УДК 004.056.5
doi:10.31799/1684-8853-2022-4-44-57

Научные статьи
Articles

Применение алгоритмов машинного обучения для обнаружения вредоносных программ в операционной системе Windows с помощью PE-заголовка

Д. Ч. Ле*[а], *канд. техн. наук, преподаватель, orcid.org/0000-0003-3735-0314, letranduc@dut.udn.vn
М. Х. Фам*[б], *инженер, orcid.org/0000-0002-2250-9428
Ч. З. Динь*[в], *канд. техн. наук, преподаватель, orcid.org/0000-0002-9993-9792
Х. Ф. До*[г], *магистр, преподаватель, orcid.org/0000-0003-0645-0021
[а]*Университет Дананга — Университет науки и техники, факультет информационных технологий, 54 Nguyen Luong Bang, Дананг, 550000, Вьетнам*
[б]*Отдел исследований и разработок, Viettel Business Solutions Корпорация, Дананг, Вьетнам*
[в]*Технологический институт почты и телекоммуникаций, 122 Hoang Quoc Viet, Ханой, Вьетнам*
[г]*Университет архитектуры Дананга, 566 Nui Thanh, Дананг, Вьетнам*

Введение: *быстрый рост числа вредоносных программ и их злонамеренное использование приводят к значительным финансовым потерям. Исследователи заинтересованы в применении методов машинного обучения для решения задачи обнаружения вредоносных программ. Однако в силу разнообразия алгоритмов машинного обучения каждый имеет свой подход в определенной ситуации.* ***Цель:*** *применить методы машинного обучения для обнаружения вредоносных программ в операционной системе Windows с использованием компонентов переносимого исполняемого (Portable Executable) заголовка; сравнить шесть алгоритмов машинного обучения по нескольким критериям.* ***Результаты:*** *сравнение различных алгоритмов, таких как случайный лес (Random Forest), дерево принятия решений (Decision Tree), наивный байесовский алгоритм (Naive Bayes), метод опорных векторов (Support Vector Machine), многослойный перцептрон (Multilayer Perceptron), метод k-ближайших соседей (k-Nearest Neighbors) с большим набором данных, показало, что применение алгоритмов случайный лес, дерево принятия решений, метод k-ближайших соседей и многослойный перцептрон позволяет довести вероятность обнаружения вредоносных программ до высокой точности (> 98 %). Особенно алгоритм случайный лес очень хорошо подходит для применения в средствах обнаружения вредоносного программного обеспечения на операционной системе Windows. Наивный байесовский алгоритм также имеет высокий показатель точности (> 96 %) и быстрое время обработки. Поэтому мы можем рассматривать возможность использовать наивный байесовский алгоритм в качестве альтернативного.*

Ключевые слова *— вредоносная программа, алгоритм машинного обучения, PE-заголовок, Windows.*

Для цитирования: Ле Д. Ч., Фам М. Х., Динь Ч. З., До Х. Ф. Применение алгоритмов машинного обучения для обнаружения вредоносных программ в операционной системе Windows с помощью PE-заголовка. *Информационно-управляющие системы*, 2022, № 4, с. 44–57. doi:10.31799/1684-8853-2022-4-44-57

For citation: Le D. T., Pham M. H., Dinh T. D., Do H. P. Applying machine learning algorithms for PE-header-based malware detection on the Windows operating system. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2022, no. 4, pp. 44–57 (In Russian). doi:10.31799/1684-8853-2022-4-44-57

Введение

В настоящее время с быстрым развитием Интернета компьютерные приложения и системное программное обеспечение (ПО) постоянно меняются и развиваются. Это сопровождается появлением различных типов вредоносных программ [1]. Сегодня вредоносные программы становятся все более разнообразными, сложными и опасными [2].

Вредоносные программы являются одной из прямых угроз информационной безопасности. Эти вредоносные файлы распространяются незаметно и быстро различными передовыми методами, такими как механизмы инъекций, антивиртуальная машина, антиотладка, упаковщик, шифрование и механизмы настойчивости [3, 4].

Вредоносные программы идентифицируются и классифицируются по их вредоносной цели или поведению [5]. Например, различают такие типы вредоносных программ, как вирусы, трояны, программы-вымогатели, бэкдоры, рекламные программные обеспечения и т. д.

В настоящее время существует много инструментов для анализа вредоносных программ: *PEStudio*, *CFF Explorer*, *DetecItEasy*, *PPEE*, *IDA*, *Ghidra*, *OllyDbg*, *x64dbg*, *radare2*, *WinDbg*, *gdb* [6]. Их использует человек, осуществляющий анализ вредоносного ПО, но работать автономно, выявляя вредоносные программы, они не способны. Кроме того, каждый инструмент имеет свои преимущества и недостатки, а результат в основном зависит от опыта человека, проводящего анализ. Можно использовать методы статического

анализа вредоносных программ (*static malware analysis*), чтобы найти некоторые важные функции и спрогнозировать поведение подозрительных файлов, но это требует много времени. Также можно использовать динамический анализ в режиме реального времени, когда подозрительный файл выполняется в виртуальной среде, а по признакам при запуске файла в операционной системе (ОС) можно узнать, является ли он вредоносным или нет. Однако у динамического анализа много недостатков, например: для каждого типа вредоносного ПО требуется соответствующая среда анализа; легко обнаруживается функцией антивиртуальной машины вредоносного ПО и т. д.

При большом количестве вредоносных программ вышеперечисленные методы сложны для реализации. Для того чтобы сократить время анализа, необходимо иметь автоматическую систему классификации вредоносных программ. В этой статье мы предлагаем использовать алгоритмы машинного обучения для обнаружения вредоносных программ в ОС Windows. В частности, мы фокусируемся на формате PE-заголовка с существенными функциями/признаками, которые помогают обнаруживать много вредоносных ПО в Windows. Многие исследователи заинтересованы в применении методов машинного обучения для решения различных проблем в анализе вредоносных программ [7–9]. Однако из-за разнообразия каждый алгоритм машинного обучения имеет свой подход в определенной ситуации. Основное внимание в этой статье сосредоточено на применении машинного обучения для обнаружения вредоносных программ на Windows. Ко всему прочему, в статье представлено сравнение шести различных алгоритмов машинного обучения на основе нескольких критериев для расширения выбора и поиска оптимального решения. Этими алгоритмами являются случайный лес (*Random Forest*, *RF*), дерево принятия решений (*Decision Tree*, *DT*), наивный байесовский алгоритм (*Naive Bayes*, *NB*), метод опорных векторов (*Support Vector Machine*, *SVM*), многослойный перцептрон (*Multilayer Perceptron*, *MLP*), метод *k*-ближайших соседей (*k-Nearest Neighbors*, *k-NN*) [10], все они поддерживаются библиотеками Python Scikit-Learn [11]. Следует отметить, что многие категории вирусного ПО, нацеленного на ОС Windows, не имеют PE-заголовка. Однако рассмотрение этих типов выходит за рамки данной статьи.

Обзор литературы

В работе [12] авторы представили фреймворк для обнаружения вредоносных программ, направленный на получение как можно меньше ложноположительного результата, используя простую многоступенчатую комбинацию (каскад) различных версий алгоритма перцептрона. Перцептрон — это бинарный классификатор, который определяет принадлежность входных данных к определенному классу на основе набора весов с векторами признаков. Следует отметить, что алгоритмы, которые привели к лучшей точности, также дали наибольшее количество ложноположительного результата. Предлагаемая схема основана на статическом анализе, который может занять много времени.

В статье [13] Х. Ратхор и др. предложили глубокое обучение на основе алгоритма случайного леса с опкодом в качестве вектора признаков для обнаружения вредоносных программ. Они разобрали исполняемый файл с помощью *objdump* и извлекли опкод из файла. В качестве данных признаков был составлен и использован главный список опкодов, состоящий из 1600 уникальных опкодов. Однако в этой статье для обнаружения вредоносных программ были созданы только алгоритм случайного леса и различные слои глубокой нейронной сети.

Авторы в статье [14] использовали динамические характеристики и шесть алгоритмов машинного обучения, чтобы обнаружить варианты (на основе уязвимости нулевого дня) версий программ-вымогателей среди вредоносных и нормальных приложений. Экспериментальные результаты показали, что предлагаемый метод может обнаружить программы-вымогатели среди вредоносных программ и доброкачественных файлов.

