense	100 ГБ	45	40

Из таблицы видно, что памяти разрабатываемая система требует намного меньше, так как PacketFense является крупной развертываемой системой и общая нагрузка на компьютеры сети при различных имплементациях продукта может отличаться от приведенных показателей в таблице. По итогу анализа эффективности разрабатываемой системы можно сделать вывод о том, что она конкурентно способна аналогичным системам и является более выгодным для использования инструментом в руках специалиста информационной безопасности при решении узконаправленных и не ресурсозатратных задач.

4.4 Предложения по применению системы прогнозирования угроз

4.4.1 Предложения по обеспечению функционирования

Перед тем, как дать некоторые рекомендации по использованию спроектированной модели и метода прогнозирования угроз, опишем функциональную схему проведения экспериментов и реализации метода специалистом. Функциональная схема приведена на рисунке 35.

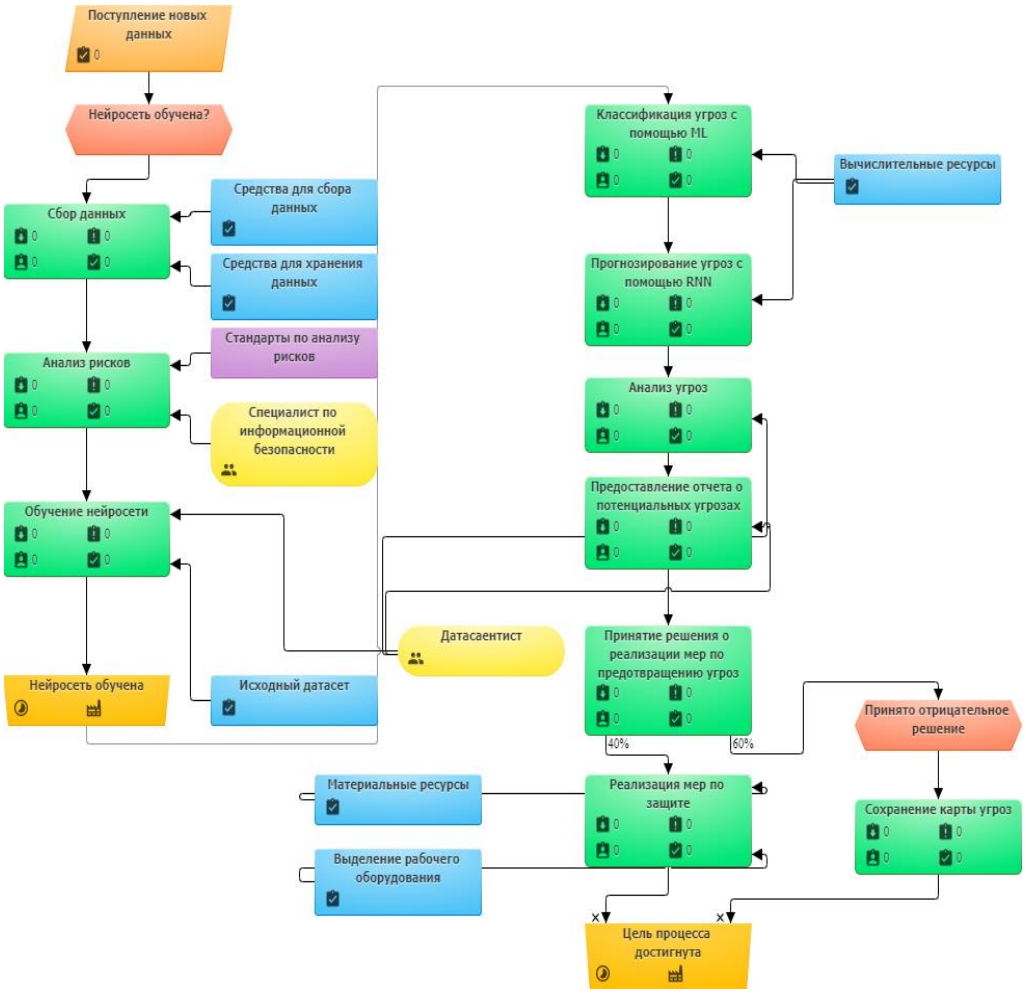

Рисунок 35 – Функциональная схема работы модели прогнозирования угроз

Из данной схемы видно, как происходит ввод в эксплуатацию системы «Нейросеть для прогнозирования угроз ТКС» на предприятие. Помимо описания подсистем и промежуточных этапов между ними, на схеме видны дополнительные ресурсы (синие блоки), которые необходимы для функционирования системы, а также специалисты (желтые блоки), которые должны будут поддерживать систему. На последних стоит остановиться поподробнее.

Датасаентист – это специалист в области машинного обучения и анализа данных. Он необходим в системе, так как именно он будет оценивать работу нейросети и подбирать для нее гипер-параметры и рабочие алгоритмы, подходящие под конкретный набор данных предприятия. Из схемы видно, что он вовлечен непосредственно в настройку подсистемы «обучение нейросети», а также он ответственен за поддержку работоспособности системы и корректности ее прогнозирования, во избежание проблем с «переобучением». Поэтому он относится также к блокам «анализ угроз» и «предоставление отчета о потенциальных угрозах».

Специалист по информационной безопасности необходим для разметки тестового набора данных, на котором будет обучаться нейросеть. Это очень важный этап, так как нейросеть при обучении будет считать за идеальным результат именно из тестового набора данных, поэтому разметка набора данных должна быть без различных аномалий, данные должны быть последовательны в их поступлении в систему, а также должна поддерживаться целостность этих данных.

