Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт кибербезопасности и защиты информации

Работа допущена к защите
Директор Института
кибербезопасности и защиты
информации, д.т.н., проф.
_____ Д.П. Зегжда
«___»_____2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ДИПЛОМНАЯ РАБОТА

ОБНАРУЖЕНИЕ ВРЕДОНОСНЫХ ПРИЛОЖЕНИЙ ДЛЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ANDROID С ИСПОЛЬЗОВАНИЕМ МЕТОДА ФАКТОРИЗАЦИИ МАТРИЦ

по направлению подготовки (специальности)
10.05.03 Информационная безопасность автоматизированных систем

Направленность (профиль)
10.05.03_08 Анализ безопасности информационных систем

Выполнил
студент гр. 4851003/60801 М.М. Башканков

Руководитель
доцент ИКиЗИ,
к.т.н. Е.Ю. Павленко

Санкт-Петербург

2021

РЕФЕРАТ

На 68 с., 10 рисунков, 5 таблиц.

КЛЮЧЕВЫЕ СЛОВА: ANDROID, ОБНАРУЖЕНИЕ ВРЕДОНОСНЫХ ПРИЛОЖЕНИЙ, МАШИНЫ ФАКТОРИЗАЦИИ, УЧЁТ СПЕЦИФИКИ ПОЛЕЙ, БЕЗОПАСНОСТЬ МОБИЛЬНЫХ УСТРОЙСТВ

Тема выпускной квалификационной работы: «Обнаружение вредоносных приложений для ОС Android с использованием метода факторизации матриц».

Данная работа посвящена созданию и оценке систем обнаружения вредоносных приложений для ОС Android. Цель настоящей дипломной работы – обнаружение вредоносных приложений для ОС Android с использованием моделей, основанных на факторизации матриц. В ходе исследования решались следующие задачи:

1. Анализ существующих работ по теме обнаружения вредоносных Android-приложений;
2. Выделение значимых для безопасности характеристик Android-приложений;
3. Создание обучающей выборки путём обработки существующих наборов данных;
4. Разработка метода обнаружения вредоносных Android-приложений на основе факторизации матриц;
5. Программная реализация предложенного метода в виде прототипов;
6. Обучение и сравнение результатов обнаружения вредоносных Android-приложений разработанных прототипов.

Объектом исследования настоящей выпускной квалификационной работы являются приложения для ОС Android. Предметом исследования – обнаружение вредоносных Android-приложений с использованием нейронных сетей, содержащих в своей основе метод факторизации матриц.

В результате были получены высокие значения точности классификации – от 94,6% до 97%. Областью применения результатов могут стать государственные и частные системы обнаружения вредоносных приложений.

Кроме того, при частичной модификации результаты могут быть применены к системам, отличным от Android.

THE ABSTRACT

68 pages, 10 figures, 5 tables.

KEYWORDS: ANDROID, MALWARE DETECTION, FACTORIZATION MACHINES, FIELD-AWARE, MOBILE SECURITY

The theme of the final qualifying work: "Android malware detection based on the matrix factorization method".

This work is devoted to the creation and evaluation of systems for detecting malicious applications for the Android OS. The aim of this thesis is the detection of malicious applications for the Android OS using models based on matrix factorization. During the study, the following tasks were solved:

1. Analysis of existing works on the detection of malicious Android applications;
2. Examine the existing security-relevant features of Android applications;
3. Creation of a training sample by processing existing datasets;
4. Development of a method for detecting malicious Android applications based on matrix factorization;
5. Software implementation of the proposed method in the form of prototypes;
6. Training and comparison of the results of detecting malicious Android applications of the developed prototypes.

The research object of this final qualifying work is applications for the Android OS. The subject of the research is the detection of malicious Android applications using neural networks based on the matrix factorization method.

As a result, high values of classification accuracy were obtained - from 94.6% to 97%. The results can be applied to public and private systems for detecting malicious applications. In addition, with partial modification, the results can be applied to non-Android systems.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время такие мобильные устройства, как смартфоны стали неотъемлемой частью жизни людей. Чаще всего именно в смартфонах находится вся личная информация человека – фотографии, данные банковских карт, телефонные номера, логины и пароли от различных сервисов и приложений, которые могут храниться как в браузере, так и в менее защищённых местах – в заметках, файлах и других программных реализациях хранилищ, которые можно извлечь считыванием внутреннего носителя смартфона.

Кроме личных данных на смартфонах может находиться и корпоративная информация, составляющая коммерческую тайну. В этом случае, согласно ст. 14 Федерального закона от 29.07.2004 N 98-ФЗ (ред. от 09.03.2021) "О коммерческой тайне", при её раскрытии, человеку, ответственному за её хранение в тайне, либо, если информация добросовестно хранилась в тайне, то злоумышленнику, раскрывшему данные, может грозить ответственность вплоть до уголовной.

Несмотря на всю значимость защиты информации на мобильных устройствах, в Российской Федерации не существует нормативных документов, содержащих предписания по обеспечению устройств защитным программным, аппаратным или программно-аппаратным обеспечением при их производстве, продаже, использовании в личных или коммерческих целях, и, соответственно, обязывающих и регламентирующих соблюдать такие предписания, ввиду их отсутствия. Это приводит к тому, что производящие компании не имеют обязательств предустановки защитных средств и могут вовсе не внедрять их на производимые мобильные устройства. В связи с чем злоумышленники получают возможность воспользоваться уязвимостью мобильных устройств, в том числе смартфонов, чтобы заполучить личную или коммерческую информацию держателя устройства.

Одним из решений этой проблемы может стать надёжная система распознавания вредоносных приложений, не позволяющая пользователю установить

на смартфон вредоносную программу, а магазинам приложений или другим сервисам, взаимодействующим с приложениями – не допустить её распространения.

От качества и надёжности такой системы будет зависеть возможность злоумышленника получить конфиденциальные данные, нарушить целостность или доступность той или иной информации. Поэтому проектируемая система должна быть максимально точной, отказоустойчивой и не содержащей уязвимости, позволяющие обойти её.

Актуальность исследования заключается в активном развитии платформы Android ОС и её распространении в мировых масштабах. В связи с таким распространением растёт и количество вредоносных приложений, которые поставляются магазинами Android-приложений. Расположение таких приложений в открытых магазинах обуславливается слабой проверкой на вредоносность, либо вовсе отсутствием проверок при загрузке и выгрузке приложений любыми людьми. Ввиду свободного доступа пользователей к таким магазинам у злоумышленников появляется возможность свободного распространения любых вредоносных программ.

Объектом исследования настоящей выпускной квалификационной работы являются приложения для ОС Android.

Предметом исследования является обнаружение вредоносных Android-приложений с использованием нейронных сетей, содержащих в своей основе метод факторизации матриц.

Цель настоящей дипломной работы – обнаружение вредоносных приложений для ОС Android с использованием моделей, основанных на факторизации матриц. В ходе исследования решались следующие задачи:

– анализ существующих работ по теме обнаружения вредоносных Android-приложений;

– выделение значимых для безопасности характеристик Android-приложений;

– создание обучающей выборки путём обработки существующих наборов данных;

– разработка метода обнаружения вредоносных Android-приложений на основе факторизации матриц;

– программная реализация предложенного метода в виде прототипов;

– обучение и сравнение результатов обнаружения вредоносных Android-приложений разработанных прототипов.

Теоретической основой выпускной квалификационной работы послужили работы отечественных и зарубежных специалистов в областях:

– информационной безопасности платформы ОС Android и, в частности, особенностей структуры, написания и работы приложений в данной системе;

– обнаружения вредоносных приложений при помощи различных методов и нейронных моделей;

– моделей нейронных сетей, построенных на методе факторизации матриц.

Практическая часть работы выполнялась на основании документации к языку программирования Python, а также к его модулям xLearn и fastFM, которые не составляют часть набора стандартных библиотек языка. Кроме того, для более объективного понимания специфики работы и отличительных особенностей модулей xLearn и fastFM были изучены соответствующие им научные работы.

Информационную базу исследования составили практические и теоретические данные, полученные в течение учебной и преддипломной практик, а также материал, изученный при прохождении дисциплин «Безопасность операционных систем», «Введение в мобильные технологии и средства связи», «Основы искусственного интеллекта». Кроме того, информационной базой являются изученные работы, публикации и статьи по смежным с данной выпускной работой темам.

Степень научной разработанности проблемы. Проблема обнаружения вредоносных приложений для ОС Android является довольно популярной для исследований, ей посвящены такие труды, как [8–20, 22–32, 36, 43–46]. В этих же работах встречается и задача выделения характеристик безопасности в Android-приложения. Теоретические и практические основы моделей нейронных сетей, основанных на методе факторизации матриц описаны в [30, 33–35, 38–40].

Научная новизна представленного исследования заключается в:

– использовании расширенного набора характеристик безопасности Android-приложений в обнаружении вредоносных приложений при помощи моделей, основанных на факторизации матриц;

– применении модели нейронной сети «Машина факторизации с учётом специфики полей» для обнаружения вредоносных Android-приложений;

– сравнении практических реализаций модели нейронной сети «Машина факторизации» с точки зрения обнаружения вредоносных Android-приложений.

Практическая значимость выпускной квалификационной работы заключается в разработке и представлении результатов системы обнаружения вредоносных приложений с применением машины факторизации с учётом специфики полей. Также, практическая значимость содержится в представлении результатов сравнения практических реализаций машин факторизации.

1 ПРОБЛЕМА БЕЗОПАСНОСТИ МОБИЛЬНЫХ УСТРОЙСТВ

Мобильные устройства быстро стали популярны по всему миру, так как являются удобными аналогами стационарных устройств. Так, согласно [1], только количество пользователей смартфонов за 4 года, с 2016 года по 2020, выросло с 3,7 миллиардов до 6, что является абсолютным большинством от общего количества живущих людей на Земле, а по прогнозу на 2023 год – достигнет 4.3 миллиарда пользователей. Количество же продаваемых в год смартфонов, согласно [2], в последнее время находилось на уровне 1.5 миллиарда устройств в год, исключением стал 2020 год, когда количество проданных смартфонов снизилось до 1.38 миллиарда устройств в связи с пандемией, однако, согласно прогнозу, уже в 2021 году рынок восстановится от последствий пандемии и будет продано сопоставимое с 2019 годом 1.53 миллиарда устройств.

Мобильное устройство – довольно общее понятие, которое включает в себя телефоны, ноутбуки, смартфоны и другие легко перемещаемые цифровые устройства. Каждое из таких устройств, в зависимости от их операционной системы, выполняемых функций, аппаратных и программных составляющих, имеет свои тонкости в плане информационной безопасности и должны рассматриваться отдельно, так как являются результатом объединения большого количества конкретных составляющих сложного вычислительного цифрового устройства.

Вопросы безопасности каждого типа мобильного устройства должны решаться и исследоваться на протяжении всего его жизненного цикла – от разработки прототипа до последнего выпущенного устройства. Более того, каждая аппаратная и программная составляющая мобильного устройства должны исследоваться отдельно.

Мобильное устройство, с точки зрения информационной безопасности, может рассматриваться и цельно, но только при его представлении заказчикам или формировании отчёта о безопасности устройства. В иных же случаях, если компоненты устройства являются частью многих аналогичных устройств, то они

должны рассматриваться отдельно друг от друга. Так же и в этой работе – рассматривается отдельный компонент мобильных устройств – операционная система Android, а в частности – Android-приложения.

Особый интерес к Android-устройствам со стороны информационной безопасности в первую очередь вызван наибольшим распространением данной операционной системы среди мобильных устройств. Согласно [3], платформа Android завоевала более половины рынка мобильных устройств уже к 2014 году и удерживает позиции с весомым преимуществом - более 70% от рынка мобильных устройств - перед единственным на данный момент конкурентом, iOS.

Основную часть как Android, так и iOS системы составляют приложения – от игр до системных настроек. Однако, в отличие от iOS, позволяющей устанавливать приложения лишь из собственного строго контролируемого магазина App Store, приложения на платформу Android могут быть установлены как из доверенного места – магазина Google Play, так и из любых других мест – магазинов, сайтов, приложений, и в этом случае перед установкой пользователю будет показано предупреждение об опасности установки приложений из не доверенных мест и предложение разрешить установку из таких мест выставлением соответствующей настройки. После выставления этой настройки любое приложение, взятое из любого места, сможет быть установлено без дополнительных подтверждений.

Такая “открытость” системы Android проявляется не только в установке приложений, но и в других нюансах её работы, например, в выдаче разрешений приложениям. Другой тонкостью платформы является её программная открытость – любой разработчик может внести изменения в систему и вместе со своими обновлениями привнести новые проблемы. Кроме того, система Android дорабатывается производителями устройств, которые также могут вносить ошибки и уязвимости вместе с изменением платформы.

Магазин приложений Google Play содержит огромное количество программ. Согласно [4], в марте 2018 года их количество достигло 3.6 миллионов, однако к сентябрю уменьшилось на миллион в связи с введением новой политики

для разработчиков приложений, но количество приложений продолжает активно возрастать и достигает почти трёх миллионов приложений по состоянию на декабрь 2020 года.

Для эффективной фильтрации и проверки приложений на безопасность требуется высокоточная система, каковой Google Play Protect назвать сложно в связи с регулярно обнаруживающимися вредоносными приложениями в магазине Google Play. Так, например, самая актуальная находка – в конце января 2021 года было обнаружено 9 вредоносных приложений, подвергающих пользователей риску удалённого доступа и банковского вредоносного ПО, обходящего двухфакторную авторизацию банковских приложений [5]. Факт присутствия в Google Play такого серьёзного вредоносного ПО говорит о том, что системы обнаружения вредоносных приложений компании Google недостаточно практичны и надёжны.

