КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій
Кафедра технологій управління

Спеціальність 122 – Комп'ютерні науки,
освітня програма "Інформаційна аналітика та впливи"

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

"Аналітика та прогнозування вірусного зараження комп'ютерів методами машинного навчання"

Студента 2-го курсу групи ІАВ-21

Гладишка Олексія Олександровича

(прізвище, ім'я, по батькові)

(підпис студента)

Науковий керівник:

доктор технічних наук, доцент

(науковий ступінь, вчене звання)

Єгорченков Олексій Володимирович

(прізвище, ім'я, по батькові)

(дата) *(підпис)*

Попередній захист:

(Висновок: "До захисту в Екзаменаційній комісії")

Завідувач кафедри
технологій управління

(підпис) *(прізвище, ініціали)* *(дата)*

Київ – 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Освітньо-кваліфікаційний рівень Магістр

Спеціальність 122 - Комп'ютерні науки

Освітня програма Інформаційна аналітика та впливи

ЗАТВЕРДЖУЮ
Завідувач кафедри
професор Морозов В.В.
_____
«____» _____20___року

ЗАВДАННЯ
НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент Гладишко Олексій Олександрович

Група ІАВ-21

1. Тема кваліфікаційної роботи
"Аналітика та прогнозування вірусного зараження комп'ютерів методами машинного навчання"

Затверджена наказом по від «____»_____ р. № ___.

2. Строк подання студентом готової роботи – "___"_____ 20___ р.

3. Цільова установка та вихідні дані до роботи розробка моделі з використанням засобів машинного навчання для прогнозування зараження комп'ютерів за допомогою мови програмування Python та допоміжних бібліотек.

4. Зміст роботи дослідження існуючих математичних інструментів та підходів визначення ймовірності зараження комп'ютерів вірусними програмами з статистичних даних; програмна імплементація відповідного математичного апарату до вирішення задачі прогнозування ймовірності зараження та її апробація, вибір інструментів для реалізації інформаційного забезпечення, реалізація та практичне застосування інформаційного забезпечення

5. Перелік графічного матеріалу (слайдів) 38 рисунків, 3 додатки, 21 слайд презентації доповіді

6. Календарний план виконання роботи:

№ з/п	Назва частин роботи	%	Виконання роботи	
			За планом	Фактично
1.	Вибір теми кваліфікаційної роботи	1	01.10.2021	01.10.2021
2.	Вивчення літературних джерел з предмету дослідження	5	05.12.2021	05.12.2021
3.	Складання розгорнутого плану кваліфікаційної роботи	4	20.01.2022	20.01.2022
4.	Ознайомлення наукового керівника з розгорнутим планом кваліфікаційної роботи. Внесення змін.	5	28.01.2022	28.01.2022
5.	Підготовка розділу 1 "Аналіз предметної області"	15	05.02.2022	05.02.2022
6.	Підготовка розділу 2 "Розвідувальний аналіз проблеми"	30	11.03.2022	11.03.2022
7.	Підготовка розділу 3 "Побудова моделі прогнозування ймовірності зараженості комп'ютерів вірусами"	25	06.04.2022	06.04.2022
8.	Підготовка розділу 4 "Розробка інформаційного забезпечення прогнозування ймовірності зараженості комп'ютерів"	10	28.04.2022	28.04.2022
9.	Оформлення кваліфікаційної роботи	5	02.05.2022	02.05.2022

10.	Передача кваліфікаційної роботи рецензенту для рецензування		04.05.2022	04.05.2022
11.	Передача кваліфікаційної роботи науковому керівникові		10.05.2022	10.05.2022
12.	Попередній захист		17.05.2022	17.05.2022

Дата видачі завдання «____» _____ 20___ р.

Керівник роботи д.т.н., доцент Єгорченков Олексій Володимирович

(посада, прізвище, ім'я, по батькові)

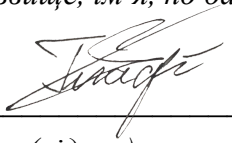
_____

(підпис)

Завдання прийняв до виконання студент групи ІАВ-21

Гладишко Олексій Олександрович

(прізвище, ім'я, по батькові)



_____

(підпис)

ЗМІСТ

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 - Комп’ютерні науки,

освітня програма “Інформаційна аналітика та впливи”

Дипломна робота магістра Гладишка Олексія Олександровича

Тема роботи – “Аналітика та прогнозування вірусного зараження комп’ютерів методами машинного навчання”.

Метою дипломної роботи магістра – дослідження методів прогнозування, які можуть бути використані для передбачення зараженості комп’ютерів вірусними програмами, аналіз тенденцій зараженості пристроїв та характеристика взаємозалежності між вірусними зараженнями та технічними характеристиками систем, розробка моделі з використанням засобів машинного навчання, для прогнозування зараження комп'ютерних пристроїв шкідливими програмами.

Об’єкт дослідження – процеси зараження комп’ютерів вірусними програмами.

Предметом дослідження – інформаційні засоби та технології аналізу даних для визначення основних трендів та ознак, що впливають на зараження комп’ютерів вірусними програмами і моделі прогнозування ймовірності таких випадків.

Наукова новизна результатів роботи полягає у створенні нової агрегованої моделі прогнозування ймовірності зараження комп’ютерів шкідливими вірусними програмами. Незважаючи на велику кількість досліджень, щодо способів захисту систем, попередні дослідження приділяли увагу конкретним заходам для

розпізнавання вірусних програм та дослідженню найбільш популярних видів кібератак.

Практичне значення отриманих результатів – запропонована в ході цієї роботи модель аналізу зараженості комп’ютерів вірусами дозволить виявляти найбільш уразливі аспекти систем, для подальшого їх виправлення та зменшення загрози кібератак.

У роботі досліджується головні закономірності та параметри, які впливають на зараженість пристроїв вірусними програмами, а також існуючі комп’ютерні математичні методи для реалізації моделі прогнозування. Набір даних складається з 7.85 мільйонів записів. На основі зазначеного датасету була побудована модель прогнозування ймовірності зараженості з використанням таких інструментальних засобів, як Python, Jupyter Notebook, бібліотек pandas, scikit-learn, plotly.

Дипломна робота складається зі вступу, основної частини, яка включає чотири розділи, висновків, списку використаних джерел та додатків. Всього налічує 90 сторінок та перелік посилань з 24 джерела на 3 сторінках.

Ключові слова: аналіз даних, комп’ютерні віруси, машинне навчання, математична модель прогнозування, програмний продукт.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ - програмний забезпечення

ПК - персональний комп’ютер

BIOS - Basic Input/Output System

ML - machine learning

РАД - розвідувальний аналіз даних

ШНМ - штучні нейронні мережі

ANN - artificial neural networks

CV - cross validation

ОС - операційна система

ВСТУП

Актуальність теми. У сучасному суспільстві різко зросло використання комп’ютерів. Як результат, користувачі персональних комп’ютерів сьогодні повинні мати комплексні механізми захисту від вірусів, щоб протистояти зростаючим загрозам комп’ютерних вірусів. Нині багато комп’ютерних вірусів призначені для саморозмноження та самовстановлення протягом дуже короткого періоду часу. Вони закодовані особливими інструкціями для знищення та просування через головний комп’ютер, і можуть дуже швидко впливати на безліч програм і додатків. Розпізнавання симптомів комп’ютерного вірусу може допомогти успішно видалити його із зараженого комп’ютера якомога швидше. Чим раніше це їх буде виявлено, тим легше буде відновити будь-які документи чи програми, які можуть бути пошкоджені, і запобігти подальшому поширенню вірусу. З стрімким розповсюдженням інформаційних технологій, збільшується і кількість збереженої інформації. Ця інформація може бути використана для проведення аналізу, тобто, виявлення загальних тенденцій змін, факторів, що впливають на них. За допомогою виявлених тенденцій та факторів можна оптимізувати роботу системи, підвищити продуктивність, скоротити кількість втрат.