Большинство заказов поступает от предприятий с высоким информационным оборотом, а также от кампаний, в которых защищаемая информация циркулирует среди большого множества работников. В настоящее время технологическая подготовка производства происходит следующим образом:

– Заключается договор и формируется заказ;

– Специалист по информационной безопасности проводит прямую экспертную оценку предприятия на наличие угроз, то есть эксперт определяет перечни параметров, характеризующих угрозы информационной безопасности, и дает субъективные коэффициенты важности каждого параметра;

– Вручную проводится статистический анализ предприятия, то есть предприятие предоставляет информацию об инцидентах информационной безопасности, а специалист сопоставляет такие параметры как частота

возникновения угроз определенного типа, их источники и причины успеха или неудачи;

– Осуществляется проверка введенных мер для списка типовых угроз и уязвимостей, и взаимосвязей между угрозами и механизмами контроля из ISO 27002;

–Производится оценка вероятности угроз и составление карты возможных угроз;

– Принятие решения о реализации мер по предотвращению угроз, выделение необходимых материальных ресурсов и оборудования;

Для диагностирования системы, контроля правильности функционирования технических и программных средств системы должен проводиться функциональный контроль этих средств по установленному регламенту, который включает проверку:

– готовности технических средств;

– функциональной готовности системы.

Система должна разрабатываться, как система открытого типа, что решает следующие задачи:

– обеспечение переобучения системы на новом датасете;

– обеспечение интегрирования системы под различные виды угроз;

– обеспечение переносимости программного обеспечения;

– функциональной интеграции задач, решаемых ранее раздельно.

4.4.2 Предложения по обеспечению надежности

Безопасность хранения данных системы «Нейросеть для прогнозирования угроз в ТКС» при аппаратных сбоях должна обеспечиваться соответствующими средствами операционной системы, а также дополнительными техническими средствами по обеспечению информационной безопасности, интегрированными на предприятии.

Сохранность информационных баз данных при отключении технического оборудования должна обеспечиваться за счет регулярного копирования и архивации данных на дополнительные носители информации.