В 2013 году Google заявил [6], что проблема безопасности Android устройств сильно завышена заинтересованными в этом сторонами. Однако, если в 2013 году данная информация и могла быть правдой в связи с тем, что Android занимал лишь 40% рынка, а количество пользователей смартфонов только превысило один миллиард, а количество приложений в Google Play только достигло одного миллиона, то сейчас ситуация стала совершенно иной – перечисленные показатели выросли в 3 раза, а в Google Play регулярно продолжают находить вредоносные приложения. Кроме того, как показывает статистика [7], количество вредоносных приложений стало резко расти после 2013 года в связи с увеличением популярности платформы.

* * *

Таким образом, в данном разделе была исследована популярность мобильных устройств, а в частности, устройств, работающих под управлением системы Android, достигает своего пика в последние годы. Количество пользователей мобильных устройств резко выросло за последние годы и продолжает расти сейчас. Количество злоумышленников, разрабатывающих вредоносные

программы, так же будет увеличиваться в связи с ростом числа потенциальных жертв.

В связи с этим, требуется постоянное совершенствование старых и разработка новых систем обеспечения безопасности. Одним из классов таких систем являются системы обнаружения вредоносных приложений, рассматриваемые в данной выпускной квалификационной работе. Кроме их рассмотрения, в следующих разделах исследуется эффективность и применимость систем, основанных на методе факторизации матриц.

2 ОБНАРУЖЕНИЕ И КЛАССИФИКАЦИЯ ВРЕДОНОСНЫХ ПРИЛОЖЕНИЙ

Теме обнаружения и классификации вредоносных приложений уделено множество работ, статей, докладов. Системы обнаружения вредоносных приложений довольно распространены, они входят в состав антивирусных средств, являются частью крупных сервисов, взаимодействующих со сторонними программами, например, таких, как Google Play. По этой причине такие системы имеют высокую ценность как для исследователей, так и для компаний, использующих их.

Принято выделять следующие методы обнаружения вредоносных приложений:

1. Статический.
2. Динамический.
3. Гибридный.

Далее будут углубленно рассмотрены перечисленные методы и соответствующие им работы для системы Android.

Статический метод обнаружения основан на анализе APK файла приложения и не учитывает поведение приложения во время работы. Путём разбора исходных кодов и манифеста приложения выделяются значащие характеристики безопасности, либо осуществляется поиск зловредного кода и на основе полученной информации делается вывод о безопасности приложения.

Так, в работах [8] и [9] рассматриваются методы статического обнаружения, основанного на характеристиках безопасности приложений и графах передачи управления, а в [10] – конкретно на поиске сигнатур. Однако данные системы практически неприменимы к реальным системам в связи со сложностью их автоматизации и не лучшими результатами обнаружения.

Динамический метод основан на анализе поведения приложения. Приложение запускается в песочнице для того, чтобы предотвратить негативные последствия и на основе анализа вызовов приложением сторонних API, системных

вызовов, обращения к памяти и сети, обнаруживает злонамеренные действия приложения.

Гибридный метод объединяет статический и динамический методы и выносит решение о безопасности приложения как на основе статических данных, так и динамических. В большинстве современных систем обнаружения вредоносных приложений используется именно этот метод. Примером такого метода является [11], где к статическому и динамическому методам добавляется ещё и кластеризация. Система показала достаточно высокую эффективность.

В основном, системы обнаружения и классификации работают на следующих принципах:

1. Сигнатурного обнаружения.
2. Вычисления условных вероятностей.
3. Машинного обучения.
    1) SVM;
    2) Random Forest;
    3) kNN;
    4) Naïve Bayes.

Системы сигнатурного обнаружения работают на основе сравнения базы обнаруженных ранее сигнатур вредоносных приложений и исходных кодов анализируемого приложения. Учитываются возможности изменения сигнатур, обфускации кода, однако данные системы не будут работать при шифровании исходного кода приложения. Из-за этого в настоящее время данные системы практически не используются и не реализуются.

Системы, работающие с условными вероятностями, заранее вычисляют условные вероятности появления характеристик безопасности в исходных кодах и манифесте для вредоносных и безопасных приложений. При анализе приложения учитываются ранее вычисленные вероятности для каждой характеристики безопасности и высчитывается вероятность того, что анализируемое приложение является вредоносным. Динамические системы, работающие на данном

принципе, вычисляют условные вероятности критических действий приложения и учитывают их при анализе.

Системы, работающие на основе машинного обучения наиболее популярны в настоящее время благодаря их точности обнаружения и возможности классификации вредоносных приложений. Наибольшую сложность в данном случае представляют:

– обучение систем;

– выбор оптимальных параметров.

Сложность обучения для различных систем приведена в Таблице 1, где n – количество обучающих примеров, d – размерность входного вектора.

Таблица 1 – Сложность обучения систем обнаружения и классификации [12]

Классификатор	Сложность
Random Forest	$O(nd\ log\ n)$
SVM	$O(max(n,d), min((n,d)^2))$
kNN	$O(ndk)$
Naïve Bayes	$O(nd)$

Существует большое количество реализованных систем на основе машинного обучения, работающих с различными методами обнаружения вредоносных приложений, одни из таких систем приведены в [13] и [14].

Кроме того, DroidMat [15] выполняет статический анализ файла манифеста и исходного кода приложения Android для извлечения характеристик безопасности, включая разрешения, аппаратные ресурсы и вызовы API. Затем он использует метод k-means и классификацию k ближайших соседей (k-NN) для обнаружения вредоносных программ.

DREBIN [16] извлекает аналогичные характеристики из файла манифеста и исходного кода приложения и использует метод опорных векторов (SVM) для классификации вредоносных программ на основе one-hot кодирования соответствующих характеристик.

Во многих работах исследуется и реализуется поиск шаблонов злонамеренного поведения с помощью графов потоков управления или графов вызовов. AppContext [17] классифицирует приложения, используя машинное обучение на основе контекстов, которые извлекаются в случае обнаружения подозрительного поведения приложения. Создаётся граф вызовов из исходного кода приложения и извлекается контекст при помощи анализа потоков информации внутри приложения. Затем AppContext получает особые характеристики безопасности из извлеченного контекста, которые затем используются в алгоритмах машинного обучения. В статье [17] было проанализировано 633 безопасных приложения из магазина Google Play и 202 образца вредоносных программ. AppContext верно идентифицировал 192 вредоносных приложения с точностью 87,7%.

Гаскон и соавторы в [18] так же использовали графы вызовов для обнаружения вредоносных программ. После извлечения графов вызовов из приложений, в их реализации применяется графическое ядро и за линейное время из графов вызовов извлекаются характеристики безопасности. Полученные характеристики подаются на вход SVM, которая в свою очередь классифицирует приложение как вредоносное или безопасное. Был проведён эксперимент со 136 тысячами безопасных и 12 тысячами вредоносных приложений. В рамках эксперимента система верно выявила 89% вредоносных программ с 1% ложных срабатываний. Минус данного метода в том, что он сильно зависит от точности извлечения графа вызовов.

Несмотря на относительно позитивные результаты перечисленных работ, в настоящее время даже такие системы, как FlowDroid [19] и IC3 [20], всё ещё не могут полностью решить проблему построения межкомпонентных потоковых графов управления (ICFG), особенно потоков, создающихся намерениями и фильтрами намерений.

Большим плюсом данных систем является то, что классифицировать вредоносные приложения они могут как по заранее выделенным классифицирующим свойствам, так и по тем, что они сами выделят в процессе обучения.

Системы обнаружения вредоносных приложений, основанные на машинном обучении, имея возможность работать как на основе результатов статического, так и динамического анализа, а следовательно, и гибридного, представляются мощным, расширяемым и самым эффективным инструментом на данный момент. В связи с этим, такие системы активно развиваются и совершенствуются, а исследователи стараются достичь их наибольшей точности обнаружения, применяя различные методы машинного обучения.

При рассмотрении вопроса безопасности мобильных устройств невозможно не упомянуть о типах вредоносных приложений. Google выделяет 14 категорий, относящихся к Android устройствам [21]. В них входят как стандартные типы вредоносных приложений:

– использующие чёрные ходы;
– вызывающие отказ в обслуживании;
– загрузчики;
– осуществляющие фишинг;
– повышающие привилегии;
– вымогатели;
– рассылающие спам;
– шпионские программы;
– троянские программы.

Так и типы, относящиеся конкретно к платформе или к мобильным устройствам в общем:

– программы-мошенники с выставлением счетов – это тип мошенничества с отправкой платных SMS-сообщений, осуществлением дорогостоящих звонков и подпиской на платные телефонные услуги;

– коммерческое шпионское ПО – предназначены для слежки за устройством, передачи личной информации с устройства без уведомления и согласия пользователя;

– программы, выполняющие рутинг, то есть получение устройством прав суперпользователя root, без предупреждения;

– приложения, содержащие нестандартную для Android устройства вредоносную функциональность, а также содержащие угрозы для иных платформ.

* * *

Таким образом, в данном разделе были исследованы методы обнаружения вредоносных Android приложений и приведена их официальная классификация от Google. Вредоносные приложения для системы Android имеют расширенную специфику ввиду особенности самой платформы Android, работы в ней программ и содержащейся в ней системы безопасности. В связи с этим, классификация включает как стандартные типы вредоносных приложений, так и относящиеся конкретно к мобильным устройствам и Android в частности.

3 ИССЛЕДОВАНИЕ СУЩЕСТВУЮЩИХ РАБОТ

Как и говорилось ранее, тема обнаружения вредоносных приложений для системы Android является довольно развитой и рассматривается во множестве работ. В данном разделе будут описаны наиболее примечательные из трудов, в какой-то мере повлиявшие на ход дальнейшего развития темы исследований безопасности системы Android. На Рисунке 1 приведена временная прямая, отображающая развитие темы обнаружения вредоносных приложений для системы Android.

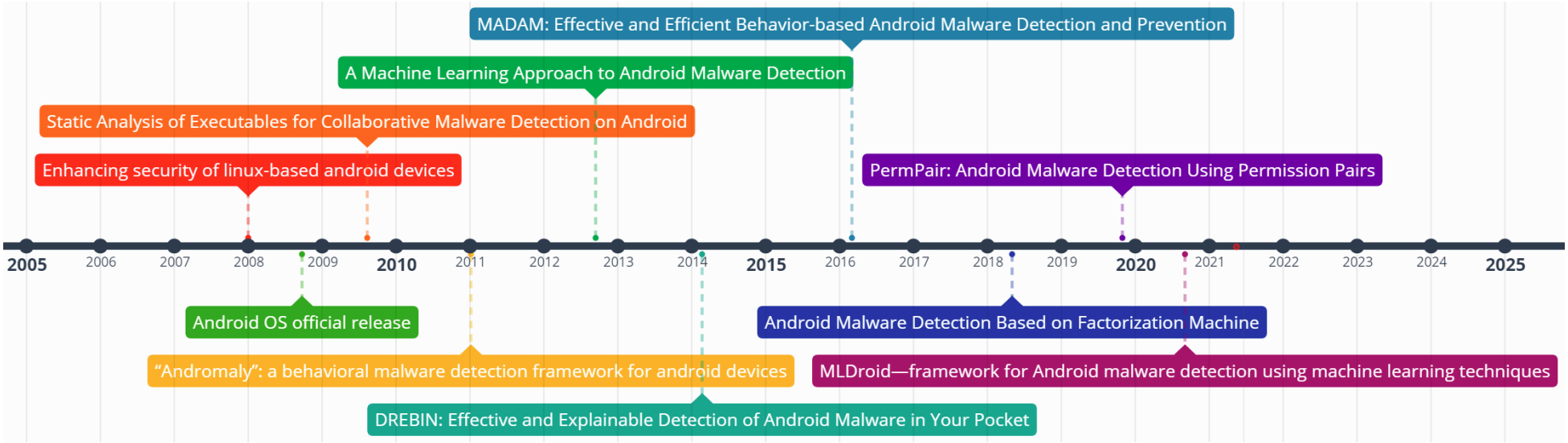

Рисунок 1 – Историческое развитие темы обнаружения вредоносных приложений для Android

Исследователи озаботились вопросами безопасности операционной системы Android и стали предлагать различные решения для её усовершенствования ещё до её официального выпуска. В январе 2008 года была представлена первая работа [22] по обнаружению вредоносных приложений, развитие которой было представлено в 2009 году [23], когда стало возможно оценить первые официальные версии платформы Android. Работы были написаны Обри-Дерриком Шмидтом и компанией из 7 человек из Берлина и Стамбула [22, 23]. Обе работы опираются на более ранние труды, относящиеся к другим мобильным системам и устройствам, где предлагалось использование сервера, осуществляющего анализ безопасности приложений. Так, каждое приложение, установленное на устройство, согласно работам, должно пересылаться на сервера компании, поставляющей операционную систему и анализироваться там. В работах же [22, 23] предлагается аналогичный подход, но с совершенствованием системы Android так, что ещё и все устройства снабжались модулем анализа приложений, который основан на алгоритмах PART, Prism и Ближайших соседей. Таким образом подразумевалась мгновенная проверка приложения и рассылка всем устройствам информации о данном приложении.

Кроме того, в работе [23] для расширения возможностей обеспечения безопасности системы предлагается использование Linux-утилит: антивируса, firewall, детекторов руткитов и других инструментов, используемых для анализа, обнаружения и предотвращения рисков безопасности на системном уровне.

Минусами данных подходов стали: использование сторонних устройств, то есть серверов, которые должны кем-то поддерживаться; использование лишних ресурсов мобильных устройств, в том числе, заряда батареи, что было особо критично для того времени; рассылка информации о приложениях должна была осуществляться через мобильные сети, либо сети Wi-Fi, которые ещё не были сильно распространены, для этого понадобился бы дополнительный протокол с постоянной синхронизацией всех устройств, работающих под ОС Android, что снова вызвало бы дополнительный расход ресурсов; расширение встроенной системы Linux и поддержка работы сторонних утилит.

Хоть предложенные улучшения и не были реализованы в Android, это был первый шаг к созданию систем обнаружения вредоносных приложений, который положил начало развитию одной из самых популярных тем для исследования в настоящее время.