Метою магістерської роботи є дослідження методів прогнозування, які можуть бути використані для передбачення зараженості комп’ютерів вірусними програмами, аналіз тенденцій зараженості пристроїв та характеристика взаємозалежності між вірусними зараженнями та технічними характеристиками систем, розробка моделі з використанням засобів машинного навчання, для прогнозування зараження комп'ютерних пристроїв шкідливими програмами.

Об’єктом дослідження є процеси зараження комп’ютерів вірусними програмами.

Предметом дослідження є інформаційні засоби та технології аналізу даних для визначення основних трендів та ознак, що впливають на зараження

комп’ютерів вірусними програмами і моделі прогнозування ймовірності таких випадків.

Наукова новизна результатів роботи полягає у створенні нової агрегованої моделі прогнозування ймовірності зараження комп’ютерів шкідливими вірусними програмами. Незважаючи на велику кількість досліджень, щодо способів захисту систем, попередні дослідження приділяли увагу конкретним заходам для розпізнавання вірусних програм та дослідженню найбільш популярні способи кібератак вірусними програмами.

Впродовж цієї магістерської роботи були досліджені загальні тенденції залежностей технічних характеристик систем, які вплинули на потрапляння шкідливого програмного забезпечення.

Практичне значення отриманих результатів полягає у тому, що запропонована в ході цієї роботи модель аналізу зараженості комп’ютерів вірусами дозволить виявляти найбільш уразливі аспекти систем, для подальшого їх виправлення та зменшення загрози кібератак. Ці дані будуть корисні для будь-яких компаній, які у своєму робочому процесі використовують мережу комп'ютерних пристроїв, а також для звичайних повсякденних користувачів.

Методи дослідження, що були використані впродовж роботи включають математичне моделювання, розвідувальний аналіз, регресійні методи, статистичні методи та інструменти для побудови графіків та візуалізації, методи машинного навчання. Для розробки моделі прогнозування ймовірності зараження комп’ютерів вірусами використовувалося програмне середовище JupyterNotebook з використанням мови програмування Python та відкритих бібліотек та API.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, 4 розділів, висновків. Також наводиться список літератури з 24 пунктів та 3 додатка. Загальний обсяг кваліфікаційної роботи становить 90 сторінок, із них 55 сторінки основного тексту, який містить 38 рисунка.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної області

Швидкий темп розвитку наукового та технічного прогресу надає неймовірні можливості покращення життя кожного з нас. Але в той самий час і з'являються нові інструменти для користувачів з негативними намірами. Навряд чи знайдеться користувач ПК, який хоча б раз в житті не постраждав від діяльності шкідливих програм або додатків, вірусів. У більшості випадків, завданої шкоди полягає в витрачений час на перевстановлення операційної системи, у гірших випадках – значні матеріальні втрати. Щоб заражати і розповсюджувати віруси поміж системами розробники шкідливого програмного забезпечення використовують інформацію про вразливості цільового ПЗ та соціальну інженерію про проведення атак. Часто не досвідчені користувачі персональних комп'ютерів відносять до вірусних програм інші види зловмисного програмного забезпечення, такі як спам або програми-шпигуни.

Щорічні фінансові збитки завдані комп'ютерними вірусами сягають кілька мільярдів доларів через призупинення роботи великих веб-додатків чи сайтів, крадіжкою персональних даних користувачі, знищення та модифікацію інформації файлів та баз даних, викликаючи системні помилки критичного характеру.

Комп'ютерні віруси коштують щороку приблизно 55 мільярдів доларів на очищення та ремонт. Найбільшим комп'ютерним вірусом в історії є вірус Mydoom, який у 2004 році завдав збитків приблизно в 38 мільярдів доларів. Серед інших відомих – хробак Sobig на 30 мільярдів доларів і хробак Klez на 19,8 мільярдів доларів. На щастя, сучасні безпечні комп'ютери та операційні системи значно ускладнюють доступ вірусів і хробаків у наше пов'язане життя [1].

Якби економічна шкода від кіберзлочинності, яка, за прогнозами, завдасть збитків на загальну суму 6 трильйонів доларів у всьому світі в 2022 році, була б

оцінена як країна, то вона була б третьою за величиною економікою світу після США та Китаю.

Очікується, що глобальні витрати на кіберзлочинність зростатимуть на 15 відсотків на рік протягом наступних п’яти років, досягнувши 10,5 трильйона доларів США щорічно до 2025 року, у порівнянні з 3 трильйонами доларів США в 2015 році. Інновації та інвестиції, експоненціально перевищує збиток, завданий природними катаклізмами за рік.

Оцінка вартості збитків базується на історичних цифрах кіберзлочинності, включаючи останній ріст у порівнянні з минулим роком, різке збільшення хакерської діяльності угруповань, спонсорованих національними державами та організованими злочинними групами, а також поверхню кібератак, яка буде на порядок більше у 2025 році, ніж це сьогодні.

Витрати на кіберзлочинність включають пошкодження та знищення даних, вкрадені гроші, втрату продуктивності, крадіжку інтелектуальної власності, крадіжку особистих і фінансових даних, розкрадання, шахрайство, порушення нормального ходу бізнесу після атаки, судове розслідування, відновлення та видалення зламаних даних і систем, а також шкоди репутації [2].

У багатьох країнах передбачена кримінальна відповідальність за створення та поширення шкідливих програм, в тому числі і вірусів. Зокрема, в Україні поширення комп'ютерних вірусів переслідується і карається відповідно до Кримінального кодексу (статті 361, 362, 363) [3].

Назва програми «комп'ютерний вірус» походить від однойменного терміну з біології за її здатність до саморозмноження. Вперше цей термін з’явився на початку 1970-х років і використовувався в програмуванні, а також у літературі, як в технічній так і художжній. Письменник Грегорі Белфорд одним із перших використав новий термін у своєму фантастичному оповіданні “Людина в рубцях”. Однак автором терміну все ж вважається Фред Коен, який у 1984 році опублікував одну з перших академічних статей, що були присвячені вірусам, де і було використано цю назву.

Наприкінці 1980-х і на початку 90-х комп’ютерні віруси вважалися локальним міфом. У той час було створено лише малу кількість шкідливого програмного забезпечення тому зараження було досить рідкісним явищем. Сьогодні ситуація значно інша: відомо понад 56 000 різних штамів вірусів, хробаків і троянських коней. На додаток до збільшення кількості вірусів, ці цифрові загарбники також стали більш складними і їх важко виявити. Ці покращення можна хоча б частково віднести до зусиль антивірусних продуктів. Оскільки антивірусна технологія вдосконалюється і дає постачальникам можливість виявляти новітні вірусні загрози, розробники вірусів розвивають загрози наступного покоління, щоб використовувати слабкі місця в антивірусних продуктах, щоб уникнути їх виявлення.