4.4.3 Предложения по эксплуатации

Условия и режим эксплуатации системы определяется составом технических средств, используемых системой. Эксперт реализующий систему прогнозирования угроз должен брать во внимание характерные для конкретного предприятия аномалии в данных и в зависимости от этого изменять подход к прогнозированию и оценке системы. Также необходим определенный уровень компетенции специалиста в области DataScience, ответственного за обучение нейросети. При интеграции системы «Нейросеть для прогнозирования угроз в ТКС» с другими системами прогнозирования и идентификации угроз нужно учитывать, каким образом они выполняют поставленную перед ними задачу, так как системы могут быть несовместимы.

4.4.4 Предложения по информационному обеспечению системы

Информационное обеспечение представляет собой совокупность необходимых для функционирования системы «Нейросеть для прогнозирования угроз в ТКС» данных и документов. Данные должны быть организованы в соответствующие таблицы базы данных, обеспечивающие оптимальный доступ к требуемой информации при реализации алгоритмов (классификаторов, рекуррентной нейросети).

В состав информационного обеспечения входят: нормативно-справочная информация, входные данные, выходные данные, система ведения отчетов, система хранения промежуточных состояний системы во время обучения и тестирования.

Программное обеспечение системы «Нейросеть для прогнозирования угроз в ТКС» состоит из:

– общесистемного ПО;

– функционального ПО.

Общесистемное ПО обеспечивает работу функционального ПО и его сетевое взаимодействие.

В состав общесистемного ПО входят:

– операционные системы (ОС);

– системы управления базами данных, включая средства импорта-экспорта и преобразования данных;

– системы, обеспечивающие форматированное и наглядное представление данных для анализа и создания отчетных печатных форм (системы представления данных).

В качестве ОС должны быть использованы совместимые высокопроизводительные сетевые ОС:

– MS Windows;

Использование других средств для импорта-экспорта и преобразования данных допускается, но не рекомендуется.

Разработка функционального ПО должна производиться на:

– языке программирования Python;

– библиотеке Pandas для удобного представление и преобразования датасета.

– библиотеке машинного обучения Scikit-learn;

- библиотеке для разработки архитектуры и обучения нейросетей TensorFlow;

4.4.5 Порядок контроля и приемки системы

Ввод в действие системы «Нейросеть для прогнозирования угроз в ТКС» должен осуществляться в соответствии с общими требованиями по внедрению систем машинного обучения в предприятия.

Приемка работ по очередям и стадиям создания, порядок согласования и утверждения приемочных документов должен осуществляться в соответствии с договорами между Заказчиком и Разработчиком.

В ходе опытной эксплуатации должен вестись рабочий журнал регистрации сведений о качестве функционирования системы. По результатам опытной эксплуатации составляется соответствующий акт завершения работ.

По завершении комплексной отладки и опытной эксплуатации системы в целом система «Нейросеть для прогнозирования угроз в ТКС» должна быть сдана в промышленную эксплуатацию.

Приемка системы «Нейросеть для прогнозирования угроз в ТКС» в промышленную эксплуатацию заключается в выполнении следующих работ:

- проверки соответствия выполненных работ требованиям технического задания;
- проверки работоспособности системы на реальных данных;
- проверки подготовленности пользователей к работе с системой;
- выработки рекомендаций по дальнейшему развитию системы.

Для приемки системы «Нейросеть для прогнозирования угроз в ТКС» в промышленную эксплуатацию должна быть создана приемочная комиссия, в которую входят представители Заказчика и Разработчика.

4.5 Требования к сбору данных для применения системы прогнозирования угроз

Для организации эффективной работы системы на этапе ее обучения и эксплуатации необходимо правильно организовать процесс сбора данных. В соответствии с этим предлагаются следующие требования к подсистеме сбора данных:

- Данные должны быть строго определенного формата, в соответствии с шаблоном, принимаемым на вход нейросети;
- Во избежание аномалий в данных, необходимо осуществлять сохранение полученной информации в той последовательности, в которой она возникла на предприятии;
- Система хранения данных должна основываться на БД семейства SQL;

- Средства для хранения данных должны быть готовы к большому количеству данных.