Аша Джерлин и Маримутху Каруппиа [15] разработали эффективную систему для обнаружения вредоносных программ с использованием программируемых интерфейсов приложений (APIs) и классификации их типа как червей, вирусов и троянов или доброкачественных файлов. Предварительно набор входных данных обработан путем нормализации данных, затем его верхние и нижние границы оценивались во время выделения признаков. Результаты экспериментов показали, что предлагаемые методы хорошо работают при проверке с использованием таких критериев, как чувствительность алгоритма классификации (*True Positive Rate*, *TPR*), специфичность алгоритма классификации (*False Positive Rate*, *FPR*), точность (*precision*), полнота (*recall*), F1-мера (F1-*score*) и точность (*accuracy*).

Чандрасекар Рави и Р. Манохаран [16] также предложили систему обнаружения вредоносных программ, которая использует набор базовых функций интерфейсов программирования приложений ОС Windows API. Для моделирования вызовов API они использовали цепочку Маркова 3-го порядка. В данной статье используется классификация на основе ассоциативного майнинга, поскольку она дает более высокую точность

(*accuracy*) обнаружения. Вредоносные файлы в основном состояли из бэкдоров, червей и троянских коней, собранных из VXHeavens.

В работе [17] авторы попытались построить систему обнаружения вредоносных программ на основе их поведения. Модель объединила метод опорных векторов и метод главных компонент.

Джисин Сюй и др. предложили решение для обнаружения вредоносных программ с аппаратной поддержкой, которое использовало методы машинного обучения для мониторинга виртуальной памяти на наличие вредоносных доступов, вызванных вредоносными программами [18]. Новые аспекты фреймворка включают методы сбора и обобщения паттернов доступа к памяти для каждой функции/системного вызова и двухуровневую архитектуру классификации. Однако авторы обучали по одной модели для каждого приложения, что может быть довольно дорого.

Одним из наиболее часто используемых решений обнаружения вредоносных программ, особенно в антивирусном ПО, является обнаружение на основе сигнатур (*signature*). Недостаток этого решения заключается в том, что оно не может обнаружить новые вредоносные программы. Современные вредоносные программы имеют несколько полиморфных слоев, чтобы избежать обнаружения или автоматически обновить себя до новой версии. В работе [19] авторы предложили использовать n-граммы в качестве сигнатур файлов для обнаружения неизвестных вредоносных программ при сохранении низкого ложноположительного соотношения. Результаты показали, что n-граммовые сигнатуры обеспечивают эффективный способ обнаружения неизвестных вредоносных программ.

В отличие от обнаружения на основе сигнатур, эвристическое сканирование использует правила и (или) алгоритмы для поиска команд, которые могут указывать на намерение и подозрительные знаки. При использовании этого метода можно обнаружить вредоносные программы без наличия сигнатур. Большинство антивирусных программ используют эвристические методы.

В нашей работе мы предлагаем метод на основе выделенных признаков из PE-заголовка. Поля заголовков подозрительных файлов выделяются и сравниваются с полями заголовков доброкачественных файлов. Мы также попытаемся обобщить PE-признаки, которые оказывают значительное влияние на обнаружение вредоносных программ в Windows.

Следует отметить, что использование информации, извлеченной из PE-заголовка, для обнаружении вредоносных программ упоминалось некоторыми исследователями ранее. Например, Цзиньронг Бай и др. [20] фокусировались на библиотеках DLL (*Dynamic Link Library*) и API-функциях, извлеченных из анализа таблицы импорта (*Import Table*) в PE-заголовке. Однако библиотеки DLL и API-функции часто меняются в зависимости от характеристик каждого типа вредоносного ПО. Поэтому необходимо проводить более тщательный и избирательный анализ важной информации PE-заголовка в процессе изучения вредоносных программ.

Другой проект [21] сосредоточен на анализе и использовании информации PE-заголовка в комбинации с машинным обучением в обнаружении вредоносных программ. Однако авторы не указали, как PE-заголовок влияет на точность и эффективность моделей машинного обучения. В статье [22] авторы предложили использовать как исходное, так и выделяемое значение из PE-заголовка для создания входных данных для модели классификации, при этом они не указывают степень влияния выделяемых признаков на критерии, используемые для оценки моделей.

Основная цель нашего исследования заключается в применении алгоритмов машинного обучения случайный лес, дерево принятия решений, наивный байесовский алгоритм, метод опорных векторов, многослойный перцептрон, метод k-ближайших соседей на основе признаков, извлеченных из PE-заголовка, для обнаружения вредоносных программ на ОС Windows. Также мы проводим сравнение различных алгоритмов на основе метрик для оценки модели обнаружения вредоносных программ, включая точность, время обработки и эффективность. Обычно для моделей машинного обучения, используемых в задачах классификации, точность (*accuracy*) является основным критерием для оценки. Между тем время обработки — это время для модели, чтобы классифицировать файл ввода как вредоносную программу, и оно показывает способность модели к масштабированию при больших наборах данных. Кроме того, чтобы более объективно оценить алгоритмы и иметь возможность выбрать правильную модель, мы полагаемся на информацию из матрицы ошибок, точности, полноты, F1-меры и кривой соотношений правильного и ложного обнаружения (ROC-кривой, *Receiver Operating Characteristic curve*), показывающей производительность и эффективность классификационной модели алгоритмов.

Вредоносные программы в ОС Windows и формат PE-заголовка

В настоящее время вредоносные программы появляются во всех ОС, и все ОС восприимчивы к вредоносным программам. Относительно легко найти вредоносные программы для каждой

ОС, от Android, iOS до macOS, Linux и Windows. Среди этих ОС Windows является самой популярной операционной системой в мире, и, следовательно, она стала объектом большинства вредоносных атак. По данным отчета о безопасности AV Test за 2019/2020 г. (https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2019-2020.pdf), в 2019 г. было разработано 114 миллионов новых вредоносных программ, а 78,64 % всех атак было распространено на ОС Windows.

В большинстве предыдущих исследований, связанных с машинным обучением, авторы в основном извлекали признаки с помощью статического или динамического анализа из образцов вредоносных программ. При статическом анализе признаки извлекаются из последовательности байтов или значений опкода. В то же время при динамическом анализе признаки извлекаются путем запуска или эмуляции кода. Эти исследования в основном требуют выполнения всей программы и меньшего внимания к PE-заголовкам — одному из ключевых компонентов вредоносных программ, работающих в Windows-среде. Более того, запуск тысяч вредоносных программ для извлечения признаков, необходимых для процесса обучения, очень сложно реализовать.

Формат Portable Executable (PE) — это формат для выполнимых файлов, объектного кода, библиотек DLL и других файлов, используемых в 32-разрядных и 64-разрядных версиях ОС Windows. PE получен из спецификации Common Object File Format (COFF), которая также используется большинством выполнимых файлов Unix. Типы файлов, которые используют PE-формат, включают: .exe, .dll, .acm, .ax, .cpl, .drv, .efi, .mui, .ocx, .scr, .sys, .tsp.

PE-заголовок содержит информацию, которая считывается загрузчиком Windows при выполнении файла. После этого содержимое файла будет загружено из файла в память. С помощью PE-заголовка программа информирует ОС о своих требованиях к выполнению, поскольку он указывает, где исполняемый файл должен быть загружен в память. Он также указывает список библиотек/функций, на которые опирается приложение; адрес, с которого начинается выполнение, и двоичные ресурсы. Проверка PE-заголовка дает огромное количество информации о подозрительном файле и его функциях. Поэтому PE-заголовок имеет большое значение при обнаружении и анализе вредоносных программ. Основная структура PE-заголовка включает в себя:

— *DOS MZ Header*: проверяет, является ли файл действительным PE-файлом или нет (основные компоненты*:* e_magic, e_ifanew);

— *DOS Stub*: отображает предупреждение, если файл не может быть запущен в Windows;

— *PE File Header*: содержит информацию о файле (основные компоненты: signature, file header, optional header...);

— *Section Table*: содержит информацию о разделах, присутствующих в PE-файлах (основные компоненты: Name1, VirtualSize, SizeOfRawData, PointerToRawData, Characteristics);

— *Sections*: размещает основное содержимое файла, включая код, данные, ресурсы и другие исполняемые файлы (основные компоненты: .TEXT, .RDATA, .DATA, .RSRC).

В PE-файлах есть много признаков, но большинство из них не помогают отличить вредоносное ПО от доброкачественного. На основе наших эмпирических исследований и углубленного анализа признаков PE-заголовка мы извлекли 54 признака, отличающих доброкачественное ПО от вредоносных. Например: Machine, SizeOfOptionalHeader, Characteristics, MajorLinkerVersion, MinorLinkerVersion, SizeOfCode, AddressOfEntryPoint, ImageBase, Subsystem, DllCharacteristics и т. д. Полное описание этих признаков представлено в (https://docs.microsoft.com/ru-ru/windows/win32/debug/pe-format).