Ця спільна еволюція привела до ряду помітних удосконалень протягом багатьох років як у вірусних, так і в антивірусних технологіях. Перші комп’ютерні віруси були простими програмами на машинній мові, які мали можливість приєднувати і поширювати ідентичні копії самих себе з програми в програму або з диска на диск. Такі віруси виявилося легко виявити антивірусними продуктами, оскільки вони завжди робили точні копії самих себе. Антивірус міг просто витягти сигнатуру вірусу – невелику послідовність байтів, знайдених у вірусі, але навряд чи знайдених в інших, не заражених програмах – і додати її до вірусної бази. Тоді антивірусне програмне забезпечення буде використовувати цю базу сигнатур під час сканування комп’ютерної системи. Якщо будь-який із файлів у системі містив підписи, що відповідають тим у базі даних, файл можна помістити на карантин та виправити.

Цей метод виявлення деякий час працював добре, але коли автори вірусів зрозуміли, як легко антивірусні продукти виявляють їхні твори, вони швидко почали розвиватися. Автори вірусів зрозуміли, що вони могли б ускладнити виявлення своїх вірусів, якби віруси не поширювали точні копії самих себе. Якщо один і той самий вірусний сигнатуру не вдається знайти в усіх заражених вірусами

файлах, антивірусним продуктам буде важко виявити вірус. Це призвело до створення зашифрованого вірусу.

Зашифрований вірус приховує свою присутність, шифруючи основну частину своєї вірусної логіки. Коли запускається інфікована програма, вірус узурпує контроль над системою, а потім використовує просту підпрограму дешифрування, щоб розшифрувати решту зашифрованої вірусної логіки, отже, логіка вірусу видима лише тоді, коли він активно працює, але ніколи, коли він запущений. зберігається на диску в зараженому файлі. Потім вірус запускає цю розшифровану логіку, щоб заразити інші файли програми. У міру зараження нових програмних файлів вірус може трохи змінити шифрування (змінюючи ключ шифрування), щоб викликати нові інфекції, які мало схожі на батьківську інфекцію.

Зашифрований вірус створив нові проблеми для антивірусних продуктів. Очевидно, оскільки основна частина вірусу була зашифрована, дослідники антивірусів не могли легко вибрати сигнатури вірусів і сподіватися знайти їх у всіх заражених файлах. Насправді ці прості зашифровані віруси все ще мали статичний компонент, який не змінюється від зараження до зараження – їхня рутинна логіка розшифровки – і цього невеликого фрагмента коду було достатньо, щоб антивірусні продукти могли витягти цей відбиток та виявити це нове покоління вірусів.

Також одними із засобом зараження шкідливими вірусними програми в обхід механізму захисту програмного забезпечення з використанням антивірусів є соціальні технології, такі як спам і фішинг. Швидко набирає популярності найсучасніший вид вірусів під назвою хробаки-ботнети [4]. Спочатку вони розповсюджувались за допомогою троянських програм, але з розвитком p2p-мереж здатні це робити самостійно.

Фішинг – це величезна загроза, яка з кожним роком стає все більш поширеною. У 2021 році дослідження Tessian показало, що співробітники отримують в середньому 14 шкідливих електронних листів на рік [5]. Особливо

сильно постраждали деякі галузі: працівники роздрібної торгівлі отримували в середньому 49. Дослідження ESET за 2021 рік показало збільшення на 7,3% атак на основі електронної пошти з травня по серпень 2021 року, більшість з яких були частиною фішингових кампаній.

Дослідження IBM у 2021 році підтвердило цю тенденцію, посилаючись на зростання фішингових атак на 2 відсотка в порівнянні з 2019 і 2020 роками, частково викликане COVID-19 та невизначеністю ланцюга поставок. Звіт CISCO про тенденції кібербезпеки за 2021 рік показує, що принаймні одна особа натиснула фішингове посилання приблизно в 86% організацій. Дані компанії свідчать про те, що близько 90% зловживань даних припадає на фішинг.

Фішингові атаки розподіляються нерівномірно протягом року. Cisco виявила, що пік фішингу, як правило, припадає на святкові дні, виявивши, що кількість фішингових атак у грудні зросла на 52%.

Служби безпеки різноманітних компаній наводили наступні дані щодо успішності фішингових атак:

– 60% організацій втратили дані
– 52% організацій зазнали крадіжок облікових даних або облікової записів
– 47% організацій були заражені програмами-вимагателями
– 29% організацій були заражені шкідливим ПЗ
– 18% організацій зазнали фінансових втрат

Віруси у складі іншого зловмисного ПЗ остаточно оформляються як засіб кіберзлочинності [6].

1.2 Постановка завдання та огляд методів його вирішення

Комп'ютерний вірус – комп'ютерна програма, яка має здатність до прихованого самопоширення. Одночасно зі створенням власних копій віруси можуть завдавати шкоди: знищувати, пошкоджувати, викрадати дані, знижувати

або й зовсім унеможливлювати подальшу працездатність операційної системи комп'ютера.

Промисловість шкідливих програм продовжує бути добре організованим, добре фінансованим ринком, присвяченим ухиленню від традиційних заходів безпеки. Як тільки комп'ютер заражений шкідливим програмним забезпеченням, злочинці можуть завдати шкоди споживачам і підприємствам багатьма способами. Тому проблема прогнозування зараження комп`ютерів є дуже актуальною у наш час.

Маючи більш ніж один мільярд підприємств і споживачів, корпорація Microsoft дуже серйозно ставиться до цієї проблеми і глибоко інвестує в поліпшення безпеки.

Як частина їхньої загальної стратегії для виконання задачі аналізу та прогнозування даних зараження комп`ютерів шкідливими програмами, корпорація Майкрософт закликає наукову спільноту до розробки рішення для прогнозування того, чи скоро машина буде вражена шкідливим програмним забезпеченням. Microsoft надає учасникам конкурсу у Kaggle набір даних про шкідливі програми, щоб заохотити прогрес у відкритому коді на ефективних методах прогнозування виникнення шкідливих програм.

Існує багато методів аналізу, але для вирішення задачі прогнозування даних зараження комп’ютерів шкідливими програмами, потрібно використовувати методи машинного навчання.

Машинне навчання (англ. machine learning) – це підгалузь штучного інтелекту в галузі інформатики, яка часто застосовує статистичні прийоми для надання комп'ютерам здатності «навчатися» (тобто, поступово покращувати продуктивність у певній задачі) з даних, без того, щоби бути програмованими явно. Еволюціонувавши з досліджень розпізнавання образів та теорії обчислювального навчання в галузі штучного інтелекту, машинне навчання досліджує вивчення та побудову алгоритмів, які можуть навчатися й робити передбачення з даних, – такі алгоритми долають слідування строго статичним

програмним інструкціям, здійснюючи керовані даними прогнози або ухвалювання рішень.

Дана галузь часто перетинається та тісно пов'язана з обчислювальною статистикою. Вона тако зосереджує свою увагу на прогнозуванні використовуючі комп’ютерні засоби. Даний вид статистики має тісні зв'язки з математичною оптимізацією, яка забезпечують цю галузь методами, теорією та прикладними областями. Інколи машинне навчання використовують у поєднанні з добуванням і обробкою даних, де друга підгалузь більше спрямована на проведенні розвідувальному аналізі отриманої інформації, і є відомою як навчання без учителя. Машинне навчання також може бути спонтанним, і застосовуваним для навчання та встановлення базових характерів поведінки різних суб'єктів, а потім застосовуваним для пошуку виразних аномалій.