В 2010 году была опубликована работа Томаса Близинга и других представителей немецкого университета [24], где для динамического анализа было предложено использование песочниц, в которых работают Android-приложения. В работе предложен гибридный способ обнаружения – во время статического анализа ищутся и логируются подозрительные и потенциально опасные функции и разрешения, во время динамического анализа подсчитывается и логируется количество системных вызовов за время работы приложения.

Минусами предложенной системы являются: дополнительная нагрузка на устройство при гибридном анализе; только логирование информации, без реагирования; использование субъективной информации для логирования – предполагается, что она может указывать на вредоносное поведение, но всё зависит от решения человека, оценивающего логи.

В заключении работы указано, что для данной системы целесообразно использовать машинное обучение для исключения человека из оценки логов приложения. Это предположение было первым упоминанием машинного обучения для обнаружения вредоносных приложений и, несомненно, верным.

К 2011 году было написано несколько работ, в некоторой степени связанных с машинным обучением. Так, в работе Икера Бургера и других [25] был предложен фреймворк Crowdroid, который основан на алгоритме k-means. Фреймворк осуществляет мониторинг системных вызовов и создаёт вектор, содержащий количество соответствующих системных вызовов, задействованных в приложении. Затем, используя алгоритм k-means, фреймворк выдаёт вердикт – является ли приложение троянским конём.

Основным минусом фреймворка является то, что алгоритм k-means не обучается заранее, как это делается в машинном обучении, он использует результаты, получаемые от клиентов. В связи с этим, пока не будет образован набор из

достаточного количества примеров, алгоритм не будет точен. Из-за данной особенности вытекают и другие проблемы: во-первых, результаты со стороны клиентов могут быть недостоверными, тогда система будет скопрометирована; во-вторых, алгоритм k-means будет всегда делить векторы данных на k классов, в данном случае 2 – вредоносные и добросовестные приложения, даже если в наборе данных алгоритма будут только добросовестные приложения.

Описанные проблемы решаются использованием предварительного обучения системы, то есть использованием машинного обучения. Однако, в 2010–2011 годах ещё не было достаточно данных о реальных вредоносных Android-приложениях, из-за чего создание такой системы было невозможно.

В работе Асафа Шабтая и других [26] было впервые применено машинное обучение для обнаружения аномалий в системе Android. Было использовано несколько алгоритмов для сравнительного анализа. Данными для обучения и обнаружения являлись системные метрики – потребление ЦП, использование сети Wi-Fi и другие. Так как примеров вредоносных приложений не было, были созданы 4 приложения, реализующие DoS атаки и кражи данных.

Это была первая работа, использующая машинное обучение, но практическая часть имела ряд минусов – маленький, искусственно созданный набор обучающих примеров, снятие метрик, относящихся не к конкретным приложениям, а ко всему устройству и другие минусы. Но благодаря проявленному интересу к машинному обучению уже в 2010–2011 годах, в следующие года и до нашего времени стали активно появляться работы, использующие различные наборы данных и алгоритмы машинного обучения для достижения наилучших результатов в обнаружении вредоносных приложений.

В 2012 году Яджин Чжоу и Сюйсянь Цзян [27] впервые описали вредоносные приложения, найденные в течение 2010–2011 годов на платформе Android. Данные приложения располагались как в официальных магазинах, так и в сторонних. В работе были проверены 4 основных на то время антивируса и все они показали довольно слабый результат – от 80% до 20% обнаруженных вредоносных приложений. Логичным выводом из данной работы стало то, что

необходимо создание надёжной системы обнаружения вредоносных приложений, доступной для Android-устройств. Кроме того, была представлена статистика использования вредоносными приложениями Android-разрешений и сделан вывод, что требуется введение более детальных разрешений, так как на тот момент все они являлись довольно общими. Впоследствии система постепенно обеспечивалась всё большим количеством более узких разрешений.

В работе Джастина Сахса и Латифура Хана [28] использована наиболее подходящая для того времени модель машинного обучения One-Class SVM. Её преимущество состоит в том, что почти не имея примеров одного из классов, в данном случае это вредоносные приложения, систему можно обучить на примерах второго класса, добросовестных приложениях, и при появлении примера, сильно отличающегося от обучающих, будет установлено, что он является вредоносным приложением. Для такой системы нужно большое количество характеристик безопасности, поэтому в работе предлагается использование всех разрешений, а также графа потока управления.

В 2014 году Даниэль Арп и другие [16] предложили легковесный фреймворк для Android-устройств на основе машины опорных векторов, который показал довольно хорошие результаты. Кроме того, для обучения модели был создан датасет Drebin, который стал одним из самых популярных для использования в обучении моделей для обнаружения вредоносных Android-приложений.

В 2016 году была представлена система обнаружения и предотвращения вредоносных программ MADAM от Андреа Саранчино и других [29]. Система работает на Android-устройствах в виде приложения и, что отличает её от других систем, не только обнаруживает, но и предотвращает работу подозрительных приложений на контролируемом устройстве. Для выявления аномального поведения система контролирует системные вызовы, отправку SMS, пользовательскую активность и метаданные приложений. Соответственно, для такого контроля требуются права уровня суперпользователя, поэтому контролируемое устройство должно быть рутованым, что является большим минусом и ограничением для использования.

В 2018 году Ченлинь Ли и другие [30] опубликовали работу, в которой представлена система обнаружения вредоносных приложений, основанная на машине факторизации – одной из рассматриваемых в данной работе моделей машинного обучения. Исследования, представленные в данной выпускной квалификационной работе мотивированы именно трудом [30].

Аншул Арора и другие [31] в 2019 году предложили систему, основанную на парах разрешений, которые являются наиболее весомыми для вредоносных приложений. В работе используется графовая структура, для которой постепенно выполняются различные оптимизации с вычислением весов для каждой из пар. Итоговые значения весов используются для классификации приложений на вредоносные и добросовестные.

Арвинд Махиндру и А. Л. Сангал [32] в 2020 году представили работу, в которой описана веб-система, использующая 21 метод машинного обучения для получения наилучших результатов обнаружения вредоносных Android-приложений. Кроме того, были исследованы различные функции и признаки, а также их влияние на точность обнаружения.

* * *

Таким образом, в данном разделе было исследовано развитие темы обнаружения вредоносных приложений для системы Android. Из количества работ и глубины проработки в них темы можно сделать вывод, что она является довольно развитой и продолжает совершенствоваться по сей день. Со временем точность обнаружения предлагаемых систем увеличивается, они оптимизируются и могут использоваться на устройствах с малым объёмом ресурсов.

4 ХАРАКТЕРИСТИКИ БЕЗОПАСНОСТИ В ANDROID-ПРИЛОЖЕНИЯХ

Под характеристиками безопасности Android приложений понимаются такие компоненты приложения, которые затрагивают аспекты безопасности Android устройств и могут быть использованы злоумышленниками для нарушения информационной безопасности устройства: кражи конфиденциальной информации, получения удалённого доступа к устройству, причинения вреда как самому устройству, так и его программным компонентам.

Общая схема обработки Android приложения для выделения характеристик безопасности состоит из следующих шагов:

1. Распаковка и декомпиляция исходных кодов приложения.
2. Выделение характеристик безопасности из файла манифеста и декомпилированных кодов.
3. Кодирование характеристик безопасности во входной вектор для обучения и тестирования нейросетевой модели.

4.1 Классификация

Все характеристики безопасности приложений можно разделить на категории согласно их ролям:

1. Список компонентов приложения. Компоненты декларируются в файле манифеста приложения и объявляют различные интерфейсы для взаимодействия как с конечным пользователем, так и с ОС Android. Имена данных компонентов используются для идентификации известных вредоносных приложений по его компонентам.
2. Используемые аппаратные средства. Если приложение нуждается в предоставлении доступа к таким аппаратным средствам устройства, как GPS, камера, сенсоры, то данные характеристики должны быть объявлены в файле манифеста приложения. Предоставление прав доступа к некоторому набору аппаратных средств может указывать на возможность утечки чувствительных

данных, например, задекларированный доступ к GPS и сети может указывать на передачу злоумышленникам местоположения устройства.

3. Разрешения приложения. Android использует механизм разрешений для контроля и сохранения конфиденциальности пользователя. Разрешения используются для предоставления прав приложению к чувствительным данным, компонентам аппаратных средств и доступу к сторонним библиотекам и интерфейсам взаимодействия приложений. Вредоносные приложения используют особые наборы разрешений по тому же принципу, что и аппаратные средства. Кроме того, имеется набор как обычных разрешений, так и опасных. Их отличие состоит в том, что опасные разрешения запрашиваются у пользователя непосредственно во время работы приложения, при этом предупреждая его об опасности предоставления таких разрешений. Именно на основе большого количества запрашиваемых опасных разрешений строится большинство вредоносных программ, поэтому они учитываются в большинстве систем обнаружения и классификации вредоносных приложений.

4. Фильтры намерений (intent filter). Intent – намерение приложения связаться с другим приложением или компонентом для получения или передачи какой-либо информации. Это основной способ “общения” различных приложений, как системных, так и прикладных. Фильтр намерений используется для того, чтобы отбрасывать ненужные для приложения намерения от других приложений и компонентов и принимать намерения, проходящие по критериям запрошенных действий, категорий намерений и передающимся данным. Как правило, вредоносные приложения производят фильтрацию намерений для получения только чувствительной и действительно нужной информации.

Кроме стандартных категорий характеристик безопасности, которые можно получить из манифеста приложения, можно выделить особые, которые помогут более точно определить приложение как вредоносное и содержатся непосредственно в коде приложения:

1. Подписки на широковещательные сообщения (broadcast receiver). Широковещательные сообщения, аналогично намерениям, содержат

рассылаемую информацию приложениям и компонентам, однако, в отличие от намерений, направлены не на конкретное приложение, а на все, подписанные на соответствующие сообщения. Если приложение, которому направлено широковещательное сообщение не запущено, то оно запускается, обрабатывает сообщение и в зависимости от результата обработки либо завершается, либо продолжает работу, запуская соответствующую активность или сервис. Примером широковещательного сообщения может быть сообщение от контроллера батареи о низком заряде батареи устройства. Соответственно, вредоносное приложение, подписываясь на широковещательные сообщения может контролировать состояние устройства, приложений и их компонентов.

2. Используемые поставщики контента (content provider). Поставщики контента являются общими базами данных, предоставляемыми приложениями для других приложений. Примерами поставщиков контента могут быть браузеры, контакты, медиа-приложения, настройки. Соответственно, поставщики могут хранить чувствительную информацию, которая является целью для злоумышленников, поэтому вредоносные приложения не редко запрашивают данные у поставщиков.

3. Встроенные API. Система Android предоставляет список стандартных API для работы с чувствительными данными, доступ к которым предоставляется посредством разрешений. Во-первых, если вызов данных API производится без запроса на это разрешения, то, возможно, данное приложение является root эксплойтом. Во-вторых, вызов функций API, работающих с чувствительными данными, может так же указывать на вредоносность приложения.

4. Сторонние API. При использовании сторонних API, некоторые функции могут выполняться без предоставления соответствующих разрешений по различным причинам. Зная о существовании таких представляющих опасность API и обнаруживая их в приложениях, можно предполагать, что такие приложения являются вредоносными.

5. Разрешения, запрашиваемые в коде приложения. Выделяя не только запрашиваемые разрешения из файла манифеста, но и конкретно используемые

при вызове API и их соответствующих функций, можно дополнить список разрешений, так как предоставление прав может осуществляться не только из файла манифеста, но и из кода приложения.

6. Граф потока управления. Выполнение каждого приложения происходит последовательно – функция за функцией. Таким образом, можно создать граф, отображающий последовательность вызовов функций приложения. При его визуализации становится понятно, как работает приложение, каков его алгоритм. Однако, в Android приложениях может присутствовать не одна точка входа, а множество – оно может запускаться путём его открытия пользователем, может выполнять задачи в виде сервиса, может открываться и осуществлять действия при получении широковещательных сообщений и так далее. Количество точек входа будет зависеть от количества объявленных интерфейсов взаимодействия в приложении. С точки зрения обнаружения вредоносных приложений становится интересно сформировать последовательность вызовов API функций, так как одни и те же API могут одинаково часто появляться и во вредоносных, и в добросовестных приложениях, однако, порядок их вызова может указать на возможные вредоносные действия приложения.

7. Динамические характеристики. Извлекаемые во время динамического анализа динамические характеристики позволяют выявить аномалии, отклонения и подозрительное или вредоносное поведение при работе приложений. Такими характеристиками могут быть потребляемые приложением ресурсы устройства, частота и адреса обращения в сеть Интернет, а также передаваемые при этом данные, отправка и принятие SMS, выполнение криптографических операций, подгрузка DEX модулей и множество других операций, которые могут указывать на вредоносность приложения.

4.2 Методы поиска

Установочный APK файл приложения – это сжатый архив, который содержит исходный код приложения, файлы ресурсов, метаданные, дополнительные библиотеки и файл манифеста приложения. Исходный код закодирован в файл с

расширением DEX (Dalvik Executable), который интерпретируется и выполняется виртуальной машиной Dalvik, которая в данное время заменена на ART, но выполняет все те же функции. Dalvik и ART – это среды выполнения приложений, реализованные в ОС Android. Файл манифеста содержит основную информацию о приложении, а также различные объявления и спецификации, в которых нуждается приложение. Файлами ресурсов являются используемые в приложении изображения, HTML файлы, строки.

Поскольку DEX файлы скомпилированы в исполняемый двоичный код, они не предназначены для чтения или интерпретации, исходный код не может быть извлечён из них напрямую. Поэтому, DEX файлы нужно дизассемблировать и декомпилировать в исходные форматы, которые будет возможно считать и интерпретировать. Такие форматы представляются Smali кодом (при дизассемблировании) и Java кодом (при декомпиляции). Smali код – байт-код машины Dalvik, который является таким же низкоуровневым языком, как и ассемблер (поэтому процесс перевода DEX файла в Smali код называют дизассемблированием).