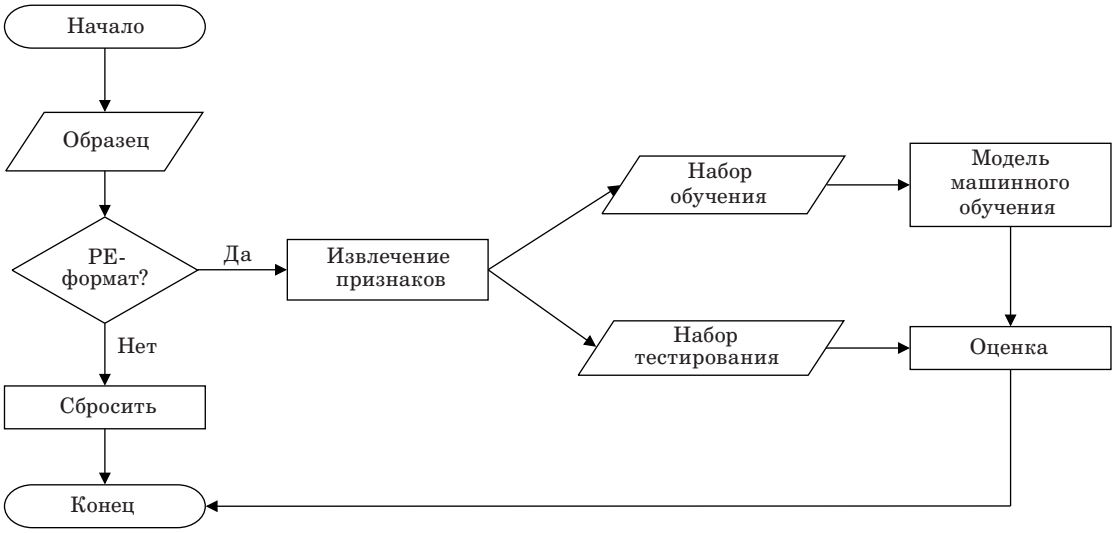
Экспериментальная часть

Рассмотрим этапы проведения экспериментов по оценке алгоритмов машинного обучения. На рис. 1 представлен алгоритм реализации.

Эксперимент проводился на Windows 10, 64-битный компьютер с процессорами Core AMD A8-4555M (1,6 ГГц, 4 ядра, 4 процессора), 8 ГБ оперативной памяти. Благодаря статическому анализу вычислительные затраты очень низки, поэтому все эксперименты проводятся на типичной системе конечных хостов.

Подготовка набора данных

Во-первых, необходимо получить набор данных для обучения алгоритмов. Для создания набора данных мы используем исполняемые файлы, зараженные вредоносными программами, предоставленные в Virusshare (https://virusshare.com/). Чтобы получить доступ к общей базе данных вредоносных программ, необходимо запросить учетную запись по электронной почте по адресу admin@virusshare.com. Набор данных вредоносных программ от Virusshare обычно не имеет расширения. Они будут переформатированы в исполняемые файлы автоматически с помощью команды *Windows Cmd*. В результате был использован набор данных, состоящий из 4761 файла, из которых 3816 были заражены



■ ***Рис. 1.*** Алгоритм реализации экспериментальной части работы
■ ***Fig. 1.*** Algorithm for the implementation of the experimental part

вредоносными программами, а 945 файлов были незараженными, доброкачественными. Следует отметить, что наш набор данных вредоносных программ имеет вредоносное ПО практически только для архитектуры x86-64. Результаты, полученные в эмпирическом процессе, могут отличаться для набора данных, содержащего вредоносное ПО для архитектуры x64.

Была использована *Python* — программа для извлечения признаков PE-заголовка. В этом эксперименте вредоносные программы помечаются как 1 (положительный ярлык) и 0 (отрицательный ярлык) для обычных файлов. После извлечения и маркировки всех файлов в допустимом формате получен набор данных для обучения и тестирования в моделях машинного обучения.

Предварительная переработка данных

Набор данных после сбора из PE-файлов был разделен на обучающий и тестовый наборы в соотношении 70 и 30 %. Набор обучения и набор тестирования были стандартизированы.

Нормализация — это хороший метод, который можно использовать, когда распределение данных неизвестно или оно не является гауссовым (колоколообразная кривая). Нормализация полезна, когда данные имеют различный масштаб, а используемый алгоритм не делает предположений о распределении данных, например *k-NN* и искусственные нейронные сети.

При нормализации данных среднее значение каждого признака равно нулю, а дисперсия равна 1. Для того чтобы формализовать данные, мы должны определить среднее и стандартное отклонение для распределения каждого признака. Затем вычесть из значения каждого признака среднее значение, далее разделить его на стандартное отклонение признака следующим образом:

$$x' = \frac{x - average(x)}{std(x)}, \qquad (1)$$

где x — исходный вектор признаков; $average(x)$ — среднее значение данного вектора признаков; $std(x)$ — стандартное отклонение.

Обучение модели

Нормализованные наборы данных будут обучаться в моделях машинного обучения. Прогнозируемые результаты моделей машинного обучения существенно зависят от гиперпараметров алгоритмов. Следовательно, очень важно выбрать экспериментальные значения для гиперпараметров в соответствии с каждой моделью, чтобы получить наилучшие результаты. Изначально модель обучается по конкретному алгоритму без внесения корректировки гиперпараметров (т. е. с использованием набора гиперпараметров по умолчанию в *scikit-learn*). После этого получены результаты с низкой точностью. По этим результатам было отмечено, что образцы всегда классифицируются в один класс. Следовательно, необходимо регулировать его параметры. Здесь использована *GridSearch* (в *scikit-learn*), чтобы регулировать гиперпараметры для этой модели. Далее эксперименты повторялись, и после настройки гиперпараметров получены лучшие результаты. Тот же процесс выполнен для других алгоритмов, и составлена таблица значений гиперпараметров (табл. 1).

■ ***Таблица 1.*** Гиперпараметры для каждой модели машинного обучения

■ ***Table 1.*** Hyperparameters for each machine learning model

Модель	Гиперпараметр	Описание
Дерево принятия решений (*DT*)	*criterion*	*Gini*
	max_depth	*3*
	min_samples_split	*2*
	min_samples_leaf	*1*
Случайный лес (*RF*)	*n_estimators*	*50*
	criterion	*Entropy*
	min_samples_split	*2*
	min_samples_leaf	*1*
Наивный байесовский алгоритм (*NB*)	*var_smoothing*	*1e-9*
Метод *k*-ближайших соседей (*k-NN*)	*n_neighbors*	*3*
	weights	*Uniform*
	leaf _size	*30*
	p	*2*
	metric	*Minkowski*
	algorithm	*Auto*
Метод опорных векторов (*SVM*)	*Kernel*	*Linear*
	Gamma	*scale*
	C	*1.0*
	Tol	*1e-3*
	max_iter	*-1*
Многослойный перцептрон (*MLP*)	*solver*	*Lbfgs*
	alpha	*1e-5*
	hidden_layer_sizes	*(5, 2)*

Оценка модели

После обучения моделей набор тестирования будет использоваться для проверки прогнозируемых результатов. С набором данных, состоящим только из небольшого количества образцов, мы можем проверить прогнозируемые результаты визуально. Однако, поскольку текущие наборы данных содержат значительное количество образцов, для обобщения результатов и оценки эффективности моделей классификации необходима матрица ошибок (*confusion matrix*). В данном эксперименте файлы с вредоносным ПО были помечены положительно (*positive*), а файлам без вредоносного ПО присваивались отрицательные метки (*negative*).

Матрица ошибок имеет следующую информацию.

TP (*True Positive* — правильно положительный): образцы с положительным ярлыком правильно классифицируются в положительный класс, т. е. общее количество вредоносных программ, которое прогнозируется как вредоносное ПО (истинное значение было «вредоносное», и модель прогнозировала «вредоносное»).

FN (*False Negative* — ложноотрицательный): положительно помеченные образцы ошибочно классифицируются в отрицательный класс, т. е. общее количество вредоносных программ, которые прогнозируются как доброкачественное ПО (истинное значение было «вредоносное», а модель прогнозировала «доброкачественное»).