У сучасному бізнесі машинне навчання має дуже широкий спектр застосувань, який з часом буде зростати. Субдомени ML включають, серед іншого, рекомендації щодо соціальних мереж і продуктів, розпізнавання зображень, медична діагностика, мовний переклад, розпізнавання мовлення та аналіз даних. Платформи соціальних мереж, такі як Facebook, Instagram та LinkedIn, використовують ML, щоб пропонувати, на основі вподобань користувачів, сторінки чи групи, до яких можна приєднатися. Ця технологія аналізує історичні дані про те, що сподобалося іншим, або про контент, подібний до того, що сподобалося користувачеві, щоб давати рекомендації або додавати їх у свою стрічку новин. Крім того, технологію ML можна використовувати в інтернет-магазинах, щоб рекомендувати товари на основі попередніх покупок, пошуків та інших дій подібних користувачів.

Дуже важливою сферою застосування технології ML сьогодні є розпізнавання зображень. Завдяки цьому платформи соціальних мереж дозволяють позначати людей на фотографіях. Поліція використовує його для пошуку підозрюваних на фото чи відео. Завдяки незліченній кількості камер,

встановлених в аеропортах, магазинах і дверних дзвінках, можна помітити багатьох злочинців або побачити, куди вони поділися.

Іншою важливою областю застосування ML є медична діагностика. Після такого інциденту, як серцевий напад, ви можете переглянути дані, щоб виявити ознаки, які ви не помітили. Історичні медичні дані можуть бути введені в систему, яку використовують лікарі або лікарні, щоб виявити взаємозв'язки між вхідними (поведінка, результат тесту, симптом) і вихідними (наприклад, інфаркт міокарда) даними. Коли лікар в подальшому внесе в систему свої нотатки та результати тестів, алгоритм зможе виявляти симптоми інфаркту набагато ефективніше, ніж люди, що дозволить реалізувати профілактичні заходи.

Іншим прикладом використання технології ML є переклад вмісту веб-сайту та додатків на мобільні платформи. Деякі програми працюють краще, а інші — гірше, що є результатом використання різних моделей, методів і алгоритмів машинного навчання.

На сьогоднішній день технологія ML – це хліб і масло для банківських систем і систем кредитних карток. Щоб виявити певні ознаки шахрайства, що не є проблемою для ML, знадобиться багато часу або буде зовсім поза межами досяжності людини. Аналіз та позначення великої кількості транзакцій (чесних чи шахрайських) у майбутньому може дозволити виявити шахрайство в окремих операціях. Ідеальним типом машинного навчання (ML) для цієї мети є аналіз даних.

В межах галузі аналізу даних машинне навчання є методом, який використовується для знаходження складних моделей та алгоритмів, які слугують прогнозуванню – в комерційному застосуванні це відоме як передбачувальна аналітика. Ці аналітичні моделі дозволяють дослідникам, науковцям з даних, інженерам та аналітикам «виробляти надійні, повторювані рішення та результати» та розкривати «приховані розуміння» шляхом навчання з історичних співвідношень та тенденцій в даних [7].

Проведений аналіз показав, що:

– проблема прогнозування зараження комп’ютерів шкідливими програмами є актуальною в наш час;

– існує багато методів аналізу, в першу чергу, методи машинного навчання, для розв’язання цієї проблеми;

– для вирішення поставленої задачі слід використати навчальний набір даних, по якій побудувати модель.

1.3 Огляд існуючих у світі інструментів боротьби з шкідливим програмним забезпеченням

Комп’ютерні віруси — це програми з виконуваним кодом, які мають унікальну здатність реплікуватися в комп’ютерній системі та швидко поширюватися з одного комп’ютера на інший, впливаючи на файли, документи та програми, щоб змінити їхню нормальну роботу. Віруси представлені як шаблони комп’ютерних навчальних кодів, які існують у комп’ютерних системах. З іншого боку, антивіруси — це програми, спеціально розроблені для протидії викликам, викликаним вірусами, оскільки вони захищають комп’ютерні системи від вірусних атак, посилаючись на засоби керування, покращені завдяки їхнім базам даних. Тому антивіруси сканують комп’ютер, використовуючи певні шаблони байтів, які вказують на відомі віруси [8]. Щоб залишатися в курсі, вони повинні постійно оновлювати свої бази даних щоразу, коли виникають нові вірусні штами.

Захист цифрових активів організації від зловмисного програмного забезпечення став проблемою через величезний обсяг і різноманітність нових загроз. На початку комерційної антивірусної індустрії – з 1990-х по 2000-ті роки – більшість рішень використовували ряд поширених методів. Хоча виявлення на основі сигнатур є корисним і все ще використовується, воно стало обмеженим через безперервну появу нових вірусів. Щоб подолати цю проблему, евристичний метод виявлення сканує підозрілі характеристики, які можна знайти в невідомих, нових вірусах і модифікованих версіях відомих вірусів. Виявлення руткітів сканує та блокує шкідливий код, призначений для отримання адміністративного доступу

до машини, а виявлення в реальному часі сканує та відстежує файли під час доступу до них.

Найважливішою частиною будь-якої сучасної антивірусної програми (крім так званого движка) є база вірусних сигнатур. Сигнатури вірусів — це певна (зазвичай дуже коротка) інформація, яка дозволяє відносно однозначно ідентифікувати певний тип або навіть ціле сімейство вірусів. Найпопулярнішими є три типи підпису:

1. сигнатури, створені за допомогою хеш-функції
2. сигнатури байтів (шаблони)
3. евристичні сигнатури

Для створення сигнатур шкідливого програмного забезпечення використовують простий і легкий метод під назвою хеш-функції. Для будь-якого великого числа чи будь-якої програми чи даних будь-якого розміру можна присвоїти відносно невелике значення, яке має фіксований розмір за допомогою застосування математичної функції. Ці значення називаються хешами. Створені таким чином ярлики надають можливість у майбутньому легко ідентифікувати конкретне вірусне програмне забезпечення.

Однак використання сигнатур такого типу веде за собою деякі суттєві недоліки. Навіть найменша зміна в коді комп’ютерних вірусів (новий варіант вірусу, поліморфне зловмисне програмне забезпечення тощо) означає, що стара сигнатура більше не виявляє нові версії вірусу. Крім того, колізії хеш-функцій (тобто випадки, коли ми отримуємо однакові хеші для різних повідомлень або програм) потенційно можуть викликати помилкові спрацьовування. Все це означає, що антивірусні рішення не можуть покладатися тільки на цей тип сигнатур.

Однак у разі поліморфного шкідливого програмного забезпечення або шкідливих файлів, що містять змінні дані, так зване нечітке хешування. Ці методи зазвичай дозволяють створити загальний підпис для різних вхідних даних, які, мають деякі загальні особливості (наприклад, програми, що містять загальні

фрагменти коду). Однак ці типи методів досить вимогливі до обчислень і не гарантують дуже високої ефективності.

Інший популярний спосіб генерування сигнатур зловмисного програмного забезпечення — використання певних вибраних послідовностей байтів, присутніх у шкідливому коді або в даних, які він використовує.

Цей метод достатньо ефективний у боротьбі з сімействами вірусів на практиці, тому що зазвичай ідентичні шаблони байтів можна знайти у всіх штамах певної шкідливої програми. Завдяки простоті цього методи він використовується практично від самого початку історії боротьби з шкідливим програмним забезпеченням. Однак він також може генерувати помилкові виявлення. Якщо ми припустимо, що сигнатурою (шаблоном байтів) певного вірусу є "FF 5c 0c 4d c2 21 1d 18", то, звичайно, будь-який файл (навіть самий звичайний текстовий файл), що містить таку послідовність байтів, створить анти-попередження про вірус.