Для получения дизассемблированного и декомпилированного кода есть множество утилит. Один из сценариев декодирования для ручного анализа приложения может представляться следующим образом:

1. Разархивация APK файла и декомпиляция исходных кодов программой Apk Manager.
2. Конвертация DEX файла в формат JAR Java кода.
3. Отображение полученного Java кода в утилите jd-gui.

Кроме этого, можно воспользоваться универсальной утилитой ApkTool, функционал которой довольно широк и удобен для получения исходных кодов для последующей работы с ними посредством других программ.

В описанных методах придётся дополнительно использовать сторонний парсер, либо реализовывать его самому, кроме этого, могут возникнуть проблемы с передачей результатов анализа в программу для их обработки. Однако, существуют пакеты, написанные на языке программирования Python для анализа

и декомпиляции APK файлов. Их преимущество состоит в том, что результаты анализа могут быть переданы в виде структур данных непосредственно в нужную программу, модуль или функцию, которая уже будет работать с результатами анализа. Наиболее примечательный пакет в данном случае – Androguard, который предоставляет широкий функционал и примечателен тем, что в нём встроен парсер, благодаря которому можно получить нужные компоненты из APK файла приложения, в том числе, перечисленные ранее характеристики безопасности, для их дальнейшего использования в оценке безопасности приложения.

4.3 Кодирование

Так как характеристики безопасности являются строками, либо структурами данных, а нейронные сети работают с числами и векторами, то после их получения путём анализа APK появляется задача кодирования характеристик для дальнейшей подачи в систему обнаружения для её обучения, тестирования или проверки приложения.

Представление такого большого количества данных, как характеристики безопасности, разумно в виде N-мерного бинарного вектора. Каждое приложение представляется таким N-мерным бинарным вектором, где N – количество характеристик безопасности, используемых системой обнаружения вредоносных приложений.

Для конкретизации предположим, что все когда-либо извлечённые характеристики формируют набор характеристик S размера $|S|$, тогда можно представить каждый APK файл в виде двоичного вектора длины $|S|$, элементы которого равны 1 в случае, если они задействованы в данном приложении, и 0 – в противном случае. Пример такого кодирования представлен на Рисунке 2, на нём же представлен и пример формирования набора характеристик S из характеристик приложений A и B.

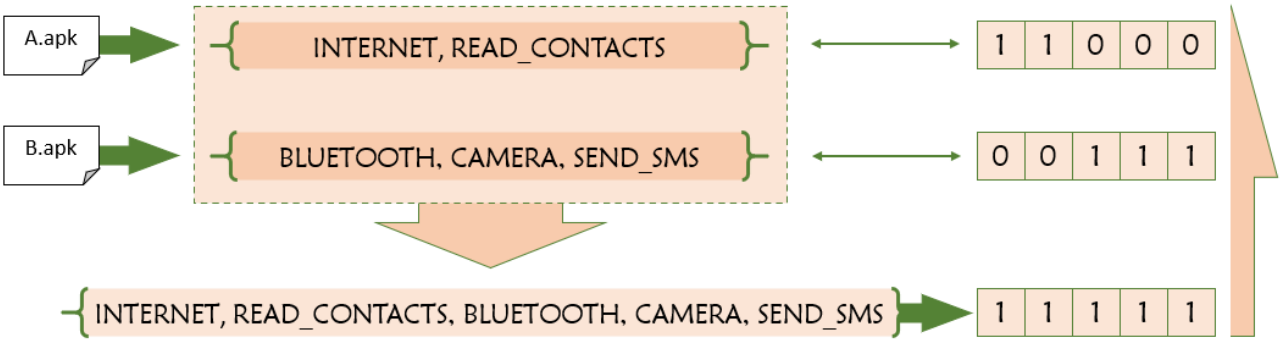

Рисунок 2 – Кодирование характеристик безопасности

Данный метод кодирования называется One-hot кодированием, когда некоторому набору элементов сопоставляется бинарный вектор, размер которого равен количеству элементов и каждый элемент данного вектора однозначно связан с элементом набора. Тогда каждый объект, содержащий некий набор этих элементов, может быть представлен в виде такого бинарного вектора, содержащего 1 на позициях элементов, принадлежащих данному объекту и 0 на остальных.

* * *

Таким образом, в данном разделе были исследованы характеристики безопасности Android приложений. Было выделено 11 групп характеристик согласно их роли в приложении и на устройстве. Сделан вывод о том, что наиболее удобным и подходящим способом извлечения характеристик из Android приложений является Python модуль Androguard в силу его широких возможностей. Описан метод One-hot кодирования для формирования входного вектора системе обнаружения вредоносных приложений из доступных в приложении характеристик безопасности.

5 МЕТОД ФАКТОРИЗАЦИИ МАТРИЦ

Факторизация (разложение) матрицы – это метод, который позволяет упростить работу с матрицей путём её разложения на две сокращённые матрицы. Данный метод широко применяется в компьютерных вычислительных системах, так как позволяет упростить и ускорить работу как над матрицами большого размера, так и над сильно разрежёнными матрицами, то есть содержащими большинство нулевых элементов.

На практике существует множество методов разложения матриц, которые применяются как для решения задач линейной алгебры, так и для других задач, например, для задачи построения моделей машинного обучения.

В данной работе наибольший интерес представляет разложение Холецкого. Данное разложение может применяться к симметричной положительно определённой матрице, а результатом разложения будет нижняя треугольная матрица и матрица, имеющая вид её транспонирования. Пример такого разложения представлен на Рисунке 3. Однако, основное преимущество данного метода состоит в том, что при разложении можно сильно сократить размер результирующих матриц без потери точности.

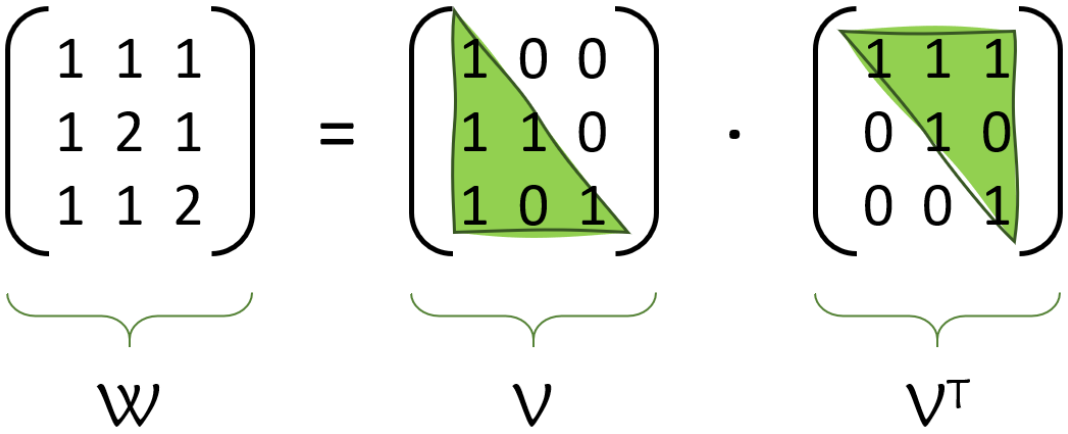

Рисунок 3 – Пример разложения Холецкого симметричной положительно определённой матрицы

5.1 Машины факторизации

В стандартных моделях нейронных сетей для обнаружения вредоносных приложений по вектору характеристик безопасности не учитывается так называемое “взаимодействие” характеристик. Очевидно, если в приложении присутствует несколько опасных характеристик, объединение некоторых из них может однозначно указывать на вредоносность приложения – именно в этом и заключается смысл учёта “взаимодействия” характеристик безопасности приложений.

“Взаимодействие” характеристик представлено их попарным влиянием друг на друга по принципу каждая-с-каждой, таким образом формируется матрица размером n^2, где n – количество характеристик. Так как характеристик большое количество, то только хранение матрицы размером их количества в квадрате требует больших затрат памяти, а учитывая то, что на каждом шаге обучения сети элементы матрицы должны меняться, то есть происходит постоянное взаимодействие с такой большой структурой, обучение потребует не только больших затрат памяти, но и времени.

Машины факторизации были представлены Штеффеном Рендлом [33] в 2010 году как класс новых нейросетевых моделей, совмещающих в себе преимущества машины опорных векторов (SVM) и факторизационных моделей. Машины факторизации могут выполнять задачи регрессии, классификации и ранжирования. Особенность машин факторизации заключается в обработке любых входных векторов, представляющих объекты реального мира, и оценке парных взаимодействий их элементов даже при сильной разрежённости этих векторов, в отличии от SVM, которые не могут работать с сильно разрежёнными данными, либо могут, но выдают низкую точность.

Ещё одно преимущество машин факторизации – линейная сложность $O(kn)$, где k – гиперпараметр, определяющий размерность факторизации, а n – количество признаков описываемого объекта (размер входного вектора).

Наибольшую популярность машины факторизации получили в рекомендательных системах, которые должны учитывать парные взаимодействия, при этом

располагая сильно разрежёнными данными. Однако, большинство данных, описывающих объекты реального мира, являются сильно разрежёнными в силу наличия множеств возможных признаков, но содержащих в себе лишь малую часть всех этих признаков. Так же и Android приложения – все вместе имеют множество, в зависимости от выбора, от десятков до сотен тысяч, характеристик безопасности, но отдельные приложения, чаще всего, включают в себя лишь несколько характеристик и в самых редких случаях порядка тысячи характеристик, что всё равно является малой частью от их общего количества.

Математически, идея взаимодействия компонентов x с весами w записывается в виде полиномиальной регрессии второго порядка:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} W_{ij} x_i x_j,$$

где $W -$ матрица весов взаимодействия x_i и x_j.

Как было описано выше, работа с такой матрицей не представляется возможной, однако, матрица W может быть разложена методом Холецкого:

$$W = VV^T.$$

Если представить v_i в качестве $i -$ ой строки V, нейронная сеть будет тренировать скрытый вектор v_i для каждого x_i и веса модели парного взаимодействия w_{ij} будут представлены как скалярные произведения соответствующих скрытых векторов для x_i и x_j:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \quad (1),$$

где скалярное произведение векторов v размерности k считается по следующей формуле:

$$\langle v_i, v_j \rangle = \sum_{m=1}^{k} v_{i,m} v_{j,m}.$$

Формула (1) и есть математическое представление модели машины факторизации. В модели машины факторизации параметры w_0, w_i, v_i тренируются и

обновляются с использованием стохастических методов, которые выполняют псевдослучайные изменения величин весов, сохраняя те изменения, которые ведут к улучшениям. На практике чаще всего применяется метод градиентного спуска.

Скрытые вектора v_i формируются в процессе обучения сети и чаще всего их размерность k во много раз меньше n – размера входного вектора x, в силу его разрежённости. Это гарантирует то, что не придётся хранить большое количество ненужных данных и работать с матрицей размера $n \times n$. Благодаря данному представлению сложность метода и снижается с $O(n^2)$ до $O(kn)$.

Машины факторизации могут быть расширены и до полиномиальной регрессии большего порядка для исследования взаимодействия более, чем двух признаков объекта:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=2}^{d} \sum_{i_1=1}^{n} \dots \sum_{i_l=i_{l-1}+1}^{n} \left(\prod_{j=1}^{l} x_{i_j} \right) \left(\sum_{f=1}^{k_l} \prod_{j=1}^{l} v_{i_j,f}^{(l)} \right) \quad (2).$$

В таком случае сложность равна $O(k_d n^d)$, но при математическом преобразовании уравнения (2) сложность сводится всё к той же линейной $O(kn)$. Доказательство этому приведено в работе [33].

5.2 Машины факторизации с учётом специфики полей

Модели, работающие по принципу машин факторизации с учётом полей, являются расширением стандартной модели машин факторизации. Они были предложены Рендлом и Шмидом-Тимом [34] в 2010 году, а также Андреасом Тёшером и другими на конкурсе KDD Cup в 2012 году. Однако, конкретная концепция машин факторизации с учётом полей была сформирована Ючин Цзюанем и другими [35] в 2016 году.

Для машин факторизации с учётом специфики полей кроме характеристик, использующихся в стандартной модели, вводится дополнительный параметр – поле. Поле – это класс, группа, либо любое другое объединение характеристик. Так, например конкретные разрешения Android приложений

(READ_CONTACTS, INTERNET и т. д.) являются характеристиками, а полем для них всех может быть «Разрешение». В отличии от довольно конкретных характеристик поля могут задаваться произвольно, например, разрешения могут быть поделены на такие поля как «Компоненты устройства» (CAMERA, BLUETOOTH и т. д.), «Мобильная связь» (CALL_PHONE, READ_SMS и т. д.), и другие, в зависимости от нужной гранулярности.

В отличии от стандартной машины факторизации, где с каждой характеристикой был связан только один скрытый вектор v_i, в расширенной модели с учётом специфики полей, при обучении, для каждой характеристики создаётся столько скрытых векторов, сколько введено полей, и полиномиальная регрессия второго порядка будет выглядеть следующим образом:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_{i,f_s}, v_{j,f_r} \rangle x_i x_j,$$

где $f_s -$ поле характеристики x_j, а $f_r -$ поле характеристики x_i.

Для лучшего понимания данной концепции можно рассмотреть пример на основе приведённых выше полей и характеристик. Допустим, модель построена на трёх характеристиках с соответствующими им полями, приведёнными в Таблице 2.

Таблица 2 – Характеристики Android приложений, разделённые на соответствующие поля

Компоненты устройства (КУ)	Мобильная связь (МС)	Местоположение (МП)
CAMERA (CAM)	CALL_PHONE (CP)	ACCESS_FINE_LOCATION (AFL)

Тогда для стандартной модели машины факторизации взаимодействие характеристик для произвольного приложения будет описываться следующим выражением:

$$\langle v_{CAM}, v_{CP} \rangle x_{CAM} x_{CP} + \langle v_{CAM}, v_{AFL} \rangle x_{CAM} x_{AFL} + \langle v_{AFL}, v_{CP} \rangle x_{AFL} x_{CP},$$

где x_n будет равно 1, если характеристика n содержится в приложении, и 0 в ином случае.