- Поступающая выборка должна быть разделена на тестовую и обучающую, в соответствии с методом кроссвалидации;

- В системе должен присутствовать достаточно объемный датасет (1000-5000 записей);

- В системе должны быть данные, имеющие причинно-следственную связь, т. е. угрозы, которые возникают в результате реализации других угроз;

ЗАКЛЮЧЕНИЕ

Подводя итог проделанной работы, отметим проводимые исследования и их результаты. В ходе дипломной работы было проведено исследование такой области, как прогнозирование и оценка возникновения угроз в телекоммуникационных системах, приведены различные модели прогнозирования угроз и отмечены их особенности и недостатки. Была составлена модель прогнозирования угроз, учитывающая временные зависимости и взаимосвязь между угрозами, что отличает ее от существующих моделей анализа возникновения угроз, также была предложена реализация данной модели с помощью алгоритмов машинного обучения.

Были описаны задачи по реализации нейросети на основе модели прогнозирования угроз, а также описаны ML-алгоритмы классификации и регрессионного анализа. Для реализации причинно-следственной связи была описана архитектура рекуррентной нейросети и ее особенности, а также алгоритмы работы необходимых для построения нейросети оптимизаторов и регуляризаторов. Была описана проблема переобучения, свойственная большинству нейросетей, и приведены методы ее решения, например, дропаут и l2-регуляризация.

Также был подобран необходимый инструментарий для реализации проекта, принято решение использовать язык программирования Python с использованием Pandas, Skikit-learn, Keras, Seaborn.

Для приведения примера реализации модели была выбрана ее имплементация с системой IDS на основе данных подключений пользователей в телекоммуникационной инфраструктуре. Был произведен анализ работы системы с различными алгоритмами и настройками, например, проводилось тестирование различного количества слоев и нейронов в них, а также тестирование алгоритма прогнозирования в LSTM сети, в результате чего была выбрана наиболее оптимальная архитектура системы. Разработанная модель показала неплохие результаты на датасете KDD-99, точность обучения

составила 0,999. Для анализа результатов на тестовой выборке были введены следующие метрики: точность, аккуратность, полнота, f-мера. Соответственно результаты модели составили: точность – 0,929, аккуратность – 0,918, полнота, 0,972, f-мера – 0,950, что говорит о возможности применения системы при решении реальных проблем информационной безопасности. Также была проанализирована производительность модели и составлено сравнение с ближайшим аналогом – системой PacketFense, по итогу которого было выявлено, что разработанная система не уступает по показателям точности и превосходит аналог в показателе ресурсопотребления. Сформулированы рекомендации по применению результатов работы в исследованиях и анализе угроз информационной безопасности.

Исходя из эмпирических результатов этой работы, мы можем утверждать, что методы глубокого обучения являются перспективным направлением в решении задач кибербезопасности, однако, хотя эффективность работы системы на искусственном наборе данных является неплохой, в результате применения модели в сети с реальным трафиком, который содержит более сложные и последние типы атак, могут возникнуть проблемы. Это и является областью для дальнейшего совершенствования модели прогнозирования угроз, как со стороны автоматизации подбора алгоритма классификации, так и со стороны идентификации большего количества сложных угроз.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Чечулин А. А. и Котенко И. В. Построение графов атак для анализа событий безопасности // журнал «Безопасность информационных технологий» Санкт-Петербургский институт информатики и автоматизации: 2014, с.18-23.

2 Переобучение [Электронный ресурс] // Режим доступа: https://wiki.loginom.ru/articles/overtraining.html

3 Джон Мэйн The Capabilities, Opportunities and Motivation Behaviour-Based Theory of Change Model // статья из научного архива ResearchGate 4 июня 2016 года.

4 С. Синклер, Л. Пирс и С. Мацнер «An application of machine learning to network intrusion detection» // IEEE Computer Society, 1999.

5 Х. Дебар и Б. Дорицци. «An application of a recurrent network to an intrusion detection system» // В Международной совместной конференции по нейронным сетям, 1992. IJCNN., vol. 2, с. 478-483 т.2. июнь 1992.