Наведемо приклад сигнатури. Припустимо виробники антивірусів погодилися, що наступний (цілком нешкідливий) рядок символів буде виявлено для цілей тестування всіма антивірусними програмами як загроза:

IJD&$87JK7_D(!!D9%%#P$UKR-DEFAULT-TEST--ANTIVIRUS-SIGNATURE!$H+H

Сучасні антивірусні програми також використовують евристичні сигнатури. Загалом, евристичні методи у випадку антивірусів — це всі інші методи виявлення загроз, на додаток до перерахованих вище традиційних методів сигнатур. Евристичні методи є найскладнішою частиною будь-якого антивірусного арсеналу. Крім того, практично кожен виробник антивірусів розробляє власні запатентовані евристичні алгоритми, і саме в цій галузі він може найбільшою мірою продемонструвати свою інноваційність і потенціал (традиційні сигнатури зазвичай майже ідентичні для багатьох різних виробників - навіть явище обміну традиційними підписами з іншими виробниками відомий, наприклад, через такі платформи, як VirusTotal).

На практиці евристичні методи виявлення вірусів можуть зводитися, наприклад, до відстеження викликів API, використання пісочниці в поєднанні з відстеженням поведінки окремих програм, відстеження аномалій, що виникають в операційній системі та файловій системі, і багатьох інших подібних прийомів.

Підходи нового покоління розширюють виявлення на основі сигнатур за допомогою виявлення поведінки, машинного навчання, пісочниці та інших методів, які оптимізовані для боротьби з такими загрозами, як шкідливі URL-адреси, викрадачі браузера, розширені постійні загрози та фішингові експлойти.

У антивірусній галузі машинне навчання зазвичай використовується для покращення можливостей виявлення продукту. У той час як звичайна технологія виявлення покладається на правила кодування для виявлення шкідливих шаблонів, алгоритми машинного навчання будують математичну модель на основі зразкових даних, щоб передбачити, чи є файл «хорошим» чи «поганим».

Простіше кажучи, це передбачає використання алгоритму для аналізу спостережуваних точок даних двох, створених вручну наборів даних: одного, який містить лише шкідливі файли, і одного, який містить лише нешкідливі файли. Потім алгоритм розробляє правила, які дозволяють йому відрізнити хороші файли від поганих, не отримавши вказівок щодо того, які типи шаблонів або точок даних шукати. Точка даних — це будь-яка одиниця інформації, пов’язана з файлом, включаючи внутрішню структуру файлу, компілятор, який використовувався, текстові ресурси, скомпільовані у файл, і багато іншого.

Алгоритм продовжує обчислювати й оптимізувати свою модель, поки не отримає точну систему виявлення, яка в ідеалі не класифікує хороші програми як погані, а погані як хороші. Він розвиває свою модель, змінюючи вагу або важливість кожної точки даних. З кожною ітерацією модель стає дещо кращою в точному виявленні шкідливих і нешкідливих файлів. Машинне навчання в найближчі роки може відігравати все більш важливу роль у світі кібербезпеки.

Однак варто зауважити, що рішення, які використовують комбінацію технологій захисту, ймовірно, забезпечать кращу безпеку, ніж продукт, який повністю заснований на штучному інтелекті. Наприклад, Emsisoft використовує силу штучного інтелекту та машинного навчання, а також інші технології захисту, такі як поведінковий аналіз та перевірки підписів [9]. Ці системи працюють в синергії, щоб подвійно та потрійно перевіряти результати один одного, щоб забезпечити найкращий захист від шкідливого програмного забезпечення.

РОЗДІЛ 2 РОЗВІДУВАЛЬНИЙ АНАЛІЗ ПРОБЛЕМИ

2.1 Підготовка даних до розвідувального аналізу

Розвідувальний аналіз даних – попередній аналіз даних з метою виявлення найзагальніших закономірностей та тенденцій, характеру та властивостей даних аналізу, законів розподілу величин, які аналізуються.

РАД використовується для знаходження зв'язків між змінними в ситуаціях, коли відсутні (або недостатні) апріорні уявлення щодо природи цих зв'язків. Як правило, при розвідувальному аналізі враховується та порівнюється велика кількість змінних, а для пошуку закономірностей використовуються значна кількість методів [10].

Результати розвідувального аналізу не використовуються для вироблення управлінських рішень. Їхнє призначення — допомога у розробці найкращої стратегії поглибленого аналізу, висування гіпотез, уточнення особливостей застосування тих чи інших математичних методів та моделей. Без розвідувального аналізу поглиблений аналіз даних проводитиметься практично «наосліп».

До основних методів розвідувального аналізу відноситься процедура аналізу розподілів змінних, кореляційний аналіз з метою пошуку коефіцієнтів, що перевершують за величиною певні граничні значення, факторний аналіз, дискримінантний аналіз, багатовимірне шкалювання, візуальний аналіз гістограм і т.д.

Першим кроком в будь-якому проекті з машинного навчання є ознайомлення з даними. Щоб це зробити, потрібно, наприклад, з'ясувати діапазони значень, що приймаються змінними, їх типи, а також дізнатися про кількість пропущених значень.

Для цього, спочатку, завантажимо всі необхідні бібліотеки (рис 2.1).

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import lightgbm as lgb
import xgboost as xgb
import time
import datetime

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold
, TimeSeriesSplit
from sklearn.metrics import mean_squared_error, roc_auc_sc
ore
from sklearn.linear_model import LogisticRegression, Logis
ticRegressionCV
import gc
from tqdm import tqdm_notebook
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
```

Рисунок 2.1 – Завантаження бібліотек

Бібліотека pandas мови програмування Python надає нам багато корисних інструментів для розвідувального аналізу даних. Це бібліотека з відкритим вихідним кодом, яка створена в основному для легкої та інтуїтивно зрозумілої роботи з реляційними або міченими даними. Pandas надає різноманітні структури даних та операції для маніпулювання числовими даними та часовими рядами. Ця бібліотека побудована на основі бібліотеки NumPy. Pandas швидкий і має високу швидкість і продуктивність для користувачів. Серед переваг бібліотеки варто виділити можливість завантажувати дані з різних файлових об’єктів, легку обробку відсутніх даних (позначених як NaN) як з плаваючою комою, так і з даними без плаваючої коми. Pandas має потужну групу функціональних можливостей для виконання операцій розділення-застосування-об'єднання над наборами даних.

Scikit-learn — це популярна та надійна бібліотека машинного навчання, яка має широкий асортимент алгоритмів, а також інструментів для візуалізації

машинного навчання, попередньої обробки, підбору моделі, вибору й оцінки. Спираючись на NumPy, SciPy і matplotlib, Scikit-learn має ряд ефективних алгоритмів для класифікації, регресії та кластеризації. До них належать машини опорних векторів, тропічні ліси, підвищення градієнта, k-середні та DBSCAN.

Scikit-learn може похвалитися відносною простотою розробки завдяки своїм послідовним і ефективно розробленим API, обширній документації для більшості алгоритмів і численним онлайн-підручникам. Бібліотека написана переважно на Python і використовує NumPy для високопродуктивної лінійної алгебри, а також для операцій з масивами. Деякі основні алгоритми Scikit-learn написані на Cython, щоб підвищити загальну продуктивність. Як бібліотека вищого рівня, яка включає декілька реалізацій різних алгоритмів машинного навчання, Scikit-learn дозволяє користувачам створювати, навчати й оцінювати модель за допомогою кількох рядків коду. Scikit-learn надає єдиний набір високорівневих API для побудови конвеєрів або робочих процесів машинного навчання.

Для візуалізації результатів дослідження була використана бібліотека plotly. Це бібліотека з відкритим вихідним кодом, яка надає список типів діаграм, а також інструменти зі зворотнім викликом для створення інформаційної панелі. Головним плюсом plotly є його інтерактивність і, звісно, візуальна якість. Plotly користується більшим попитом, ніж інші бібліотеки, такі як Matplotlib і Seaborn. Plotly надає список діаграм, які мають анімацію в 1D, 2D і 3D.

Для оптимізації використання оперативної пам`яті комп`ютера, задамо типи даних вручну:

– не числовим присвоєно тип «category»;
– бінарним – int8;
– 64 бітним – 32 або 16 бітні, якщо можливо [11].

Типи даних деяких стовпчиків зображено на рисунку 2.2.