Соответственно, для машины факторизации с учётом специфики полей взаимодействие характеристик будет выглядеть таким образом:

$$\langle v_{CAM,\text{МС}}, v_{CP,\text{КУ}} \rangle x_{CAM} x_{CP} + \langle v_{CAM,\text{МП}}, v_{AFL,\text{КУ}} \rangle x_{CAM} x_{AFL} + \langle v_{AFL,\text{МС}}, v_{CP,\text{МП}} \rangle x_{AFL} x_{CP}.$$

Здесь скрытые векторы v совершенно иные – если в первом случае для характеристики CAMERA скрытый вектор v_{CAM} был одним для взаимодействий x_{CAM} с x_{CP} и x_{CAM} с x_{AFL}, то во втором случае, в связи с тем, что x_{CP} и x_{AFL} принадлежат различным полям, для характеристики CAMERA создаётся два скрытых вектора – $v_{CAM,\text{МС}}$ и $v_{CAM,\text{МП}}$.

5.3 Сравнение моделей

Несмотря на то, что модель машин факторизации с учётом специфики полей отличается от стандартной модели лишь увеличенным количеством скрытых обучаемых векторов, они будут по-разному проявлять себя в зависимости от решаемой задачи. Тогда выбор наиболее подходящей модели для соответствующей задачи будет гарантировать повышение точности классификации, регрессии и ранжирования. Поэтому важно выделить условия, при которых стоит использовать ту или иную модель машин факторизации. Плюсы и минусы каждой модели приведены в Таблице 3.

Таблица 3 – Сравнение моделей машин факторизации

	Стандартная модель	Модель с учётом специфики полей
Время обучения и распознавания (при одинаковом параметре k)	Меньше	Больше
Используемая память (при одинаковом параметре k)	Меньше	Больше

Таблица 3 – Сравнение моделей машин факторизации (продолжение)

		Стандартная модель	Модель с учётом специфики полей
Размерность k скрытых векторов, при которой достигается наибольшая точность		Больше	Меньше
Набор данных	Плотность	Любой	Сильно разрежённый
	Разделимость	Любой	На большое количество классов (полей)
	Тип данных	Любой	Категориальные

Рассматривая данный вопрос со стороны используемых ресурсов, то есть занимаемой памяти и затрачиваемого времени на тренировку и распознавание, машины факторизации с учётом специфики полей уступают стандартной модели при одинаковых параметрах моделей в связи с тем, что стандартная модель обучает и хранит лишь один скрытый вектор для каждой характеристики, а расширенная – по вектору на каждое поле.

С другой стороны, в связи с тем, что машины факторизации с учётом специфики полей обучают скрытые вектора относительно отдельных полей, а не на основе всех характеристик, как в стандартной модели, размерность k скрытых векторов может быть в разы меньше при той же точности распознавания в отличии от обычных машин факторизации.

Кроме того, в оригинальной работе [35] доказано, что машины факторизации с учётом специфики полей будут показывать лучшие результаты при достаточной разрежённости данных, в которых может быть выделено достаточно большое количество полей. В этой же работе модель была протестирована как на категориальных, так и на числовых данных и сделан вывод, что для числовых данных результаты работы машин факторизации с учётом специфики полей не лучше, чем стандартной модели.

* * *

Таким образом, в данном разделе был исследован метод факторизации матриц и основанная на нём модель машинного обучения – машины факторизации, а также её расширение – машины факторизации с учётом специфики полей. Математической основой моделей, помимо факторизации, является полиномиальная регрессия второго порядка для исследования парного взаимодействия характеристик и высшего порядка для, соответственно, взаимодействий высшего порядка. Было произведено сравнение моделей и сделан вывод, что расширенная модель должна применяться в случае сильной разрежённости данных, которые должны являться категориальными, а также разделимыми на достаточное количество полей (классов). Данные Android приложений подходят под перечисленные условия из чего можно сделать вывод, что при применении модели машин факторизации с учётом полей точность обнаружения вредоносных приложений должна быть выше, чем у стандартной модели.

6 РАЗРАБОТКА СИСТЕМ ОБНАРУЖЕНИЯ

Реализуемые системы обнаружения вредоносных Android-приложений являются бинарными классификаторами, так как проблема обнаружения основана только на двух классах – вредоносных и безопасных приложений. Таким образом, не требуется использование сложных моделей, выполняющих множественную классификацию.

Весь код написан на языке программирования Python ввиду его простоты и большого количества реализованных модулей. Включаемые в реализуемые системы модули не поддерживают ОС Windows, поэтому создание систем обнаружения происходило на ОС Ubuntu 18.04.

Для создания обучающей выборки, а также самого обучения требуется как можно большее количество оперативной памяти и доступных CPU, иначе процесс будет уничтожен ядром. Поэтому была выделена виртуальная машина с ОС Ubuntu на базе института. Системе было выделено 32 Гб ОЗУ и 8 ядер CPU от процессора Intel Xeon CPU E7- 8860 с рабочей частотой 2.27 ГГц.

6.1 Доступные практические данные

Для разработки систем первым делом надо проанализировать доступные наборы данных и уже реализованные библиотеки машинного обучения для того, чтобы избежать выполнения работы, которая была сделана ранее в других работах.

6.1.1 Датасеты

Для систем обнаружения вредоносных Android приложений датасетами являются сами приложения. Наибольшую сложность представляет то, что большинство наборов являются закрытыми ввиду содержания в них вредоносных примеров и к ним сложно получить доступ. Кроме того, если набор данных является не надёжным, то нужно дополнительно проводить анализ содержащихся

в нём приложений на вредоносность или безопасность при помощи дополнительных средств – антивирусов, систем обнаружения и других инструментов.

Одним из наиболее популярных наборов данных является DREBIN [16]. Данный набор содержит образцы вредоносных программ, собранных с 2010 по 2012 года. В него включено 5560 приложений из 179 семейств вредоносных программ. DREBIN является закрытым набором данных и получить к нему доступ можно только связываясь с его держателями по почте.

Для ещё двух популярных закрытых датасетов, Genome [27] и AMD [36], вовсе были прекращены выдачи разрешений на доступ и получить сейчас их не представляется возможным. Однако это были большие и качественные наборы. Genome содержал более 1200 приложений, покрывающих большинство существующих на тот момент семейств вредоносных программ и собранных с 2010 по 2011 года. В AMD входило 24553 образца, разделенных на 135 разновидностей среди 71 семейства вредоносных программ, собранных в период с 2010 по 2016 год.

Самый большой на данный момент из известных наборов данных – AndroZoo [37]. Он постоянно расширяется и на конец мая 2021 года содержит более 15 миллионов приложений. AndroZoo собирает приложения из нескольких источников, в том числе, из официального магазина Android приложений Google Play. После попадания в базу данных AndroZoo приложение анализируется десятками различных антивирусов для однозначного определения того, является ли приложение безопасным.

VirusShare – большой закрытый репозиторий, содержащий образцы вредоносных приложений различных семейств под основные системы. Обнаруженные приложения под ОС Android разделены на года согласно их публикации. Всего в репозитории содержится более 180 тысяч вредоносных Android приложений. Доступ к ним можно получить путём отправления запроса по электронной почте.

Наборы данных Канадского института кибербезопасности UNB находятся в открытом доступе и содержат большое количество репозиториев, созданных на базе института. Репозитории содержат как сами приложения – вредоносные и

безопасные, так и сформированные CSV файлы, содержащие вектора характеристик приложений и готовые к подаче в системы машинного обучения.

Кроме того, приложения содержатся и в магазинах Android приложений – Google Play и на альтернативных платформах для распространения приложений. Однако при их ручном сборе нельзя быть уверенным в их безопасности, нужно будет организовывать работу по их проверке и формировании баз безопасных и вредоносных приложений. Такой способ чаще всего является излишним, так как перечисленные ранее репозитории скорее всего уже хранят все образцы, которые можно найти в открытом доступе.

6.1.2 Модули и библиотеки

Благодаря развитию проектов с открытым исходным кодом появилось множество людей и компаний, публикующих свои результаты разработок в репозиториях версионного контроля, базах данных, на сайтах. Вместе с этим появилась не только возможность использования готовых открытых модулей и библиотек, но и выбора между ними в связи с их разнообразием. Для любых реализуемых программных комплексов пропадает надобность в написании собственной версии готовых алгоритмов, что упрощает и ускоряет разработку.

Для данной выпускной квалификационной работы нужно исследовать модули и библиотеки, реализующие машины факторизации и их расширение в виде учёта специфики полей. В связи с тем, что таких работ много, то после анализа нужно выбрать лучшие и наиболее удобные из них и сравнить на основе тестирования построенных на них систем обнаружения.

В оригинальной работе [38], спустя 2 года представленной концепции машин факторизации была представлена их реализация libFM на языке C++. В ней доступна оптимизация методом стохастического градиентного спуска (SGD) и чередующихся наименьших квадратов (ALS), а также байесовский вывод с использованием цепей Маркова Монте-Карло (MCMC).

libFFM – реализация машин факторизации с учётом специфики полей, включающая в себя возможность раннего прекращения обучения, так как модель

склонна к переобучению. Библиотека так же написана на C++, допускает распараллеливание, но при этом становится недетерминированной, и использует модуль SSE для быстрых вычислений.

Наиболее удобными являются Python реализации ввиду простоты языка. Такими модулями являются:

1. pyFM – это минималистичная реализация машин факторизации с использованием Cython. Предыдущая же версия использовала numpy. Интерфейс библиотеки схож с популярным для машинного обучения модулем scikit-learn.

2. fastFM – реализация машин факторизации, включающая как классификацию, так и регуляризацию. Как и оригинальная библиотека содержит в себе три решателя – SGD, ALS, MCMC. Связанная с модулем работа [39] описывает данную реализацию. Она так же построена на Cyton в связи с тем, что ядро модели написано на языке программирования C. Также в работе сказано, что обеспечиваются быстрые операции с разреженными матрицами и векторами, что также и упрощает работу с разделением памяти между C и Python. При экспериментах точность библиотеки была такой же, как и в libFM, однако по времени выполнения превосходила оригинальную библиотеку.

3. xLearn [40] – реализация линейной модели логистической регрессии, машин факторизации и их расширения с учётом специфики полей. В документации утверждается, что модуль работает в разы быстрее стандартных библиотек libFM и libFFM, а также обеспечивает простоту и масштабируемость. Кроме того, предоставляет оболочку для работы через командную строку и язык программирования R.

Кроме перечисленных модулей есть ещё множество реализаций, но все они предоставляют либо ту же, либо меньшую функциональность, поэтому не нуждаются в описании. Также, все перечисленные модули наиболее востребованы, согласно оценкам GitHub, поэтому их рассмотрение будет наиболее интересно и для того большинства разработчиков, использующих их.

6.2 Выбор наборов данных и их обработка

Теперь нужно сформировать достаточно большой набор данных для обучения и тестирования. Для этого был подан запрос на доступ в репозиторий вредоносных приложений VirusShare и спустя время получены данные для авторизации и возможность скачивания наборов.

Репозиторий VirusShare был выбран в связи с его распространённостью в использовании, благодаря чему можно будет сравнить результаты работы реализуемых систем обнаружения с другими трудами. Кроме того, репозиторий предоставляет обширный выбор наборов данных по годам их публикации. Минусом является то, что не все наборы есть возможность скачать ввиду их раздачи при помощи технологии Torrent и отсутствии раздающих. Таким образом был выбран набор данных VirusShare_Android_APK_2016, содержащий более 12 тысяч вредоносных Android приложений.

Так как требуется два набора приложений – вредоносные и безопасные, был дополнительно взят набор данных CICMalAnal2017 [41] из репозитория UNB. Он содержит более 10 тысяч образцов, 4000 из которых – вредоносные и 6500 – безопасные. Данные собраны из различных источников, в том числе большинство из безопасных приложений – из магазина Google Play, опубликованные с 2015 по 2017 года. Кроме того, авторами набора данных было установлено 5065 безопасных приложений на реальные устройства для их проверки на отсутствие вредоносных действий. В репозитории набор безопасных приложений разделён по годам, но для работы были взяты и объединены архивы всех годов.

Так как наборы приложений довольно большие и их разбор занимает много времени, кроме того, возможны исключительные случаи, когда работа внезапно прерывается, нужно организовать промежуточное сохранение результатов. В связи с тем, что в Python удобно организована работа с json форматом посредством одноимённого стандартного модуля, было решено сохранять данные именно в этом формате.

Формат и пример заполнения JSON файла представлен на Рисунке 4. Файл состоит из 5 основных полей:

1. all_features – список, содержащий имена всех характеристик безопасности разобранных приложений.
2. list_of_vectors – список, содержащий списки, представляющие бинарные вектора, соответствующие списку all_features. Если n-ая характеристика из all_features содержится в приложении, то на n-ой позиции вектора будет находиться 1, иначе – 0.
3. results – список, показывающий каким является приложение – вредоносным или безопасным. Если n-ый вектор списка vectors представляет вредоносное приложение, то на n-ой позиции списка results будет находиться -1, если же безопасное, то 1.
4. stage – этап, на котором было выполнено последнее сохранение (malware или benign).
5. num – количество обработанных приложений на соответствующем этапе.