6 М. Таваллаи, Э. Багери, В. Лу и А. А. Горбани. Подробный анализ набора данных KDD CUP 99 // Cisda, стр. 16. IEEE, Июль 2009 г.

7 Редько, В.Г. Эволюция, нейронные сети, интеллект: Модели и концепции эволюционной кибернетики // В.Г. Редько. - М.: Ленанд, 2015. - 224 с.

8 Галушкин, А.И. Нейронные сети: основы теории. // А.И. Галушкин. - М.: РиС, 2014. - 496 с.

9 Рекуррентные нейросети [Электронный ресурс] // Режим доступа: https://neurohive.io/ru/osnovy-data-science/rekurrentnye-nejronnye-seti/

10 Гафаров Ф.М., Галимянов А.Ф. Искусственные нейронные сети и их приложения: Учебное пособие // издательство казанского университета: 2018, с. 72-90.

11 Функции активации [Электронный ресурс] // Режим доступа: http://datareview.info/article/eto-nuzhno-znat-klyuchevyie-rekomendatsii-po-glubokomu-obucheniyu-chast-2/

12 Градиентный спуск [Электронный ресурс] // Режим доступа: https://habr.com/ru/post/308604/

13 Batch normalization [Электронный ресурс] // Режим доступа: https://habr.com/ru/post/277345/

14 Дропаут [Электронный ресурс] // Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Dropout

15 Хенрик Бринк, Джозеф Ричардс, Марк Феверолф «Машинное обучение» // Питер: 2017.

16 Баррет Зоф и Куок Ле Neural Architecture Search with Reinforcement Learning // статья с конференции Google Brain 2017

17 Дерево решений [Электронный ресурс] // Режим доступа: https://loginom.ru/blog/decision-tree-p1

18 Израилов К. Е. Модель прогнозирования угроз телекоммуникационной системы на базе искусственной нейронной сети // «Вестник Инжэкона выпуск 8» Санкт-Петербургский государственный инженерно-экономический университет: 2012, с.150-153.

19 Галушкин, А.И. Нейронные сети: основы теории. // А.И. Галушкин. - М.: РиС, 2014. - 496 с.

20 Крошилин С. В. Возможные угрозы безопасности экономических информационных систем и методы их устранения // Проблемы и методы управления экономической безопасностью регионов: Материалы межвузовской научной конференции профессорско-преподавательского состава. - Коломна: КГПИ, 2006. - С. 240-244.

ПРИЛОЖЕНИЕ А

```
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import Normalizer
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
traindata = pd.read_csv('kddtrain.csv', header=None)

traindata.columns= ["threat", "duration","protocol_type","service","flag","src_bytes","dst_bytes","land"
                    ,"wrong_fragment","urgent","hot","num_failed_logins","logged_in",
                     "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations"
                    ,"num_shells",
"num_access_files","num_outbound_cmds","is_host_login","is_guest_login"
                    ,"count", "srv_count","serror_rate", "srv_serror_rate","rerror_rate","srv_rerror_rate"
                    ,"same_srv_rate","diff_srv_rate",
"srv_diff_host_rate","dst_host_count","dst_host_srv_count"
                    ,"dst_host_same_srv_rate", "dst_host_diff_srv_rate",
"dst_host_same_src_port_rate","dst_host_srv_diff_host_rate"
                    ,"dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate",
"dst_host_srv_rerror_rate"]
testdata = pd.read_csv('kddtest.csv', header=None)
traindata[0:10]
X = traindata.iloc[:,1:42]
Y = traindata.iloc[:,0]
C = testdata.iloc[:,0]
T = testdata.iloc[:,1:42]

scaler = Normalizer().fit(X)
trainX = scaler.transform(X)

scaler = Normalizer().fit(T)
testT = scaler.transform(T)



traindata = np.array(trainX)
```

```
trainlabel = np.array(Y)

testdata = np.array(testT)
testlabel = np.array(C)
model = DecisionTreeClassifier()
model.fit(traindata, trainlabel)
print(model)

expected = testlabel
predicted = model.predict(testdata)
proba = model.predict_proba(testdata)

np.savetxt('classical/predictedlabelDT.txt', predicted, fmt='%01d')
np.savetxt('classical/predictedprobaDT.txt', proba)

y_train1 = expected
y_pred = predicted
accuracy = accuracy_score(y_train1, y_pred)
recall = recall_score(y_train1, y_pred , average="binary")
precision = precision_score(y_train1, y_pred , average="binary")
f1 = f1_score(y_train1, y_pred, average="binary")



print("-----------------------------------------------")
print("accuracy")
print("%.3f" %accuracy)
print("precision")
print("%.3f" %precision)
print("recall")
print("%.3f" %recall)
print("f1score")
print("%.3f" %f1)
model = AdaBoostClassifier(n_estimators=100)
model.fit(traindata, trainlabel)

# make predictions
expected = testlabel
predicted = model.predict(testdata)
proba = model.predict_proba(testdata)

np.savetxt('classical/predictedlabelAB.txt', predicted, fmt='%01d')
np.savetxt('classical/predictedprobaAB.txt', proba)
```

```
# summarize the fit of the model


y_train1 = expected
y_pred = predicted
accuracy = accuracy_score(y_train1, y_pred)
recall = recall_score(y_train1, y_pred , average="binary")
precision = precision_score(y_train1, y_pred , average="binary")
f1 = f1_score(y_train1, y_pred, average="binary")
print("---------------------------------------------")
print("accuracy")
print("%.3f" %accuracy)
print("precision")
print("%.3f" %precision)
print("racall")
print("%.3f" %recall)
print("f1score")
print("%.3f" %f1)
```