FP (*False Positive* — ложноположительный): отрицательно помеченные образцы ошибочно классифицируются в положительный класс, т. е. общий доброкачественный файл, который предсказывается как вредоносное ПО (истинное значение было «доброкачественное», а модель прогнозировала «вредоносное»).

TN (*True Negative* — правильно отрицательный): отрицательно помеченные образцы правильно классифицируются в отрицательный класс, т. е. общий доброкачественный файл, который предсказывается как доброкачественное ПО (истинное значение было «доброкачественное», и модель прогнозировала «доброкачественное»).

Точность (или достоверность — *accuracy*) — это отношение правильно спрогнозированных наблюдений к общему количеству наблюдений. Точность модели показывает производительность модели и рассчитывается по следующей формуле:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \qquad (2)$$

Для оценки эффективности алгоритмов машинного обучения мы также можем использовать некоторые из следующих метрик: точность (*precision*), полноту (*recall*), ROC-кривую и F1-меру.

Точность (*precision* — *P*) является положительным прогностическим значением, которое может быть получено из матрицы ошибок. Она определяется как количество сделанных прогнозов, которые на самом деле являются правильными из всех прогнозов, основанных на положительном классе. Другими словами, это доля прогнозируемых вредоносных программ, которые на самом деле являются вредоносными программами. Точность является важной метрикой, поскольку неправильное предсказание вредоносного ПО опасно для реальных систем. Ее можно рассчитать по формуле

$$Precision = \frac{TP}{TP + FP}. \qquad (3)$$

Полнота (*recall — R*), также известна как чувствительность, или коэффициент *TP* (*TPR*), используется для определения соответствующих точек данных в процентах. Определяется как количество образцов положительного класса, которые были правильно спрогнозированы. Другими словами, это доля фактических вредоносных программ от прогнозируемых вредоносных программ. Значение полноты может быть рассчитано следующим образом:

$$Recall = \frac{TP}{TP + FN}. \quad (4)$$

ROC-кривая (*Receiver Operating Characteristic curve*) может быть использована для мультиклассовых классификаторов. Показатель *TP* (*TPR*) и показатель *FP* (*FPR*) классификатора используются для построения ROC-кривой. *FPR* также упоминается как ложные тревоги, определяя общее количество неправильно положительных предсказаний среди всех отрицательных образцов в наборе данных. Мы можем рассчитать *FPR* следующим образом:

$$FPR = \frac{FP}{TN + FP}. \quad (5)$$

F1-мера (F1-*score*) представляет собой комбинацию точности и полноты. Это гармоническое среднее между точностью и полнотой. Она принимает значение в диапазоне (0,1]. F1-мера рассчитывается следующим образом:

$$\text{F1-}score = \frac{2 * Recall * Precision}{Recall + Precision}. \quad (6)$$

Кроме того, более высокие значения точности, полноты и F1-меры указывают на эффективность классификации.

Полученные результаты и анализ моделей

В этом разделе сравним уровень влияния признаков на обучение моделей, потому что разные признаки играют разную роль в обучении. Экспериментально покажем, что существуют некоторые признаки, называемые лучшими, которые имеют гораздо большую степень влияния, чем другие.

Лучшие признаки, выбранные из 54 признаков PE-заголовка и отсортированные по степени влияния на процесс обнаружения вредоносного ПО, показаны в табл. 2

Мы видим, что признак *Machine* имеет самый значительный показатель воздействия — 23,21 %. Другие признаки хотя и не достигли самого высокого показателя, все же играют существенную роль в классификации вредоносных и доброкачественных файлов. В частности, на 9-й из 54 признаков приходится 79,63 % от показателя воздействия.

Использование лучших признаков из табл. 2 полностью оправдано, особенно при статическом анализе вредоносного ПО, поскольку они содержат важную информацию, необходимую вредоносному ПО для эффективного выполнения в ОС Windows.

Machine: это поле в заголовке файла *COFF*. Оно устанавливает тип архитектуры целевой машины, например Intel, AMD. Это поле частично определяет, какой процессор является целью текущего вредоносного ПО.

ImageBase: когда вредоносная программа выполняется, загрузчик Windows создает процесс для этой вредоносной программы. Загрузчик Windows копирует и загружает PE-файл и секции вредоносной программы с диска в виртуальную память процесса. Поле *ImageBase* указывает предпочтительный адрес, по которому исполняемый файл должен быть отображен в памяти. По умолчанию используется адрес 0x00400000 для 32-разрядных исполняемых файлов, для .dll он иной — 0x00100000.

MajorSubsystemVersion: обычно используется для проверки совместимости системы с текущим PE-файлом. Вредоносные программы часто используют это поле в сочетании с *MajorOperatingSystemVersion*, чтобы убедиться, что они выполняются на зараженной системе, поддерживающей максимум встроенных функций вредоносной программы.

■ ***Таблица 2.*** Лучшие признаки и соответствующий показатель воздействия

■ ***Table 2.*** Best features and the corresponding impact indicator

Ранг	Признак	Показатель воздействия, %
1	Machine	23,21
2	ImageBase	19,37
3	MajorSubsystemVersion	14,42
4	Characteristics	7,37
5	SizeOfOptionalHeader	3,74
6	MajorLinkerVersion	3,68
7	MajorOperatingSystemVersion	3,03
8	Subsystem	2,89
9	MinorLinkerVersion	1,92

Characteristics: указывает атрибуты объекта или файла изображения. Оно помогает определить, имеет ли вредоносная программа вид .exe или .dll, поскольку оба эти расширения используют формат PE-заголовка. По полю *Characteristics* с помощью флага *IMAGE_FILE_32BIT_MACHINE* мы также можем определить, нацелена ли эта вредоносная программа на 32-разрядную или 64-разрядную Windows.

SizeOfOptionalHeader: обеспечивает точный размер необязательного заголовка (*Optional Header*), который необходим для правильного анализа PE-файла. Кроме того, значение этого поля также влияет на сложность PE-файла, поскольку оно напрямую связано с размером других важных полей, таких как *ExportTable*, *ImportTable*, *ResourceTable*, *ImporAddressTable*.

MajorLinkerVersion и **MinorLinkerVersion**: указывают версию компоновщика, который используется для получения объектных файлов (сгенерированных компилятором или ассемблером) и объединения их в исполняемый файл. Обычно эти значения используются для сравнения с *Rich Headers* [23], чтобы найти упакованные файлы.

MajorOperatingSystemVersion: указывает минимальную версию ОС, необходимую для использования этого исполняемого файла.

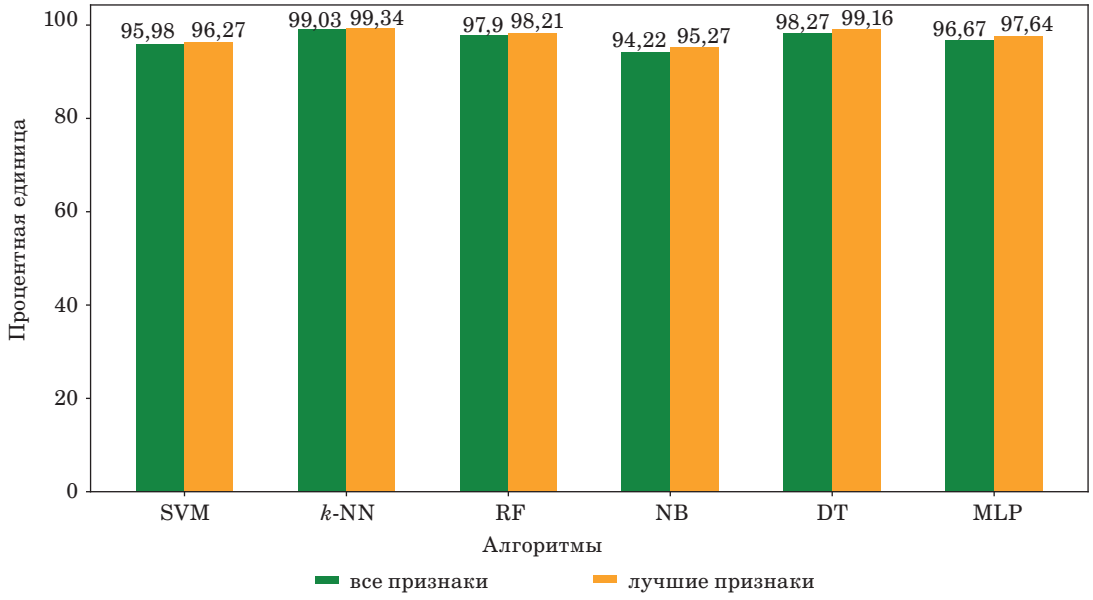
SubSystem: определяет, какая подсистема Windows (если она существует) требуется для запуска файла. Это поле указывает, что PE-файл будет запущен с помощью драйвера, графического или командного интерфейса пользователя.