```
dtypes = {
        'MachineIdentifier':
'category',
        'ProductName':
'category',
        'EngineVersion':
'category',
        'AppVersion':
'category',
        'AvSigVersion':
'category',
        'IsBeta':
'int8',
        'RtpStateBitfield':
'float16',
        'IsSxsPassiveMode':
'int8',
        'DefaultBrowsersIdentifier':
'float32',
        'AVProductStatesIdentifier':
'float32',
        'AVProductsInstalled':
'float16',
        'AVProductsEnabled':
'float16',
        'HasTpm':
'int8',
        'CountryIdentifier':
'int16',
```

Рисунок 2.2 – Типи даних

Завантаження даних відбувається з використанням заданих типів (рис 2.3).

```
train = pd.read_csv('../input/train.csv', dtype=dtypes)
```

Рисунок 2.3 – Завантаження даних

Почнемо розвідувальних аналіз з побудови таблиці з інформацією про кількість унікальних та пропущених значень. Тут і далі, як основа використовується код одного з учасників Kaggle [12].

Код для побудови цієї таблиці продемонстровано на рисунку 2.4.

```
stats = []
for col in train.columns:
    stats.append((col, train[col].nunique(), train[col].isnull().su
m() * 100 / train.shape[0], train[col].value_counts(normalize=True,
dropna=False).values[0] * 100, train[col].dtype))

stats_df = pd.DataFrame(stats, columns=['Feature', 'Unique_values',
'Percentage of missing values', 'Percentage of values in the bigges
t category', 'type'])
stats_df.sort_values('Percentage of missing values', ascending=Fals
e)
```

Рисунок 2.4 – Код для побудови таблиці

Результат виконання цього коду, а саме фрагмент таблиці, можна побачити на рисунку 2.5.

	Feature	Unique_values	Percentage of missing values	Percentage of values in the biggest category
28	PuaMode	2	99.974119	99.974119
41	Census_ProcessorClass	3	99.589407	99.589407
8	DefaultBrowsersIdentifier	1730	95.141637	95.141637
68	Census_IsFlightingInternal	2	83.044030	83.044030
52	Census_InternalBatteryType	78	71.046809	71.046809
71	Census_ThresholdOptIn	2	63.524472	63.524472
75	Census_IsWIMBootEnabled	2	63.439038	63.439038
31	SmartScreen	21	35.610795	48.379658
15	OrganizationIdentifier	49	30.841487	47.037662
29	SMode	2	6.027686	93.928812

Рисунок 2.5 – Фрагмент побудованої таблиці

Аналізуючи цю таблицю можемо зробити кілька цікавих висновків:

– «PuaMode» та «Census_ProcessorClass» мають майже 100% пропущених значень;

– в колонці «DefaultBrowsersIdentifier» 95% значень відносяться до одної категорії;

– всього 26 колонок, в яких одна категорія містить 90% значень.

Ці колонки не є збалансованими тому видалимо їх, використовуючи код, що зображений на рисунку 2.6, та виведемо дані, що залишились (рис. 2.7).

```
good_cols = list(train.columns)
for col in train.columns:
    rate = train[col].value_counts(normalize=True, dropna=False).va
lues[0]
    if rate > 0.9:
        good_cols.remove(col)
```

```
train = train[good_cols]
```

Рисунок 2.6 – Видалення незбалансованих даних

```
train.head()
```

	MachineIdentifier	EngineVersion	AppVersion	AvSigVersion	AVProductStatesIdentifier	AVProductsInstalled
0	0000028988387b115f69f31a3bf04f09	1.1.15100.1	4.18.1807.18075	1.273.1735.0	53447.0	1.0
1	000007535c3f730efa9ea0b7ef1bd645	1.1.14600.4	4.13.17134.1	1.263.48.0	53447.0	1.0
2	000007905a28d863f6d0d597892cd692	1.1.15100.1	4.18.1807.18075	1.273.1341.0	53447.0	1.0
3	00000b11598a75ea8ba1beea8459149f	1.1.15100.1	4.18.1807.18075	1.273.1527.0	53447.0	1.0
4	000014a5f00daa18e76b81417eeb99fc	1.1.15100.1	4.18.1807.18075	1.273.1379.0	53447.0	1.0

Рисунок 2.7 – Дані датасету

Перевіримо відношення кількості значень поля «HasDetections» (рис 2.8).

```
train['HasDetections'].value_counts()
```

```
0    4462591
1    4458892
Name: HasDetections, dtype: int64
```

Рисунок 2.8 – Відношення кількості значень поля «HasDetections»

Це поле є збалансованим, що в є оптимальним для побудови точної моделі прогнозування.

2.2 Реалізація розвідувального аналізу

Перейдемо до побудови графіків за допомогою бібліотеки Plotly. Перший графік побудуємо по кількості сенсорних та звичайних пристроїв та їх зараження шкідливими програмами (рис. 2.9).