ПРИЛОЖЕНИЕ Б

```
from __future__ import print_function
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
np.random.seed(1337)
from keras.preprocessing import sequence
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding
from keras.layers import LSTM, SimpleRNN, GRU
from keras.datasets import imdb
from keras.utils.np_utils import to_categorical
from sklearn import metrics
from metrics import accuracy_score
from sklearn.preprocessing import Normalizer
import h5py
from keras import callbacks
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
traindata = pd.read_csv('kddtrain.csv', header=None)

traindata.columns= ["threat", "duration","protocol_type","service","flag","src_bytes","dst_bytes","land"
                    ,"wrong_fragment","urgent","hot","num_failed_logins","logged_in",
                     "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations"
                    ,"num_shells",
"num_access_files","num_outbound_cmds","is_host_login","is_guest_login"
                    ,"count", "srv_count","serror_rate", "srv_serror_rate","rerror_rate","srv_rerror_rate"
                    ,"same_srv_rate","diff_srv_rate",
"srv_diff_host_rate","dst_host_count","dst_host_srv_count"
                    ,"dst_host_same_srv_rate", "dst_host_diff_srv_rate",
"dst_host_same_src_port_rate","dst_host_srv_diff_host_rate"
                    ,"dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate",
"dst_host_srv_rerror_rate"]
testdata = pd.read_csv('kddtest.csv', header=None)
traindata[0:10]
X = traindata.iloc[:,1:42]
Y = traindata.iloc[:,0]
C = testdata.iloc[:,0]
T = testdata.iloc[:,1:42]
```

```
scaler = Normalizer().fit(X)
trainX = scaler.transform(X)
scaler = Normalizer().fit(T)
testT = scaler.transform(T)
y_train = np.array(Y)
y_test = np.array(C)
X_train = np.array(trainX)
X_test = np.array(testT)
batch_size = 64
model = Sequential()
model.add(Dense(1024,input_dim=41,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(768,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.01))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
checkpointer = callbacks.ModelCheckpoint(filepath="C:/Users/Никита/Documents/Диплом
/dnn/kddresults/dnn3layer/check.hdf5", verbose=1, save_best_only=True, monitor='loss')
csv_logger = CSVLogger('C:/Users/Никита/Documents/Диплом
/dnn/kddresults/dnn3layer/training_set_dnnanalysis.csv',separator=',', append=False)
model.fit(X_train, y_train,  batch_size=batch_size, nb_epoch=100, callbacks=[checkpointer,csv_logger])
model.save("C:/Users/Никита/Documents/Диплом/dnn/kddresults/dnn3layer/dnn3layer_model.hdf5")
score = []
name = []
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
import os
for file in os.listdir("C:/Users/Никита/Documents/Диплом/dnn/kddresults/dnn3layer/"):
  model.load_weights("C:/Users/Никита/Documents/Диплом/dnn/kddresults/dnn3layer/"+file)
  y_train1 = y_test
  y_pred = model.predict_classes(X_test)
  accuracy = accuracy_score(y_train1, y_pred)
  recall = recall_score(y_train1, y_pred , average="binary")
  precision = precision_score(y_train1, y_pred , average="binary")
  f1 = f1_score(y_train1, y_pred, average="binary")
  print("---------------------------------------------")
```

```
            print("accuracy")
            print("%.3f" %accuracy)
            print("recall")
            print("%.3f" %recall)
            print("precision")
            print("%.3f" %precision)
            print("f1score")
            print("%.3f" %f1)
            score.append(accuracy)
            name.append(file)
        model.load_weights("C:/Users/Никита/Documents/Диплом
/dnn/kddresults/dnn3layer/"+name[score.index(max(score))])
        pred = model.predict_classes(X_test)
        proba = model.predict_proba(X_test)
        np.savetxt("dnnres/dnn3predicted.txt", pred)
        np.savetxt("dnnres/dnn3probability.txt", proba)

        accuracy = accuracy_score(y_test, pred)
        recall = recall_score(y_test, pred , average="binary")
        precision = precision_score(y_test, pred , average="binary")
        f1 = f1_score(y_test, pred, average="binary")
```