Анализ

Мы пытались проверить, влияет ли использование лучших признаков на обнаружение вредоносных программ. На самом деле, нельзя сказать, что использование только лучших признаков статистически улучшило результаты. Однако благодаря многим экспериментам мы получили результаты, изложенные ниже. Следует отметить, что эти результаты могут отличаться в зависимости от набора данных, ОС и применимых алгоритмов.

Анализ на основе точности (accuracy)

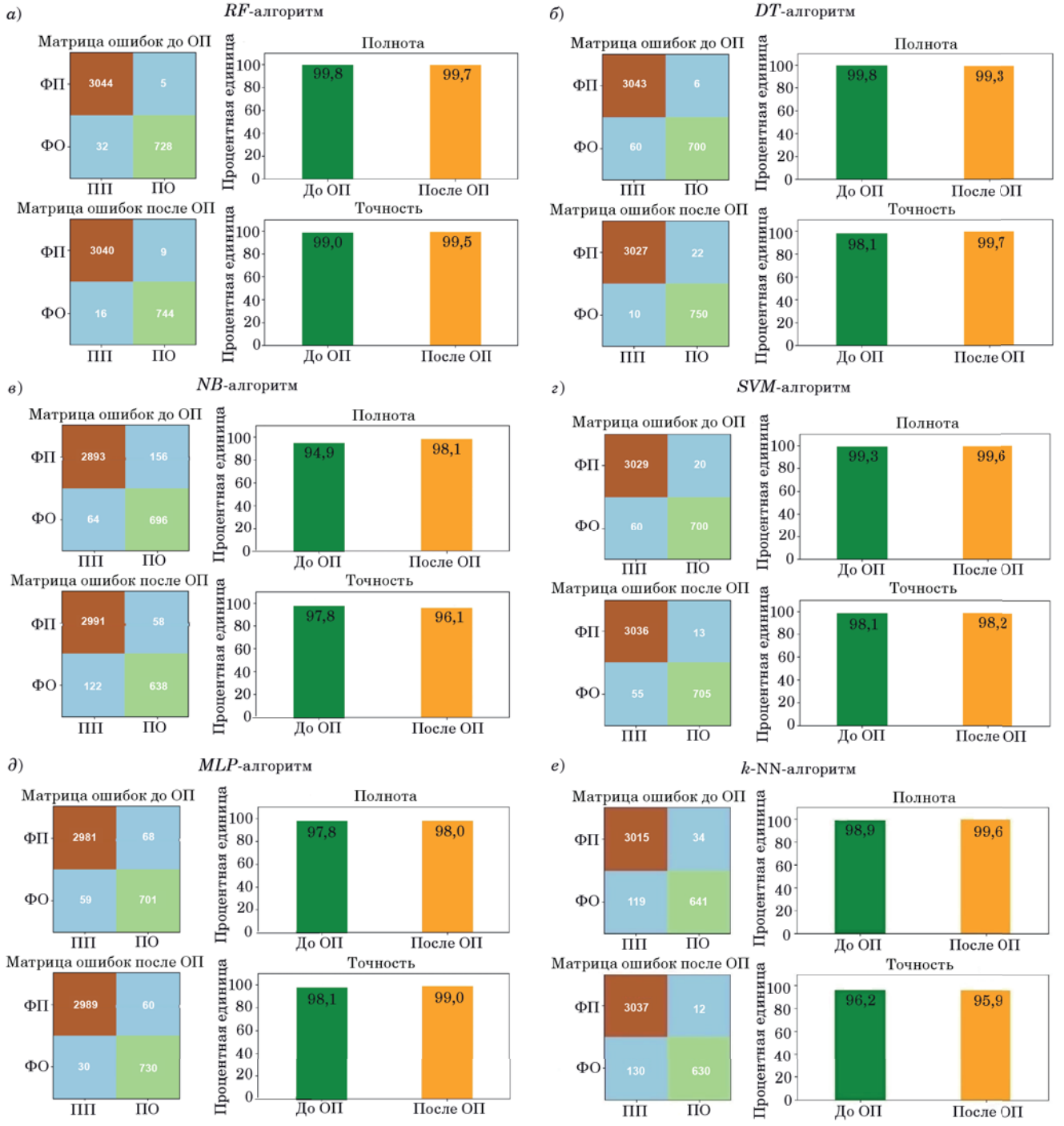
Метрика точности для всех алгоритмов представлена на рис. 2. Результаты экспериментов показывают, что точность всех алгоритмов относительно высока. Повышение точности алгоритмов также указывает на необходимость и влияние отбора признаков, который повышает точность до 1,05 %. Хотя это значение невелико, оно также частично свидетельствует об улучшении модели. Эта разница была обусловлена многими факторами. Одним из важных факторов является обучающий набор данных. Если этот набор данных достаточно велик и точен, алгоритмы будут более эффективными. Тогда эта разница станет более очевидной. Здесь мы игнорируем статистическую погрешность, поскольку оба случая выполняются в абсолютно одинаковых условиях. Производительность алгоритмов *k-NN, DT* и *MLP* является многообещающей. Их максимальная точность доходит до 99,34 %. Таким образом, можно предварительно сделать вывод, что рабочие модели стабильны, а качество набора данных хорошее.



■ ***Рис. 2.*** Сравнение алгоритмов по показателю точность (*accuracy*)
■ ***Fig. 2.*** Comparison of algorithms in terms of accuracy

Анализ на основе полноты и точности (precision)

На рисунке 3 показаны флуктуации индексов *TP*, *TN*, *FP*, *FN* в матрице ошибок каждого алгоритма; изменения соответствующих метрик, таких как полнота, точность при использовании всех признаков PE-заголовка и лучших признаков PE-заголовка. Результат наглядно показы-



■ ***Рис. 3.*** Матрица ошибок, полнота (*recall*) и точность (*precision*) алгоритмов случайный лес (*а*), дерево принятия решений (*б*), наивный байесовский алгоритм (*в*), метод опорных векторов (*г*), многослойный перцептрон (*д*), метод *k*-ближайших соседей (*е*) до и после использования отбора признаков: *ОП* — отбор признаков; *ПО* — прогнозированный отрицательный; *ПП* — прогнозированный положительный; *ФО* — фактический отрицательный; *ФП* — фактический положительный

■ ***Fig. 3.*** Confusion matrix, recall and precision of the algorithms Random Forest (*а*), Decision Tree (*б*), Naive Bayes (*в*), Support Vector Machine (*г*), Multilayer Perceptron (*д*), k-Nearest Neighbors (*е*) before and after using the feature selection

вает увеличение и уменьшение этих индексов в матрице ошибок. Большинство алгоритмов увеличивают показатель *TP*, чтобы продемонстрировать эффективность лучшего набора признаков PE-заголовка для обнаружения большего количества вредоносных программ. Для *FN* большинство значений немного уменьшаются или уменьшаются вдвое (для алгоритмов *NB, SVM* и *k-NN*) (рис. 3, *в*, *г*, *е*). Этот результат показывает, что выбор признаков помогает этим алгоритмам существенно снизить количество ошибочных прогнозов.

Индекс *FP* также значительно снизился в алгоритмах *RF*, *DT* и *MLP* (рис. 3, *а*, *б*, *д*), но почти вдвое (с 64 до 122 образцов, рис. 3, *в*) увеличился при использовании лучшего набора признаков для алгоритма *NB*. Это означает, что вычисление условной вероятности для некоторых признаков может привести к неправильному прогнозированию незараженных файлов во вредоносные. Однако в сочетании с индексом *TN* при использовании лучших признаков PE-заголовка эти алгоритмы значительно улучшаются, позволяя избежать путаницы между вредоносными и обычными файлами.

Кроме того, *RF* и *DT* можно рассматривать как два алгоритма с очень высокой вероятностью прогнозирования файла как вредоносного ПО с точностью 99,5 и 99,7 % соответственно (рис. 3, *а*, *б*). Мало того, *RF* и *k-NN* (рис. 3, *а*, *е*) имеют впечатляющий коэффициент полноты — 99,7 и 99,6 % соответственно. Это означает, что вероятность обнаружения реального вредоносного ПО с помощью *RF* и *k-NN* очень высока или вероятность пропуска реального вредоносного ПО с *RF*, *k-NN* очень мала.