Был написан код, как составляющая реализуемого классификатора, который считывает набор входных APK файлов и переводит их характеристики в список бинарных векторов. Для этого первым делом были сформированы два каталога с безопасными и вредоносными приложениями. Таким образом, сначала считываются имена приложений из каталогов, пути к которым записываются в коде в переменные datasets_folder (общая часть пути), benign_folder и malware_folder, и формируются соответствующие списки.

```
trust = sorted([f for f in listdir(datasets_folder + benign_folder) if isfile(join(datasets_folder + benign_folder, f))])
malware = sorted([f for f in listdir(datasets_folder + malware_folder) if isfile(join(datasets_folder + malware_folder, f))])
```

После этого, отдельно для вредоносных и безопасных приложений вызывается функция, отвечающая за перебор файлов.

```
get_features_from_files(trust, datasets_folder + be-
nign_folder)
```

В ней запускается цикл по всем полученным ранее именам файлов:

```
while i < len(listfiles):
        filename = listfiles[i]
        i += 1
        features = get_features(directory + '/' + filename)
        if not features:
            continue
        list_of_vectors.append(get_features_vector(features))
```

Здесь listfiles – переданный список имён приложений, который, образуя полный путь, передаётся в функцию get_features. Данная функция при помощи средств модуля Androguard декодирует APK файл, дизассемблирует исходные коды и возвращает нужные характеристики приложения путём вызова соответствующих функций. Кроме того, перед обработкой приложения файл APK проверяется на то, что он является ZIP файлом и организуется обработка исключений для корректного продолжения работы даже в случае проблем с APK файлом.

```
def get_features(apk_name):
    features = []
    if zipfile.is_zipfile(apk_name):
        try:
            apk, dalvik, analysis = AnalyzeAPK(apk_name)
            features += apk.get_permissions()
            features += apk.get_declared_permissions()
            features += apk.get_features()
            features += apk.get_libraries()
            features += apk.get_providers()
            features += apk.get_receivers()
            features += apk.get_requested_aosp_permissions()
            features += apk.get_uses_implied_permission_list()
            del apk
            del dalvik
            del analysis
        except Exception:
```

```
            print("Exception throws!")
    else:
        print("Bad zip file!")
    return features
```

Таким образом, при помощи модуля Androguard формируется список следующих характеристик:

– разрешения, объявленные в манифесте приложения;
– разрешения, запрашиваемые во время работы приложения;
– используемые аппаратные средства;
– используемые библиотеки;
– используемые поставщики контента;
– подписки на широковещательные сообщения;
– привилегированные разрешения системы Android;
– разрешения, выдаваемые как зависимости других разрешений или встроенные разрешения используемой версии SDK.

Если приложение было удачно обработано, то результат работы функции get_features подаётся на вход функции get_features_vector, которая, в свою очередь, кодирует полученные характеристики в бинарный вектор, соответствующий конкретному приложению.

Первым делом формируется список feature_vector, содержащий нули на всех позициях. Размер данного списка равен размеру списка all_features, который содержит все названия ранее считанных характеристик.

```
feature_vector = [0] * len(all_features)
```

Далее, каждую характеристику нужно разобрать так, чтобы получить название характеристики, не зависящее от конкретного пакета приложения.

```
feature = rawfeature[rawfeature.rfind('.')+1:]
```

Далее, если полученное имя характеристики содержится в списке all_features, то на соответствующей позиции feature_vector устанавливается 1.

```
if feature in all_features:
    feature_vector[all_features.index(feature)] = 1
```

Если характеристика не содержится в all_features, то она добавляется в all_features, а в конец каждого вектора, соответствующего приложению, закодированному ранее, добавляется элемент, содержащий 0.

```
else:
    all_features.append(feature)
    for vector in list_of_vectors:
        vector.append(0)
    feature_vector.append(1)
```

Результатом функции является закодированный вектор характеристик конкретного Android-приложения, который добавляется в результирующий список векторов list_of_vectors.

```
list_of_vectors.append(get_features_vector(features))
```

В результате работы кода, отвечающего за формирование обучающей выборки, получаем список list_of_vectors, содержащий обучающие вектора, а также список all_features, содержащий имена всех полученных ранее характеристик из обучающих приложений. Кроме того, формируется список results, в котором хранятся результаты эталонной классификации. Также, ввиду того, что обработка приложений для формирования обучающей выборки - самая трудоёмкая задача в контексте реализуемой системы (некоторые приложения могут обрабатываться около 10 минут и это время пропорционально размеру приложения), чтобы не обрабатывать уже закодированные ранее файлы снова, как было описано ранее, результаты записываются в файл result.json каждые dump_every_n обработанных приложений. Мной было выставлено значение в 5 приложений.

Общая схема кодирования приложений в вектора и сохранения их в виде JSON файла представлена на Рисунке 4.

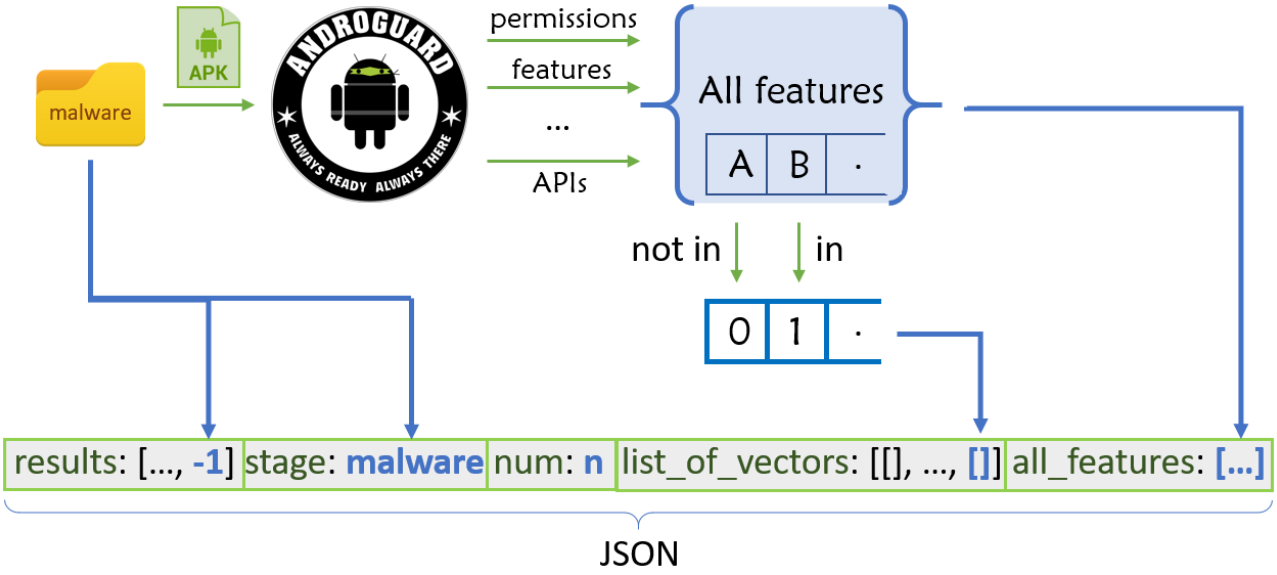

Рисунок 4 – Схема обработки Android приложений

Результаты обработки приложений наборов данных VirusShare и CICMalAnal2017 – соответствующие им JSON файлы были выложены в открытый репозиторий GitHub [42]. Наборы данных имели большое количество приложений, которые не смогли обработаться Andrguard'ом. Было пропущено около тысячи безопасных приложений и около трети вредоносных, в итоге JSON файлы содержат вектора 5642 безопасных и 8744 вредоносных приложений.

Формат JSON не является стандартным для наборов данных в машинном обучении, так как чаще всего используется формат CSV. Однако, для некоторых реализаций алгоритмов формат JSON является более удобным, поэтому может быть полезен исследователям, ведущим работы с такими реализации.

6.3 Выбор библиотек для реализации систем

Теперь, когда обучающая выборка сформирована, нужно выбрать модули для задачи обнаружения вредоносных Android приложений и использовать их для реализации систем обнаружения и их дальнейшего сравнения.

Были выбраны два модуля – fastFM и xLearn в связи с тем, что их разработчики в соответствующих источниках утверждают, что эти библиотеки работают быстрее, чем libFM, но насколько они отличаются по скорости друг от друга

– неизвестно. Кроме того, оба модуля многофункциональные – могут выполнять различные задачи, а xLearn предоставляет не только машины факторизации, но и другие модели и возможности. В связи с этим, данные библиотеки являются наиболее популярными в качестве пакетов для Python.

6.3.1 FastFM

FastFM предоставляет процедуры стохастического градиентного спуска (SGD), координатного спуска (CD), а также метод Монте-Карло на основе марковских цепей (MCMC). Модуль можно использовать для задач регрессии, классификации и ранжирования.

В теории MCMC должна лучше справляться с задачей классификации вредоносных приложений, однако, на практике, для большинства задач MCMC и SGD справляются одинаково эффективно. Поэтому для данной работы был выбран метод стохастического градиентного спуска, а кроме того, он имеет наиболее быстрое прогнозирование и эффективно работает с большими наборами данных, что важно для решаемой задачи.

Реализация системы обнаружения вредоносных Android приложений при помощи fastFM начинается со считывания файла results.json, формат которого был описан в пункте 6.2. Данный файл содержит результаты обработки приложений – бинарные вектора для обучения.

```
if isfile('results.json'):
        with open('results.json') as json_file:
            json_res = json.load(json_file)
            all_features = json_res['all_features']
            list_of_vectors = json_res['vectors']
            results = json_res['results']
            del json_res
```

FastFM для обучения принимает бинарные вектора, отвечающие за содержание в приложениях соответствующих характеристик безопасности, в виде сжатой разреженной матрицы пакета scipy. Для этого список векторов преобразуется в данный формат с помощью функции sparse.csc_matrix.

```
matrix = scipy.sparse.csc_matrix(list_of_vectors)
```

Вектор, описывающий класс, к которому относится соответствующее приложение, должен иметь вид массива numpy, поэтому список results преобразуется в массив функцией np.asarray.

```
result_array = np.asarray(results)
```

После этого вызывается функция training, в которой начала создаётся объект для модели машины факторизации с соответствующими параметрами, а затем обучается на созданном наборе данных.

```
fm = sgd.FMClassification(n_iter=1000, init_stdev=0.1, l2_reg_w=0, l2_reg_V=0, rank=2, step_size=0.1)
fm.fit(matrix, result_array)
```

Допускается следующий набор параметров машины факторизации:

– n_iter – количество итераций алгоритма;

– init_stdev – инициализация параметра stdev;

– l2_reg_w – штрафной вес в L2 регуляризации для линейных коэффициентов;

– l2_reg_V – штрафной вес в L2 регуляризации для парных коэффициентов;

– rank – ранг факторизации;

– step_size – размер шага для SGD.

После обучения сети можно получить все содержащиеся в ней веса:

– w0_ – байесовский терм;

– w_ – линейные коэффициенты;

– V_ – коэффициенты матрицы факторизации второго порядка.

Совокупность данных параметров описывает обученную модель машины факторизации. Таким образом, сохраняя данные параметры, сохраняется обученная модель, эти параметры можно будет использовать для дальнейшего восстановления модели без надобности обучения заново.

После обучения сети её можно использовать для классификации. Метод predict используется для прогноза классификации. Получая на вход матрицу,

содержащую вектор конкретного тестируемого приложения, описанного ранее вида, метод возвращает 1, если приложение безопасно, -1, если приложение вредоносное.

Метод predict_proba получает на вход всё ту же матрицу, а возвращает вероятность того, что классификация была выполнена верно.

Для тестирования модели была вынесена часть приложений из исходных наборов в отдельную папку. При тестировании модели эти приложения обрабатываются так же, как и при формировании обучающей выборки, за исключением того, что если обнаруживаются новые характеристики безопасности, то они никак не влияют на бинарный вектор приложения и результат классификации.

6.3.2 xLearn

xLearn поддерживает оптимизацию стохастическим градиентным спуском (SGD), адаптивным градиентным методом (Adagrad) и метод следующего за регуляризирующим лидером (FTRL).

По сравнению с традиционным методом SGD, Adagrad адаптирует скорость обучения к характеристикам, выполняя больше обновлений для редко появляющихся и меньше обновлений для часто появляющихся характеристик. По этой причине он хорошо подходит для работы с разреженными данными.

FTRL (Follow-the-Regularized-Leader) - метод, который широко используется для крупномасштабных разреженных задач. Однако для использования FTRL необходимо настраивать большее количество гиперпараметров по сравнению с SGD и Adagrad.

В данном случае был выбран метод Adagrad в связи с тем, что он лучше проявляет себя при работе с разреженными данными, чем SGD, и при этом не требует большого количества гиперпараметров, как FTRL.

xLearn использует форматы libsvm или CSV для стандартной модели машин факторизации и libffm для модели с учётом специфики полей. В связи с этим сформированный набор данных был переформатирован из JSON формата в libsvm. Кроме того, для машин факторизации с учётом специфики полей были

выделены поля характеристик обработанных ранее приложений, соответствующие типам выводимых Androguard характеристик, и на основе этого сформированы файлы для обучения и тестирования, содержащие те же данные, что и для модуля fastFM.

xLearn предоставляет более простой интерфейс создания и переиспользования моделей. Так, модель стандартной машины факторизации создаётся и тренируется согласно Рисунку 5, соответствующему следующему коду:

```
fm_model = xl.create_fm()
fm_model.setTrain("./results.libsvm")
fm_model.setValidate("./testing.libsvm")
param = {'task':'binary', 'lr':0.2, 'lambda':0.002, 'k':4, 'metric': 'acc'}
fm_model.fit(param, "./model.out")
```

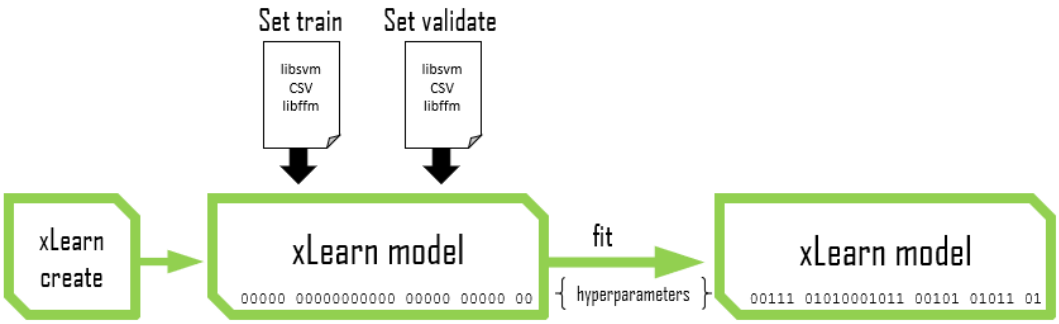

Рисунок 5 – Схема работы модели, построенной на основе xLearn

Для модели может настраиваться большее количество параметров, но наиболее интересующие – это:

– lr – скорость обучения;

– lambda – параметр L2 регуляризации;

– k – ранг факторизации.