ПРИЛОЖЕНИЕ В

```
from __future__ import print_function
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1337)
from keras.preprocessing import sequence
from keras.regularizers import l2
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, BatchNormalization
from keras.layers import LSTM, SimpleRNN, GRU
from keras.datasets import imdb
from keras.utils.np_utils import to_categorical
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import Normalizer, MinMaxScaler
import h5py
from keras import callbacks
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
traindata = pd.read_csv('kddtrain.csv', header=None)
traindata.columns= ["threat", "duration","protocol_type","service","flag","src_bytes","dst_bytes","land"
                    ,"wrong_fragment","urgent","hot","num_failed_logins","logged_in",
                     "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations"
                    ,"num_shells",
"num_access_files","num_outbound_cmds","is_host_login","is_guest_login"
                    ,"count", "srv_count","serror_rate", "srv_serror_rate","rerror_rate","srv_rerror_rate"
                    ,"same_srv_rate","diff_srv_rate",
"srv_diff_host_rate","dst_host_count","dst_host_srv_count"
                    ,"dst_host_same_srv_rate", "dst_host_diff_srv_rate",
"dst_host_same_src_port_rate","dst_host_srv_diff_host_rate"
                    ,"dst_host_serror_rate","dst_host_srv_serror_rate","dst_host_rerror_rate",
"dst_host_srv_rerror_rate"]
testdata = pd.read_csv('kddtest.csv', header=None)
testdata.columns = traindata.columns
traindata = traindata.drop(columns = ["num_outbound_cmds","is_host_login"])
testdata = testdata.drop(columns = ["num_outbound_cmds","is_host_login"])
traindata.head()
fig, ax = plt.subplots(figsize=(14, 6), dpi=80)
for i in range(6):
```

```
    ax.plot(traindata[traindata.columns[i]], label=traindata.columns[i], linewidth=1)
plt.legend()
plt.show()
trainvalues = traindata.values
trainvalues = trainvalues.astype('float32')
testvalues = testdata.values
testvalues = testvalues.astype('float32')

scaler = MinMaxScaler(feature_range=(0, 1))
trainvalues = scaler.fit_transform(trainvalues)

scaler = MinMaxScaler(feature_range=(0, 1))
testvalues = scaler.fit_transform(testvalues)

traindataset = trainvalues[0:len(trainvalues),:]
testdataset = testvalues[0:len(testvalues),:]

batch_size = 64
testdataset.shape
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
reframedtrain = series_to_supervised(traindataset)
```

```
reframedtrain.drop(reframedtrain.columns[list(i for i in range(41, 80))], axis=1, inplace=True)
reframedtest = series_to_supervised(testdataset)
reframedtest.drop(reframedtest.columns[list(i for i in range(41, 80))], axis=1, inplace=True)
reframedtrain[0:3]
timestep = 1
valuestrain = reframedtrain.values
valuestest = reframedtest.values
trainX, trainY = valuestrain[:,:-1], valuestrain[:,-1]