```
plot_categorical_feature('Census_IsTouchEnabled', True)
```

```
Census_IsTouchEnabled has 2 unique values and type: int8.
0    0.874457
1    0.125543
Name: Census_IsTouchEnabled, dtype: float64
```

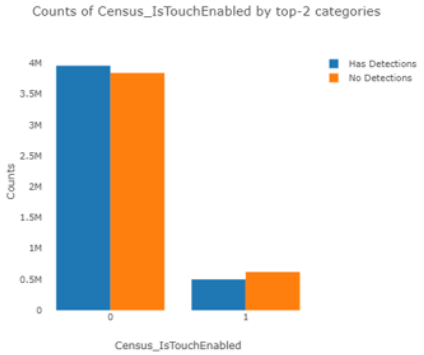


Рисунок 2.9 – Кількість значень поля «Census_IsTouchEnabled»

Як і очікувано Microsoft має набагато більше комп`ютерів, ніж сенсорних пристроїв, але і відсоток заражень вище у сенсорних пристроїв.

Наступний графік побудовано за значенями поля «EngineVersion», окремо для сенсорних (рис. 2.10) та звичайних пристроїв (рис. 2.11).

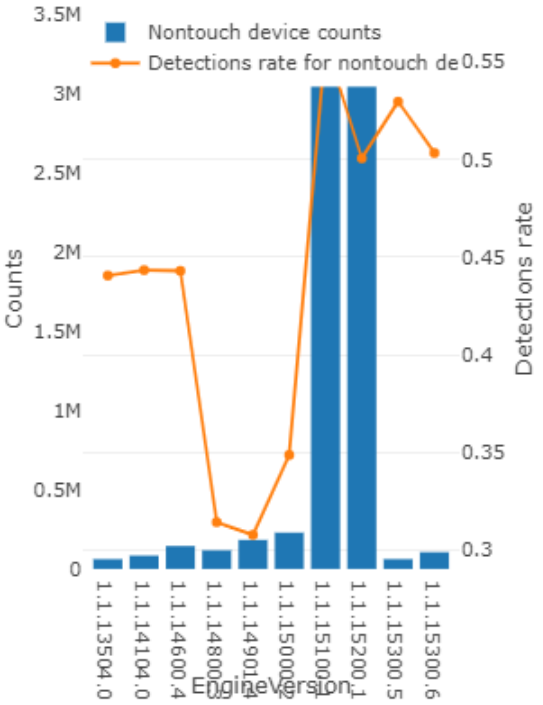


Рисунок 2.10 – Топ 10 категорій «EngineVersion» для сенсорних пристроїв

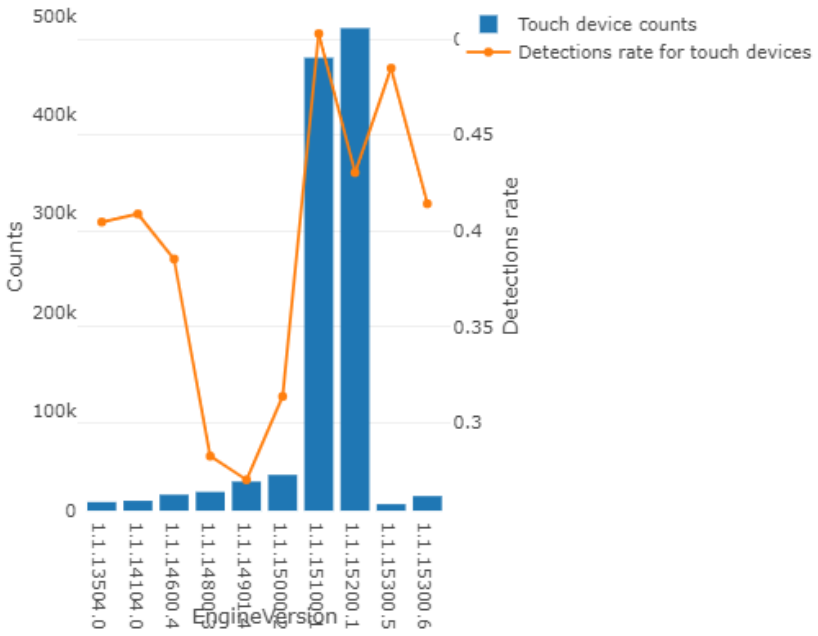


Рисунок 2.11 – Топ 10 категорій «EngineVersion» для звичайних пристроїв

Як ми бачимо, дві категорії мають 84% усіх значень. Різниця у відсотках зараження помітна. Інші категорії мають різні відсотки зараження, через меншу кількість записів у них.

Графік за полем «AvSigVersion» продемонстровано на рисунку 2.12.

```
plot_categorical_feature('AvSigVersion')
```

```
AvSigVersion has 8531 unique values and type: category.
1.273.1420.0    0.011469
1.263.48.0      0.010987
1.275.1140.0    0.010899
1.275.727.0     0.010362
1.273.371.0     0.009748
Name: AvSigVersion, dtype: float64
```

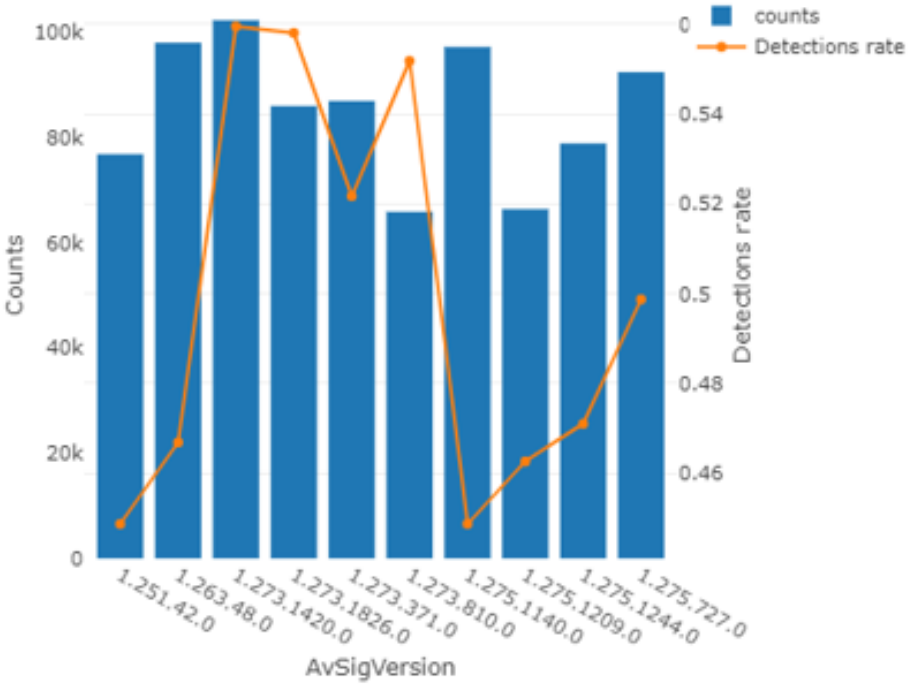


Рисунок 2.12 – Топ 10 категорій «AvSigVersion»

Це поле містить дуже велику кількість категорій, що означає часту зміну цього програмного забезпечення.

Графік за полем «AVProductsInstalled» продемонстровано на рисунку 2.13.

```
plot_categorical_feature('AVProductsInstalled', True)
```

```
AVProductsInstalled has 8 unique values and type: float16.
 1.0    0.695949
 2.0    0.275628
 3.0    0.023326
NaN     0.004060
 4.0    0.000982
Name: AVProductsInstalled, dtype: float64
```

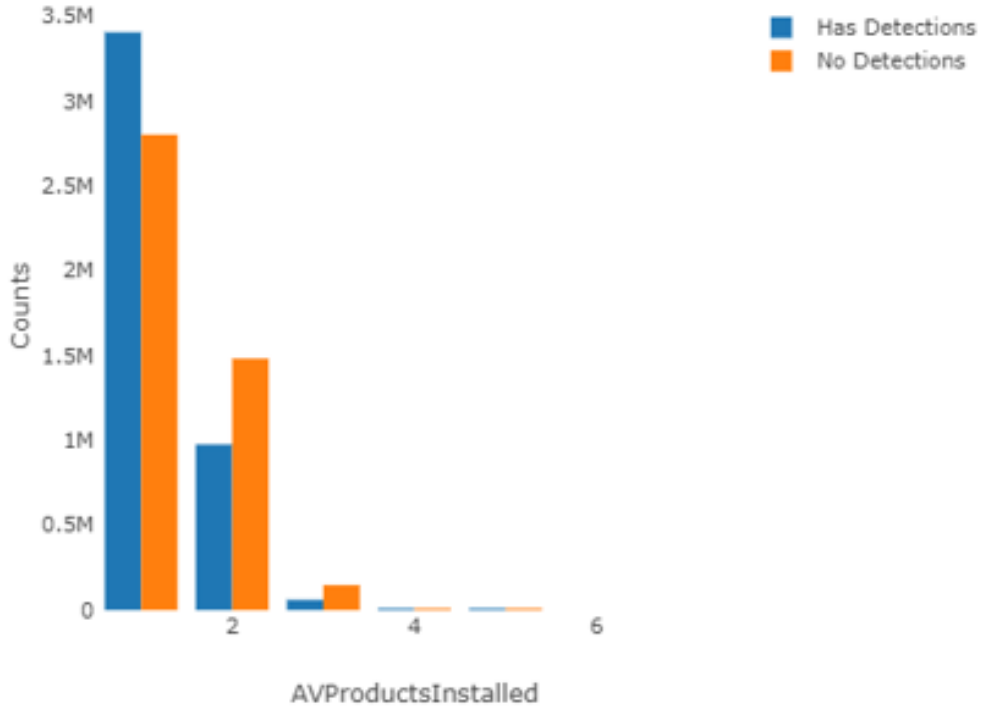


Рисунок 2.13 – Топ 8 категорій «AVProductsInstalled»

З цього графіку можна зробити цікавий висновок, якщо комп'ютер має один антивірус, то можливість зараження зменшується, але якщо два, то збільшується. Усі інші категорії мають мало записів, тому поєднаємо їх (рис. 2.14).