Для расширенной модели с учётом специфики полей меняются только входные файлы, а при создании модели используется метод create_ffm вместо create_fm.

6.4 Выбор оптимальных параметров моделей

Показатели, относительно которых оцениваются и сравниваются модели машинного обучения называются метриками. Все они рассчитываются относительно составляющих матрицы ошибок, которая представлена в виде Таблицы 4, где $\bar{y}$ – результат классификации от модели, y – истинная классификация объекта.

Таблица 4 – Матрица ошибок

	$y = 1$	$y = 0$
$\bar{y} = 1$	True Positive (TP)	False Positive (FP)
$\bar{y} = 0$	False Negative (FN)	True Negative (TN)

Основные используемые метрики и их вычисление:

– $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ – доля верно классифицированных моделью объектов;

– $precision = \frac{TP}{TP+FP}$ – доля истинно положительных объектов от их положительно классифицированного общего количества;

– $recall = \frac{TP}{TP+FN}$ – доля верно классифицированных положительных объектов от их общего количества;

– $F1 = 2 * \frac{precision*recall}{precision+recall}$ – среднее гармоническое precision и recall;

– AUC – площадь под кривой ошибок $TPR = \frac{TP}{TP+FN}$ и $FPR = \frac{FP}{FP+TN}$.

Для системы обнаружения вредоносных приложений наиболее важна точность в связи с использованием в магазинах приложений и в антивирусах, которые должны, в том числе для своей репутации, с одинаковой точностью пропускать безопасные и блокировать вредоносные приложения. Кроме того, данная метрика отлично подходит для систем обнаружения в связи с возможностью её тестирования на большом количестве данных, содержащих классы безопасных и вредоносных примеров одинаково объёма.

Для достижения наивысших показателей метрик, в данном случае – точности, для моделей должны быть подобраны соответствующие параметры. Для этого был использован метод последовательного перебора параметров с последующим обобщением путём построения графиков для каждой из моделей с целью максимизации точности определения класса примеров. Перед этим был разделён исходный набор данных на наборы для обучения и для тестирования, и после этого обучена модель и получены результаты перебора параметров на данных для тестирования.

В результате были получены следующие наборы параметров:

1. FastFM – машина факторизации.
   1) $rank = 150\ 000$;
   2) $n\_iter\ = 3000$;
   3) $step\_size = 0.007$.
2. xLearn – машина факторизации.
   1) $k = 5$;
   2) $lambda = 0.00001$;
   3) $lr = 2$.
3. xLearn – машина факторизации с учётом специфики полей.
   1) $k = 5$;
   2) $lambda = 0.00001$;
   3) $lr = 2$.

На Рисунках 6–8 представлены графики зависимости точности моделей машинного обучения от соответствующих, использующихся в них, параметров.

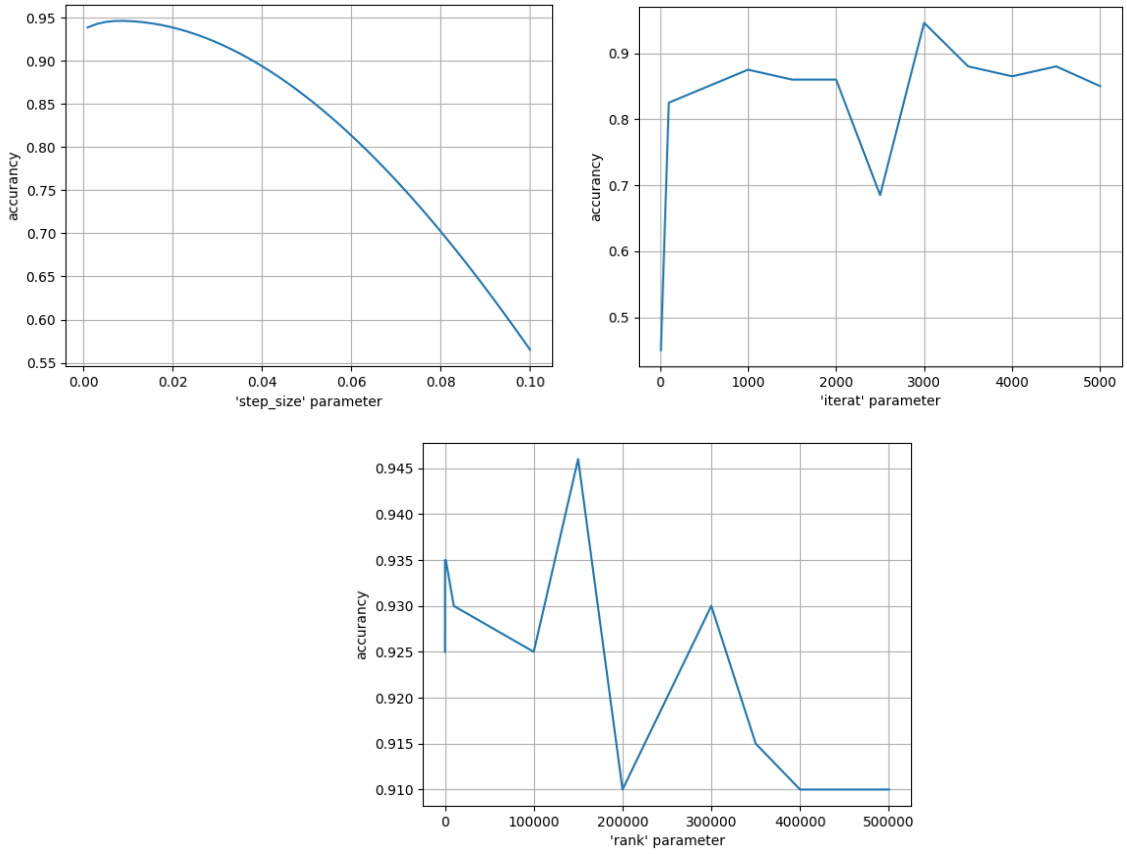

Рисунок 6 – Зависимость точности модели fastFM от значений её параметров

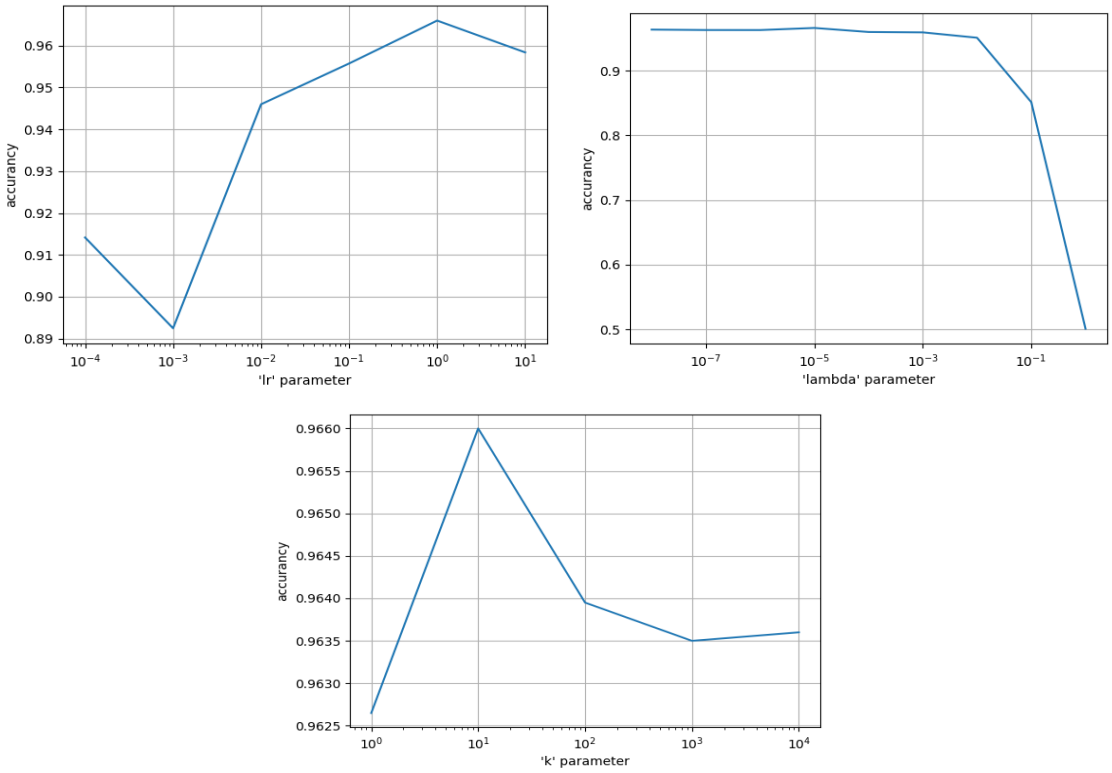

Рисунок 7 – Зависимость точности модели машин факторизации xLearn от значений её параметров

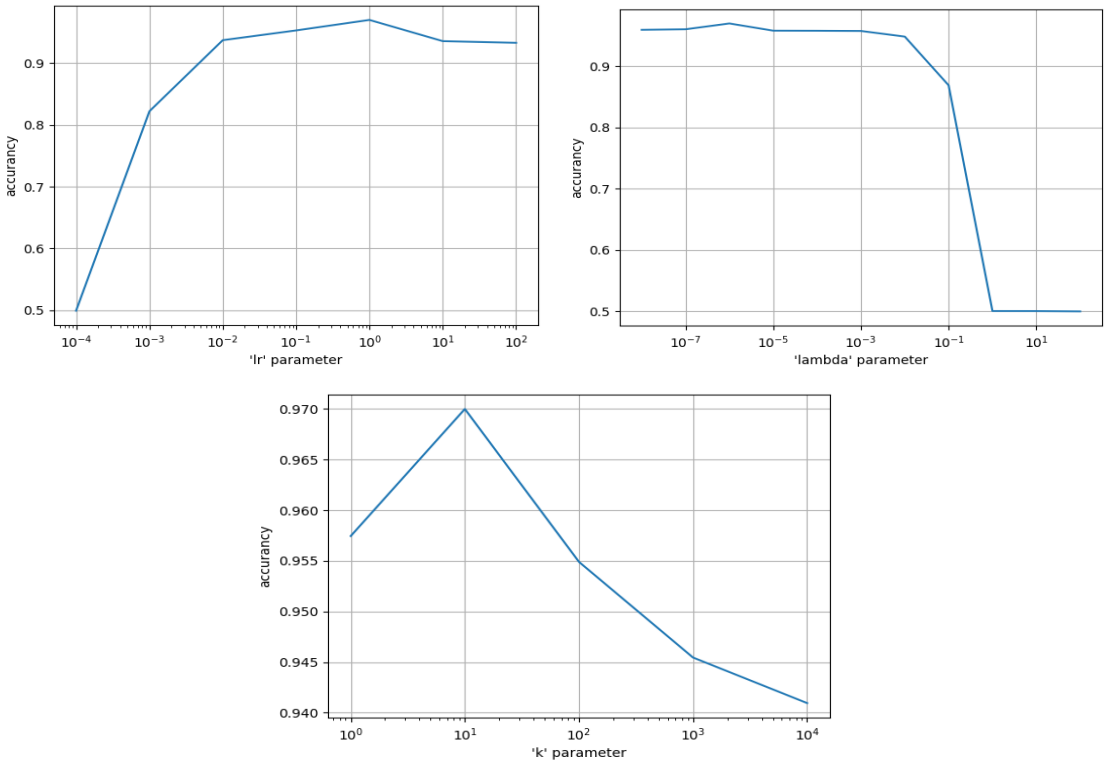

Рисунок 8 – Зависимость точности модели машин факторизации с учётом специфики полей xLearn от значений её параметров

* * *

Таким образом, в данном разделе были реализованы модели машинного обучения, основой для которых стали машины факторизации и их расширение с учётом специфики полей. Были сформированы наборы данных для обучения моделей и их дальнейшего тестирования. Перебором параметров были получены графики, показывающие зависимость точности обнаружения вредоносных приложений от значений параметров. Максимумы на графиках показывают наивысшую полученную точность моделей. Для fastFM это 94,6%, для xLearn, основанной на машине факторизации – 96,6%, основанной на расширении с учётом специфики полей – 97%.

7 СРАВНЕНИЕ И ОЦЕНКА РЕЗУЛЬТАТОВ

Два критерия, по которым можно сравнить реализованные модели – это скорость обучения и точность классификации. Кроме того, в связи с распространённостью набора данных VirusShare, есть возможность сравнить точность классификации полученных моделей с другими моделями, так же использующими этот набор данных.

В пункте 6.4 раздела 6 настоящей работы были определены наилучшие параметры для моделей fastFM и xLearn относительно точности классификации. В Таблице 5 показаны максимально достигнутые точности классификации для каждой из моделей.

Таблица 5 – Точность классификации обученных моделей

	fastFM	xLearn (машина факторизации)	xLearn (с учётом специфики полей)
Точность	94,6%	96,6%	97%

Полученная точность является достаточно высокой и превосходит большинство работ, направленных на решение задачи обнаружения вредоносных Android приложений и использующих набор данных VirusShare. Так, в работах [43-45] авторам удалось достичь только от 92% до 95,5% точности классификации. В ранее рассмотренной работе [29] была получена сопоставимая с данной работой точность – 96,9%. Но есть работы и превышающие полученную точность классификации – например, работа [46], в которой точность составила 97,66%. Сравнительная диаграмма точности различных работ приведена на Рисунке 9.