testX, testY = valuestest[:,:-1], valuestest[:,-1]
print(trainX.shape)
print(trainY.shape)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
print(trainX.shape, trainY.shape, testX.shape, testY.shape)
testY
l2_lambda = 0.0001
model = Sequential()
model.add(LSTM(128, input_shape=(trainX.shape[1], trainX.shape[2]), kernel_regularizer=l2(l2_lambda),
dropout=0.01 ,recurrent_dropout=0.01, return_sequences=True))
model.add(Dense(256, kernel_regularizer=l2(l2_lambda), activation='relu'))
model.add(Dropout(0.01))
model.add(BatchNormalization())
model.add(Dense(1, kernel_regularizer=l2(l2_lambda)))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
checkpointer = callbacks.ModelCheckpoint(filepath="C:/Users/Shigatau/Documents/Диплом/Intrusion-
Detection-Systems-master/dnn/kddresults/LSTM/check.hdf5", verbose=1, save_best_only=True, monitor='loss')
csv_logger = CSVLogger('C:/Users/Shigatau/Documents/Диплом/Intrusion-Detection-Systems-
master/dnn/kddresults/LSTM/training_set_dnnanalysis.csv',separator=',', append=False)
history = model.fit(trainX, trainY,  batch_size=batch_size,validation_data=(testX, testY), verbose=2,
epochs=100, callbacks=[checkpointer,csv_logger])
model.save("C:/Users/Shigatau/Documents/Диплом/dnn/kddresults/LSTM/dnn3layer_model.hdf5")
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
score = []
name = []
from numpy import concatenate
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
import os
for file in os.listdir("C:/Users/Shigatau/Documents/Диплом/dnn/kddresults/LSTM/"):
    model.load_weights("C:/Users/Shigatau/Documents/Диплом/dnn/kddresults/LSTM/"+file)
    yhat = model.predict(testX)
    testX1 = testX.reshape((testX.shape[0], testX.shape[2]))
    yhat = yhat.reshape((yhat.shape[0], yhat.shape[2]))
    inv_yhat = concatenate((yhat, testX1[:, 1:]), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:,0]
    # invert scaling for actual
    testY = testY.reshape((len(testY), 1))
    inv_y = concatenate((testY, testX1[:, 1:]), axis=1)
    inv_y = scaler.inverse_transform(inv_y)
    inv_y = inv_y[:,0]
    for i in range(len(inv_yhat)):
        if inv_yhat[i]>0.5:
            inv_yhat[i] = 1
        else:
            inv_yhat[i] = 0
    accuracy = accuracy_score(inv_y, inv_yhat)
    recall = recall_score(inv_y, inv_yhat , average="binary")
    precision = precision_score(inv_y, inv_yhat , average="binary")
    f1 = f1_score(inv_y, inv_yhat, average="binary")
    print("----------------------------------------------")
    print("accuracy")
    print("%.3f" %accuracy)
    print("recall")
    print("%.3f" %recall)
    print("precision")
    print("%.3f" %precision)
    print("f1score")
    print("%.3f" %f1)
    score.append(accuracy)
    name.append(file)
np.savetxt("C:/Users/Shigatau/Documents/Диплом/Intrusion-Detection-Systems-
master/dnn/dnnres/LSTMpredicted.txt", inv_yhat)
```