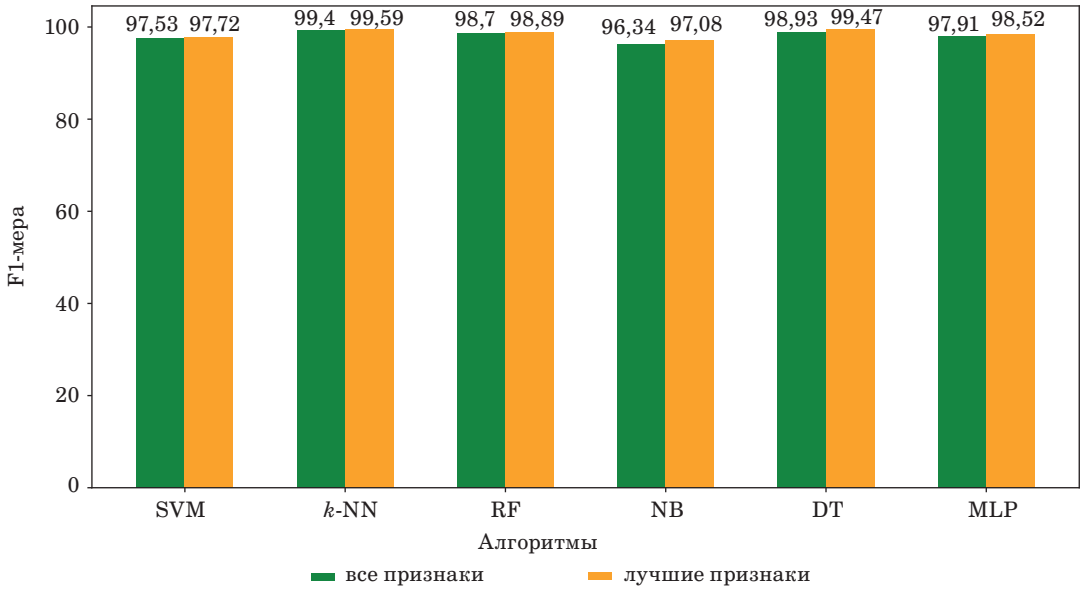
Анализ на основе F1-меры

Как известно, использование только точности или полноты метрик не позволяет правильно оценить качество модели. Поэтому для оценки вышеуказанных моделей следует использовать метрику F1-меру. На рис. 4 показано значение F1-меры всех алгоритмов в случае использования лучших признаков PE-заголовка и в случае без выбора этих признаков. Мы видим, что это значение улучшается благодаря лучшим признакам. Это означает, что лучший набор признаков PE-заголовка играет важную роль в обнаружении вредоносного ПО, поскольку чем выше F1-мера, тем лучше классификатор.

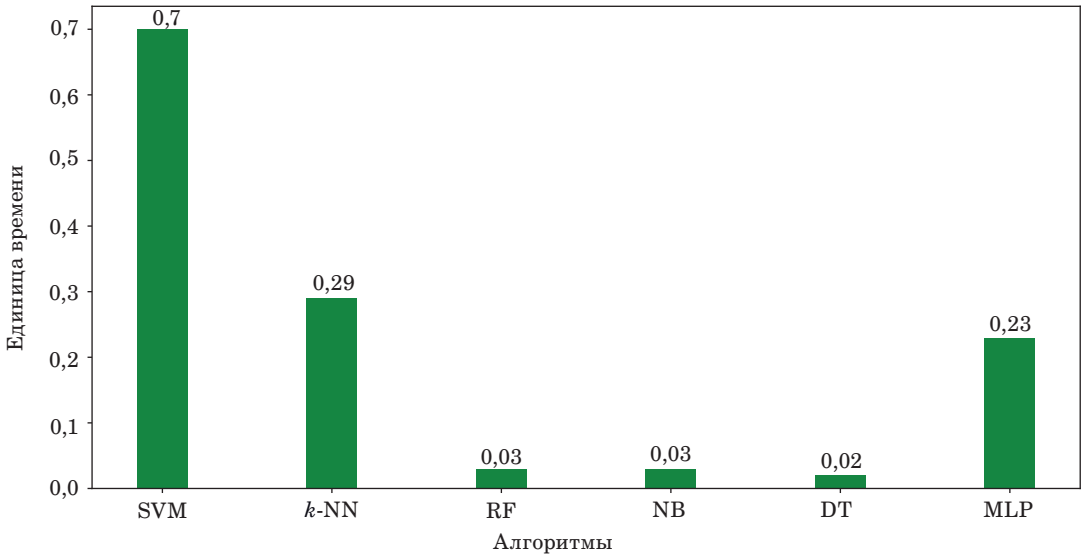
Из анализа на основе точности и полноты легко увидеть, что F1-мера для *DT* и *RF* будет очень высока. Кроме того, алгоритм *k-NN* также имеет высокую F1-меру, которая доказывает, что *k-NN* может быть полностью использован для обнаружения вредоносных программ.

В дополнение к вышеперечисленным метрикам время обработки также является важной метрикой, используемой для сравнения алгоритмов. На рис. 5 показано время обработки всех алгоритмов.

Мы видим, что *RF*, *NB* и *DT* являются алгоритмами с быстрым временем обработки. При высокой скорости обработки данных современных компьютеров обработать структуры деревьев или вычислить значения вероятности для составления прогнозов можно со скоростью



■ ***Рис. 4.*** F1-мера алгоритмов до и после отбора лучших признаков
■ ***Fig. 4.*** F1-score of the algorithms before and after using the best feature selection

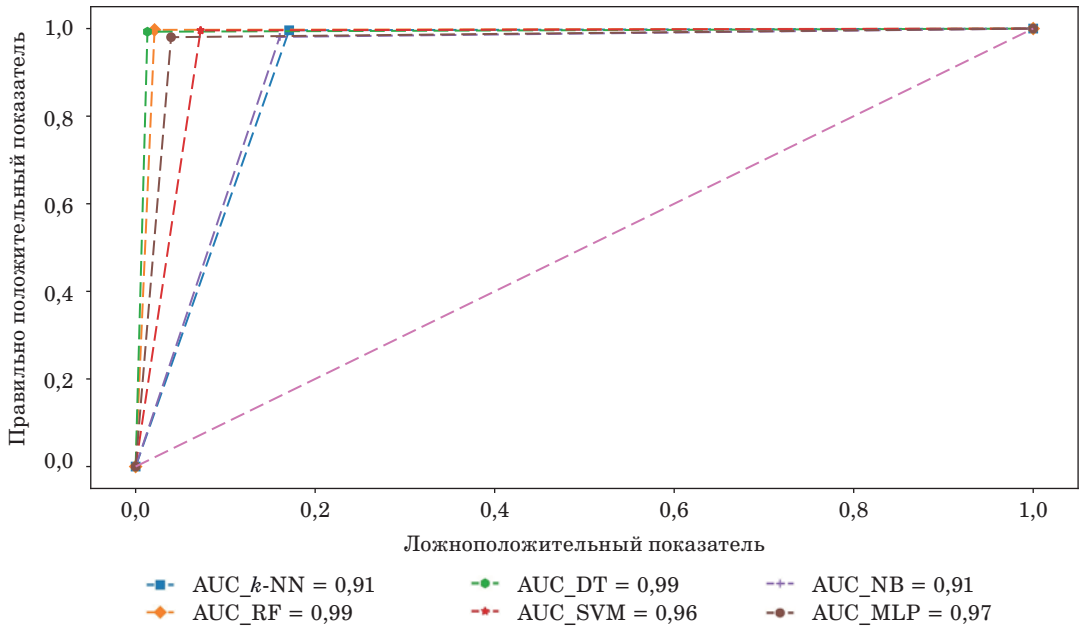


■ ***Рис. 5.*** Относительное время обработки алгоритмов
■ ***Fig. 5.*** Relative processing time of the algorithms

0,02–0,03 с. Алгоритмы *SVM*, *k-NN* и *MLP* показывают более длительное время обработки данных, поскольку оно в значительной степени зависит от природы алгоритма. В частности, вычисление расстояния между несколькими соседями (в *k-NN*), поиск суперплоскости (в *SVM*) или построение нейронной сети с большим количеством скрытых слоев (в *MLP*) для составления прогнозов занимают много времени. Кроме того, скорость процессора также частично влияет на время обработки этих алгоритмов. Поэтому экспериментальные результаты времени обработки являются относительными.