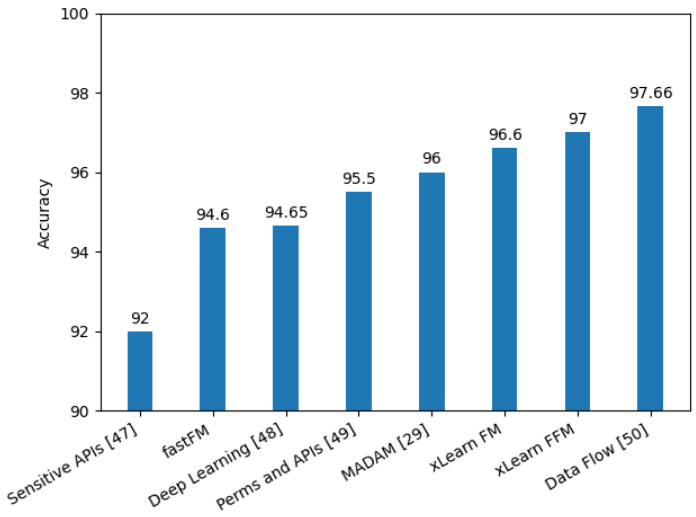

Рисунок 9 – Сравнительная диаграмма точности классификации

В документации к обоим пакетам – fastFM и xLearn, указано, что скорость их работы выше скорости работы традиционной модели libFM, поэтому было интересно сравнить оба этих модуля друг с другом для определения наиболее быстрого из них.

Время работы моделей включает в себя время обучения и тестирования. Для того, чтобы модели были в равных условиях, для них были выставлены сопоставимые параметры. Модели обучались 10 эпох для получения усреднённых результатов для одной эпохи. Зависимость времени работы моделей от ранга факторизации представлена на Рисунке 10.

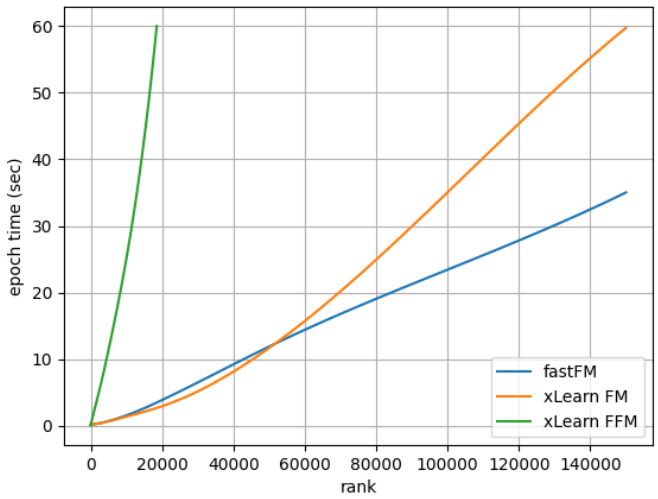

Рисунок 10 – Зависимость времени работы моделей от ранга факторизации

Таким образом, xLearn на основе машин факторизации работает немного быстрее с небольшим рангом факторизации, однако при большом ранге fastFM обрабатывает модель заметно быстрее.

Отдельно нужно рассматривать xLearn, основанную на учёте специфики полей. В связи с тем, что модель обучает большее количество скрытых векторов, время её работы заметно больше при любом ранге факторизации.

* * *

Таким образом, в данном разделе была проведена оценка точности и времени работы каждой реализации машин факторизации. Было установлено, что реализация стандартной модели в модуле fastFM быстрее производит обучение и тестирование модели при большом ранге факторизации (заметная разница появляется при ранге равном 100 000 и растёт с увеличением ранга). Реализация же xLearn отлично проявила себя в точности классификации за счёт адаптивного обучения на каждой из эпох и превосходит fastFM при любом ранге факторизации. Минусом xLearn является то, что она недетерминированная за счёт параллелизации при обучении. Параллелизацию можно отключить для моделей, но в этом случае она станет ещё медленнее.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе были исследованы методы машинного обучения, построенные на факторизации матриц – машины факторизации и их расширение, учитывающее специфику полей. Были проанализированы существующие работы, реализации и наборы данных для построения систем обнаружения вредоносных Android приложений.

В практической части были обработаны датасеты VirusShare и CICMalAnal2017, для них в формате JSON сформированы наборы векторов, представляющих Android приложения, а результаты выложены в общедоступный репозиторий [42]. Были реализованы, обучены и протестированы модели машинного обучения на основе пакетов fastFM и xLearn для языка Python.

При тестировании моделей были получены высокие показатели точности классификации Android приложений. Для машин факторизации, построенных на основе пакета fastFM точность достигла 94,6%, на основе xLearn – 96,6%. Для расширенной модели с учётом специфики полей, построенной на основе xLearn точность достигла 97%. Таким образом, полученные на практике результаты подтверждают теоретическое повышение точности путём применения алгоритма машин факторизации с учётом специфики полей для обнаружения вредоносных Android приложений.

Кроме того, была проанализирована скорость работы fastFM и xLearn. Согласно результатам для стандартной модели машин факторизации, xLearn работал немного быстрее при небольшом ранге факторизации и значительно уступал при достаточно большом (приблизительно от ранга равного 100 000). xLearn на основе расширенной модели работал намного дольше в связи с большим количеством скрытых векторов.

В качестве дальнейшей работы может выступить исследование зависимости точности классификации приложений от количества выделяемых полей и содержащихся в них характеристик. Кроме того, можно применять алгоритмы оптимизации гиперпараметров для получения наилучших результатов без

надобности полного перебора параметров модели машинного обучения. Также стоит рассмотреть зависимость точности классификации от обучающего набора, то есть его размера и входящих в него приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Number of smartphone users worldwide from 2016 to 2026. [Электронный ресурс]. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. – (дата обращения: 03.06.2021).

2. Number of smartphones sold to end users worldwide from 2007 to 2021. [Электронный ресурс]. URL: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. – (дата обращения: 10.05.2021).

3. Mobile operating systems' market share worldwide from January 2012 to January 2021. [Электронный ресурс]. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. – (дата обращения: 10.05.2021).

4. Number of available applications in the Google Play Store from December 2009 to December 2020. [Электронный ресурс]. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. – (дата обращения: 10.05.2021).

5. 10 Google Play Apps Found Containing Banking Malware. [Электронный ресурс]. URL: https://www.infosecurity-magazine.com/news/ten-google-play-apps-banking/. – (дата обращения: 10.05.2021).

6. Google: Android Malware Threat is Vastly Exaggerated. [Электронный ресурс]. URL: https://www.infosecurity-magazine.com/news/google-android-malware-threat-is-vastly/. – (дата обращения: 10.05.2021).

7. AV-TEST Malware. [Электронный ресурс]. URL: https://www.av-test.org/en/statistics/malware/. – (дата обращения: 10.05.2021).

8. Павленко Е. Ю., Дремов А. С. Обнаружение вредоносных участков кода Android-приложений на основе анализа графов потоков управления и графов потоков данных //Проблемы информационной безопасности. Компьютерные системы. – 2017. – №. 2. – С. 109-126.

9. Павленко Е. Ю., Игнатьев Г. Ю., Зегжда П. Д. Статический анализ безопасности Android-приложений //Проблемы информационной безопасности. Компьютерные системы. – 2017. – №. 4. – С. 73-79.

10. Venugopal D., Hu G. Efficient signature based malware detection on mobile devices //Mobile Information Systems. – 2008. – Т. 4. – №. 1. – С. 33-49.

11. Павленко Е. Ю., Ярмак А. В., Москвин Д. А. Применение методов кластеризации для анализа безопасности Android-приложений //Проблемы информационной безопасности. Компьютерные системы. – 2016. – №. 3. – С. 119-126.

12. Chen T. et al. TinyDroid: a lightweight and efficient model for Android malware detection and classification //Mobile information systems. – 2018. – Т. 2018.

13. Павленко Е. Ю., Игнатьев Г. Ю. Выявление вредоносных Android-приложений с использованием сверточной нейронной сети //Проблемы информационной безопасности. Компьютерные системы. – 2018. – №. 3. – С. 107-119.

14. 13. Павленко Е. Ю., Суслов С. М. Выявление вредоносных приложений для операционной системы Android с использованием капсульной нейронной сети //Проблемы информационной безопасности. Компьютерные системы. – 2019. – №. 1. – С. 100-111.

15. Wu D. J. et al. Droidmat: Android malware detection through manifest and api calls tracing //2012 Seventh Asia Joint Conference on Information Security. – IEEE, 2012. – С. 62-69.

16. Arp D. et al. Drebin: Effective and explainable detection of android malware in your pocket //Ndss. – 2014. – Т. 14. – С. 23-26.

17. Yang W. et al. Appcontext: Differentiating malicious and benign mobile app behaviors using context //2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. – IEEE, 2015. – Т. 1. – С. 303-313.

18. Gascon H. et al. Structural detection of android malware using embedded call graphs //Proceedings of the 2013 ACM workshop on Artificial intelligence and security. – 2013. – С. 45-54.

19. Arzt S. et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps //Acm Sigplan Notices. – 2014. – Т. 49. – №. 6. – С. 259-269.

20. Octeau D. et al. Composite constant propagation: Application to android inter-component communication analysis //2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. – IEEE, 2015. – Т. 1. – С. 77-88.

21. Malware categories. [Электронный ресурс]. URL: https://developers.google.com/android/play-protect/phacategories. – (дата обращения: 12.05.2021).

22. Schmidt A. D. et al. Static analysis of executables for collaborative malware detection on android //2009 IEEE International Conference on Communications. – IEEE, 2009. – С. 1-5.

23. Schmidt A. D. et al. Enhancing security of linux-based android devices //Proceedings of 15th International Linux Kongress. – Lehmann, 2008. – С. 1-16.

24. Bläsing T. et al. An android application sandbox system for suspicious software detection //2010 5th International Conference on Malicious and Unwanted Software. – IEEE, 2010. – С. 55-62.

25. Burguera I., Zurutuza U., Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android //Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. – 2011. – С. 15-26.

26. Shabtai A. et al. “Andromaly”: a behavioral malware detection framework for android devices //Journal of Intelligent Information Systems. – 2012. – Т. 38. – №. 1. – С. 161-190.

27. Zhou Y., Jiang X. Dissecting android malware: Characterization and evolution //2012 IEEE symposium on security and privacy. – IEEE, 2012. – С. 95-109.

28. Sahs J., Khan L. A machine learning approach to android malware detection //2012 European Intelligence and Security Informatics Conference. – IEEE, 2012. – С. 141-147.

29. Saracino A. et al. Madam: Effective and efficient behavior-based android malware detection and prevention //IEEE Transactions on Dependable and Secure Computing. – 2016. – Т. 15. – №. 1. – С. 83-97.

30. Li C. et al. Android malware detection based on factorization machine //IEEE Access. – 2019. – Т. 7. – С. 184008-184019.

31. Arora A., Peddoju S. K., Conti M. Permpair: Android malware detection using permission pairs //IEEE Transactions on Information Forensics and Security. – 2019. – Т. 15. – С. 1968-1982.

32. Mahindru A., Sangal A. L. MLDroid—framework for Android malware detection using machine learning techniques //Neural Computing and Applications. – 2021. – Т. 33. – №. 10. – С. 5183-5240.

33. Rendle S. Factorization machines //2010 IEEE International Conference on Data Mining. – IEEE, 2010. – С. 995-1000.

34. Rendle S., Schmidt-Thieme L. Pairwise interaction tensor factorization for personalized tag recommendation //Proceedings of the third ACM international conference on Web search and data mining. – 2010. – С. 81-90.

35. Juan Y. et al. Field-aware factorization machines for CTR prediction //Proceedings of the 10th ACM conference on recommender systems. – 2016. – С. 43-50.

36. Li Y. et al. Android malware clustering through malicious payload mining //International symposium on research in attacks, intrusions, and defenses. – Springer, Cham, 2017. – С. 192-214.

37. Allix K. et al. Androzoo: Collecting millions of android apps for the research community //2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). – IEEE, 2016. – С. 468-471.

38. Rendle S. Factorization machines with libfm //ACM Transactions on Intelligent Systems and Technology (TIST). – 2012. – Т. 3. – №. 3. – С. 1-22.

39. Bayer I. fastfm: A library for factorization machines //The Journal of Machine Learning Research. – 2016. – Т. 17. – №. 1. – С. 6393-6397.

40. Get Started with xLearn ! [Электронный ресурс]. URL: https://xlearn-doc.readthedocs.io/en/latest/python_api/index.html. – (дата обращения: 15.05.2021).

41. Lashkari A. H. et al. Toward developing a systematic approach to generate benchmark android malware datasets and classification //2018 International Carnahan Conference on Security Technology (ICCST). – IEEE, 2018. – С. 1-7.

42. APK datasets for machine learning. [Электронный ресурс]. URL: https://github.com/maxherobrine/android_apk_datasets/. – (дата обращения: 10.05.2021).

43. Zhao C. et al. Quick and accurate android malware detection based on sensitive APIs //2018 IEEE International Conference on Smart Internet of Things (SmartIoT). – IEEE, 2018. – С. 143-148.

44. Sandeep H. R. Static analysis of android malware detection using deep learning //2019 International Conference on Intelligent Computing and Control Systems (ICCS). – IEEE, 2019. – С. 841-845.

45. Zhao C., Wang C., Zheng W. Android malware detection based on sensitive permissions and APIs //International Conference on Security and Privacy in New Computing Environments. – Springer, Cham, 2019. – С. 105-113.

46. Wu S. et al. Effective detection of android malware based on the usage of data flow APIs and machine learning //Information and software technology. – 2016. – Т. 75. – С. 17-25.