ROC-кривая всех алгоритмов показана на рис. 6. Следует отметить, что:

— чем ближе кривая проходит по левой границе, а затем идет вдоль верхней границы пространства ROC, тем точнее будет результат прогнозирования. Поэтому, чем ближе кривая приближается к диагонали 45 градусов в про-



■ ***Рис. 6.*** ROC-кривая
■ ***Fig. 6.*** ROC-curve

странстве ROC, тем менее точным является прогнозирование;

— площадь под кривой *AUC* (*Area Under the Curve*), ограниченная пространством ROC, является мерой точности прогнозирования, например, 1 — оптимально, 0,5 — плохо. Эта область является показателем хорошей или плохой дискриминации.

Основываясь на свойствах ROC-кривой в сочетании с результатами эксперимента, можно показать наилучший алгоритм прогнозирования на основе значения левой верхней границы и площади под ROC-кривой — *AUC*. Мы видим, что *RF* и *DT* являются лучшими алгоритмами для эффективного прогнозирования, поскольку их ROC-кривые проходят близко к левой верхней границе, и значение *AUC* этих алгоритмов также очень высокое (0,99/1).

Заключение

Кибератаки с использованием вредоносных программ становятся все более популярными, что делает обнаружение вредоносных программ необходимым требованием, особенно в контексте того, что вредоносные программы создаются все умнее, разнообразнее.

Среди популярных ОС именно Windows подвергается наиболее мощным и распространенным атакам вредоносными программами. Стоит отметить, что все приложения, работающие под ОС Windows, имеют формат файла PE-заголовка. В данной работе мы показываем результаты экспериментальных исследований по обнаружению вредоносных программ в ОС Windows с помощью алгоритмов машинного обучения, основанных на признаках PE-заголовка. Исходя из эксперимента, изложенного выше, и полученных результатов, делаем вывод, что применение алгоритмов машинного обучения для обнаружения вредоносных программ вполне возможно и дает хорошие результаты.

Мы протестировали шесть различных алгоритмов: случайный лес (*RF*), дерево принятия решений (*DT*), наивный байесовский алгоритм (*NB*), метод опорных векторов (*SVM*), многослойный перцептрон (*MLP*), метод *k*-ближайших соседей (*k-NN*) — с большим набором данных. Результаты показали, что:

— алгоритмы *RF*, *DT*, *k-NN* и *MLP* могут обнаруживать вредоносные образцы с очень высокой точностью (>98 %). Однако *MLP* и *k-NN* могут не подойти, поскольку время обработки значительно больше, чем у алгоритмов *DT* и *RF*;

— у *RF* очень высокие показатели точности и полноты. Этот факт доказывает, что *RF*-алгоритм очень хорошо подходит для обнаружения вредоносного ПО на ОС Windows.

В дополнение можно также рассмотреть возможность использования алгоритма *NB* в качестве альтернативы. Хотя точность *NB* не так высока, как *DT*, *k-NN* или *MLP*, она достаточно высокая (> 96 %). Кроме того, *NB* считается простым алгоритмом для реализации и имеет быстрое время обработки.

Мы также отметили лучшие признаки, позволяющие идентифицировать вредоносные файлы с высокой точностью и эффективностью. Однако в этой статье мы использовали только вредоносное ПО для архитектуры x86-64, поэтому результаты, полученные в эмпирическом процессе, могут отличаться для набора данных, содержащего вредоносное ПО для архитектуры x64.

Следует подчеркнуть, что многие категории вирусного ПО, нацеленного на ОС Windows, не имеют PE-заголовка. Следовательно, необходимо найти более комплексное новое решение для обнаружения многих типов вредоносных ПО в ОС Windows.

В будущих работах мы сосредоточимся на применении алгоритмов глубокого обучения для обнаружения вредоносных программ и их классификации.

Литература

1. **Pandey A. K., Tripathi A. K., Kapil G., Singh V., Khan M. W., Agrawal A., Kumar R., Khan R. A.** *Trends in Malware Attacks: Identification and Mitigation Strategies*. Critical Concepts, Standards, and Techniques in Cyber Forensics. IGI Global, 2020. Pp. 47–60. doi:10.4018/978-1-7998-1558-7.ch004
2. **Alrzini J., Pennington D.** A review of polymorphic malware detection techniques. *Intern. Conf. on Interdisciplinary Computer Science and Engineering (ICICSE2020)*, 2020. https://pureportal.strath.ac.uk/en/publications/a-review-of-polymorphic-malware-detection-techniques (дата обращения: 29.06.2022).
3. **Mohanta A., Saldanha A.** *Persistence Mechanisms*. Malware Analysis and Detection Engineering. Apress, Berkeley, CA, 2020. Pp. 213–236. doi:10.1007/978-1-4842-6193-4_8
4. **Afianian A., Niksefat S., Sadeghiyan B., Baptiste D.** Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 2019, vol. 52, no. 6, pp. 1–28. doi:10.1145/3365001
5. **Alaeiyan M., Parsa S., Conti M.** Analysis and classification of context-based malware behavior. *Computer Communications*, 2019, vol. 136, pp. 76–90. doi:10.1016/j.comcom.2019.01.003
6. **Sikorski M., Honig A.** *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Soft-*

ware. No Starch Press, 2012. 766 p. doi:10.1016/j.cose.2012.05.004
7. **Ucci D., Aniello L., Baldoni R.** Survey of machine learning techniques for malware analysis. *Computers & Security*, 2019, vol. 81, pp. 123–147. doi:10.1016/j.cose.2018.11.001
8. **Shalaginov A., Banin S., Dehghantanha A., Franke K.** *Machine learning aided static malware analysis: A survey and tutorial*. Cyber Threat Intelligence. Springer, Cham, 2018. Pp. 7–45. doi:10.1007/978-3-319-73951-9_2
9. **Chowdhury M., Rahman A., Islam R.** Malware analysis and detection using data mining and machine learning classification. *Intern. Conf. on Applications and Techniques in Cyber Security and Intelligence*. Edizioni della Normale, Cham, 2017, pp. 266–274. doi:10.1007/978-3-319-67071-3_33
10. **Singh A., Thakur N., Sharma A.** A review of supervised machine learning algorithms. *2016 3rd Intern. Conf. on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2016, pp. 1310–1315.
11. **Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J.** Scikit learn: Machine learning in Python. *The Journal of Machine Learning Research*, 2011, vol. 12, pp. 2825–2830.
12. **Gavrilut D., Cimpoesu M., Anton D., Ciortuz L.** Malware detection using machine learning. *2009 Intern. Multiconf. on Computer Science and Information Technology*, IEEE, 2009, pp. 735–741. doi:10.1109/imcsit.2009.5352759
13. **Rathore H., Agarwal S., Sahay S. K., Sewak M.** Malware detection using machine learning and deep learning. *Intern. Conf. on Big Data Analytics*, Springer, Cham, 2018, pp. 402–411. doi:10.1007/978-3-030-04780-1_28
14. **Bae S. I., Lee G. B., Im E. G.** Ransomware detection using machine learning algorithms. *Concurrency and Computation: Practice and Experience*, IEEE, 2020, vol. 32, no. 18, e.5422. doi.org/10.1002/cpe.5422
15. **Jerlin M. A., Marimuthu K.** A new malware detection system using machine learning techniques for API call sequences. *Journal of Applied Security Research*, 2018, vol. 13, no. 1, pp. 45–62. doi:10.1080/19361610.2018.1387734
16. **Ravi C., Manoharan R.** Malware detection using Windows API sequence and machine learning. *International Journal of Computer Applications*, 2012, vol. 43, no. 17, pp. 12–16. doi:10.5120/6194-8715
17. **Chaudhary S., Garg A.** A machine learning technique to detect behavior based malware. *2020 10th Intern. Conf. on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, 2020, pp. 655–659. doi:10.1109/confluence47617.2020.9058173
18. **Xu Z., Ray S., Subramanyan P., Malik S.** Malware detection using machine learning based analysis of virtual memory access patterns. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2017, pp. 169–174. doi:10.23919/date.2017.7926977
19. **Santos I., Penya Y. K., Devesa J., Bringas P. G.** N-grams-based file signatures for malware detection. *Proc. of the 11th Intern. Conf. on Enterprise Information Systems (ICEIS (2))*, 2009, vol. 9, pp. 317–320. doi:10.5220/0001863603170320
20. **Bai J., Wang J., Zou G.** A malware detection scheme based on mining format information. *The Scientific World Journal*, vol. 2014, pp. 1–12. doi:10.1155/2014/260905
21. **Kim S.** *PE header Analysis for Malware Detection*. Master Thesis, San Jose State University, 2018. 50 p.
22. **Kumar A., Kuppusamy K. S., Aghila G.** A learning model to detect maliciousness of portable executable using integrated feature set. *Journal of King Saud University — Computer and Information Sciences*, 2019, vol. 31, no. 2, pp. 252–265. doi:10.1016/j.jksuci.2017.01.003
23. **Webster G. D., Kolosnjaji B., Pentz C. V., Kirsch J., Hanif Z. D., Zarras A., Eckert C.** Finding the needle: A study of the PE32 rich header and respective malware triage. *Intern. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Cham, 2017, pp. 119–138. doi:10.1007/978-3-319-60876-1_6

UDC 004.056.5
doi:10.31799/1684-8853-2022-4-44-57

Applying machine learning algorithms for PE-header-based malware detection on the Windows operating system

D. T. Le[a], PhD., Lecturer, orcid.org/0000-0003-3735-0314, letranduc@dut.udn.vn
M. H. Pham[b], Engineer, orcid.org/0000-0002-2250-9428
T. D. Dinh[c], PhD., Lecturer, orcid.org/0000-0002-9993-9792
H. P. Do[d], M. Sc., Lecturer, orcid.org/0000-0003-0645-0021
[a]The University of Danang — University of Science and Technology, Information Technology Faculty, 54 Nguyen Luong Bang, 550000, Da Nang, Vietnam
[b]R&D Department, Viettel Business Solutions Corporation, Vietnam
[c]Posts and Telecommunications Institute of Technology, 122 Hoang Quoc Viet, Hanoi, Vietnam
[d]Danang Architecture University, 566 Nui Thanh, Danang, Vietnam

Introduction: The rapid growth of malware and its malicious use result in significant financial losses for various organizations. Many researchers are interested in applying machine learning methods to solve the problem of malware detection. Nevertheless, because

of the diversity of algorithms, each machine learning algorithm has its advantages and disadvantages for a given situation. **Purpose:** To apply machine learning for malware detection in the Windows operating system using Portable Executable header; to compare six different machine learning algorithms based on several criteria. **Results:** The comparison of various algorithms, including such classifiers as Random Forest, Decision Tree, Naive Bayes, Support Vector Machine, Multilayer Perceptron, k-Nearest Neighbors algorithm with a large dataset shows that some algorithms such as Random Forest, Decision Tree, k-Nearest Neighbors, and Multilayer Perceptron can detect malware with very high accuracy (> 98%). The Random Forest algorithm is especially well suited for Windows OS malware detection. At the same time, Naive Bayes classifier also has a high accuracy rate (> 96%) and fast processing time. Therefore, we may consider using Naive Bayes as an alternative.

Keywords — malware, machine learning algorithms, PE-header, Windows.

For citation: Le D. T., Pham M. H., Dinh T. D., Do H. P. Applying machine learning algorithms for PE-header-based malware detection on the Windows operating system. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2022, no. 4, pp. 44–57 (In Russian). doi:10.31799/1684-8853-2022-4-44-57

Reference

1. Pandey A. K., Tripathi A. K., Kapil G., Singh V., Khan M. W., Agrawal A., Kumar R., Khan R. A. *Trends in Malware Attacks: Identification and Mitigation Strategies*. In: *Critical Concepts, Standards, and Techniques in Cyber Forensics*. IGI Global, 2020. Pp. 47–60. doi:10.4018/978-1-7998-1558-7.ch004
2. Alrzini J., Pennington D. A review of polymorphic malware detection techniques. *Intern. Conf. on Interdisciplinary Computer Science and Engineering (ICICSE2020)*, 2020. Available at: https://pureportal.strath.ac.uk/en/publications/a-review-of-polymorphic-malware-detection-techniques (accessed 29 June 2022).
3. Mohanta A., Saldanha A. *Persistence Mechanisms*. In: *Malware Analysis and Detection Engineering*. Apress, Berkeley, CA, 2020. Pp. 213–236. doi:10.1007/978-1-4842-6193-4_8
4. Afianian A., Niksefat S., Sadeghiyan B., Baptiste D. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 2019, vol. 52, no. 6, pp. 1–28. doi:10.1145/3365001
5. Alaeiyan M., Parsa S., Conti M. Analysis and classification of context-based malware behavior. *Computer Communications*, 2019, vol. 136. pp. 76–90. doi:10.1016/j.comcom.2019.01.003
6. Sikorski M., Honig A. *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*. No Starch Press, 2012. 766 p. doi:10.1016/j.cose.2012.05.004
7. Ucci D., Aniello L., Baldoni R. Survey of machine learning techniques for malware analysis. *Computers & Security*, 2019, vol. 81, pp. 123–147. doi:10.1016/j.cose.2018.11.001
8. Shalaginov A., Banin S., Dehghantanha A., Franke K. *Machine learning aided static malware analysis: A survey and tutorial*. In: *Cyber Threat Intelligence*. Springer, Cham, 2018. Pp. 7–45. doi:10.1007/978-3-319-73951-9_2
9. Chowdhury M., Rahman A., Islam R. Malware analysis and detection using data mining and machine learning classification. *Intern. Conf. on Applications and Techniques in Cyber Security and Intelligence*. Edizioni della Normale, Cham, 2017, pp. 266–274. doi:10.1007/978-3-319-67071-3_33
10. Singh A., Thakur N., Sharma A. A review of supervised machine learning algorithms. *2016 3rd Intern. Conf. on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2016, pp. 1310–1315.
11. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J. Scikit learn: Machine learning in Python. *The Journal of Machine Learning Research*, 2011, vol. 12, pp. 2825–2830.
12. Gavrilut D., Cimpoesu M., Anton D., Ciortuz L. Malware detection using machine learning. *2009 Intern. Multiconf. on Computer Science and Information Technology*, IEEE, 2009, pp. 735–741. doi:10.1109/imcsit.2009.5352759
13. Rathore H., Agarwal S., Sahay S. K., Sewak M. Malware detection using machine learning and deep learning. *Intern. Conf. on Big Data Analytics*, Springer, Cham, 2018, pp. 402–411. doi:10.1007/978-3-030-04780-1_28
14. Bae S. I., Lee G. B., Im E. G. Ransomware detection using machine learning algorithms. *Concurrency and Computation: Practice and Experience*, IEEE, 2020, vol. 32, no. 18, e.5422. doi.org/10.1002/cpe.5422
15. Jerlin M. A., Marimuthu K. A new malware detection system using machine learning techniques for API call sequences. *Journal of Applied Security Research*, 2018, vol. 13, no. 1, pp. 45–62. doi:10.1080/19361610.2018.1387734
16. Ravi C., Manoharan R. Malware detection using Windows API sequence and machine learning. *International Journal of Computer Applications*, 2012, vol. 43, no. 17, pp. 12–16. doi:10.5120/6194-8715
17. Chaudhary S., Garg A. A machine learning technique to detect behavior based malware. *2020 10th Intern. Conf. on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, 2020, pp. 655–659. doi:10.1109/confluence47617.2020.9058173
18. Xu Z., Ray S., Subramanyan P., Malik S. Malware detection using machine learning based analysis of virtual memory access patterns. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2017, pp. 169–174. doi:10.23919/date.2017.7926977
19. Santos I., Penya Y. K., Devesa J., Bringas P. G. N-grams-based file signatures for malware detection. *Proc. of the 11th Intern. Conf. on Enterprise Information Systems (ICEIS (2))*, 2009, vol. 9, pp. 317–320. doi:10.5220/0001863603170320
20. Bai J., Wang J., Zou G. A malware detection scheme based on mining format information. *The Scientific World Journal*, vol. 2014, pp. 1–12. doi:10.1155/2014/260905
21. Kim S. *PE Header Analysis for Malware Detection*. Master Thesis, San Jose State University, 2018. 50 p.
22. Kumar A., Kuppusamy K. S., Aghila G. A learning model to detect maliciousness of portable executable using integrated feature set. *Journal of King Saud University — Computer and Information Sciences*, 2019, vol. 31, no. 2, pp. 252–265. doi:10.1016/j.jksuci.2017.01.003
23. Webster G. D., Kolosnjaji B., Pentz C. V., Kirsch J., Hanif Z. D., Zarras A., Eckert C. Finding the needle: A study of the PE32 rich header and respective malware triage. *Intern. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Cham, 2017, pp. 119–138. doi:10.1007/978-3-319-60876-1_6