```
train.loc[train['AVProductsInstalled'].isin([1, 2]) == False, 'AVProductsInstalled'] = 3
test.loc[test['AVProductsInstalled'].isin([1, 2]) == False, 'AVProductsInstalled'] = 3
```

Рисунок 2.14 – Об`єднання усіх категорій, крім 1 та 2

Графік за полем «CountryIdentifier» продемонстровано на рисунку 2.15.

```
plot_categorical_feature('CountryIdentifier', True, 20)
```

```
CountryIdentifier has 222 unique values and type: int16.
43     0.044519
29     0.039006
141    0.037372
93     0.031791
171    0.031449
Name: CountryIdentifier, dtype: float64
```

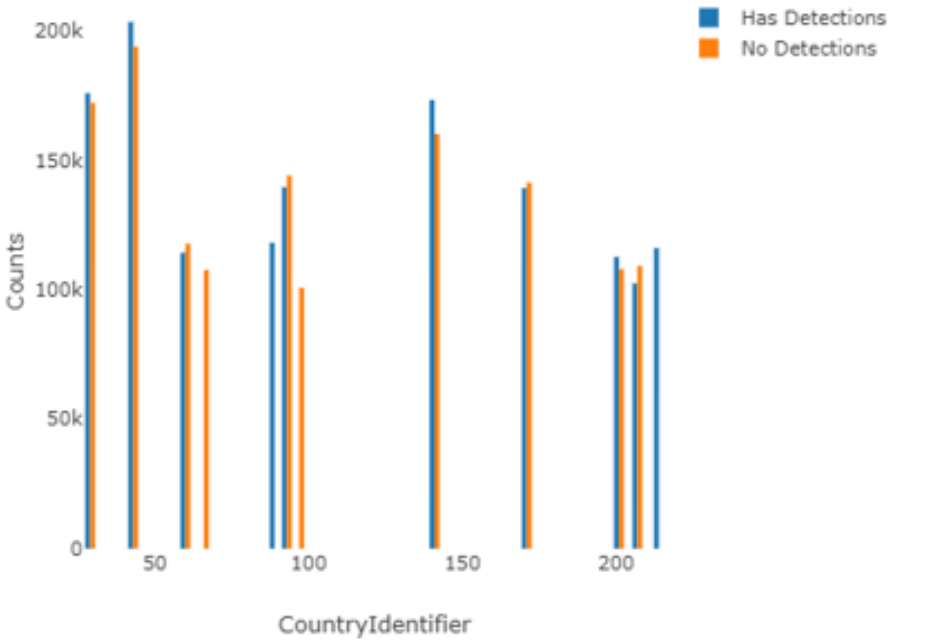


Рисунок 2.15 – Топ 20 категорій «CountryIdentifier»

Значення цього поля мають невірний тип даних, тому замінимо його на category (рис. 2.16). Більшість країн мають ~50% відсотків заражень.

```
train['CountryIdentifier'] = train['CountryIdentifier'].astype('category')
test['CountryIdentifier'] = test['CountryIdentifier'].astype('category')
```

Рисунок 2.16 – Зміна типу даних поля «CountryIdentifier»

Поле «CityIdentifier» має ту ж саму проблему. Графік продемонстровано на рисунку 2.17.

```
plot_categorical_feature('CityIdentifier', True, 20)
```

```
CityIdentifier has 107366 unique values and type: float32.
NaN          0.036475
 130775.0    0.010627
 16668.0     0.009503
 82373.0     0.009338
 10222.0     0.008050
Name: CityIdentifier, dtype: float64
```

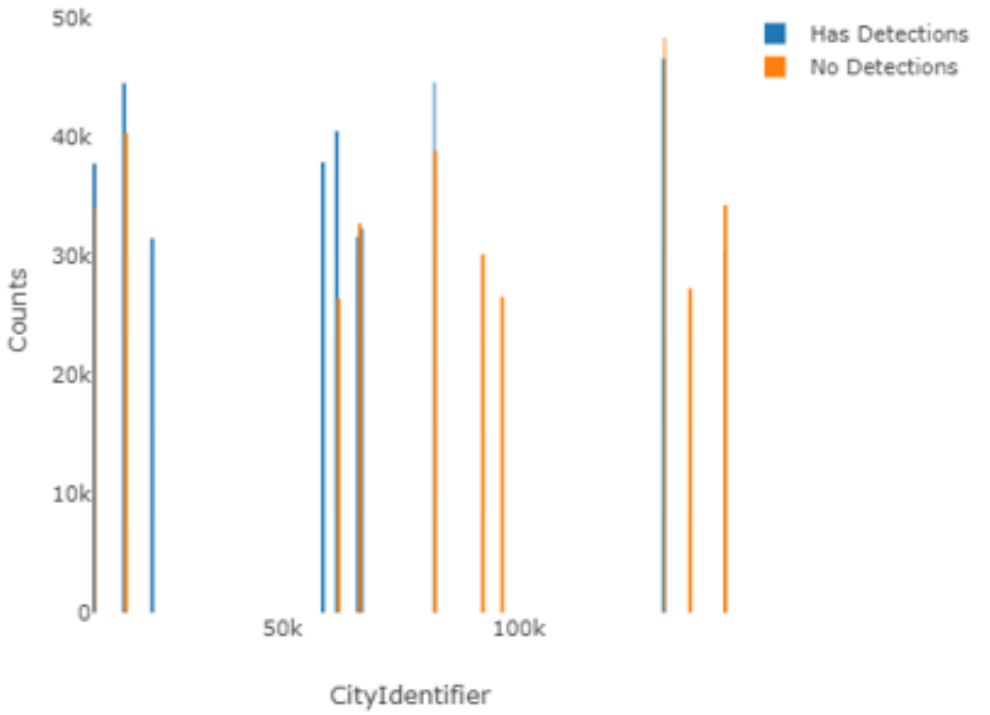


Рисунок 2.17 – Топ 20 категорій «CityIdentifier»

Змінимо тип даних (рис. 2.18).

```
train['CityIdentifier'] = train['CityIdentifier'].astype('category'
)
test['CityIdentifier'] = test['CityIdentifier'].astype('category')
```

Рисунок 2.18 – Зміна типу даних поля «CityIdentifier»

Графік за полем «OrganizationIdentifier» продемонстровано на рисунку 2.19.

```
plot_categorical_feature('OrganizationIdentifier', True)
```

```
OrganizationIdentifier has 49 unique values and type: float16.
 27.0    0.470377
NaN      0.308415
 18.0    0.197745
 48.0    0.007156
 50.0    0.005100
Name: OrganizationIdentifier, dtype: float64
```

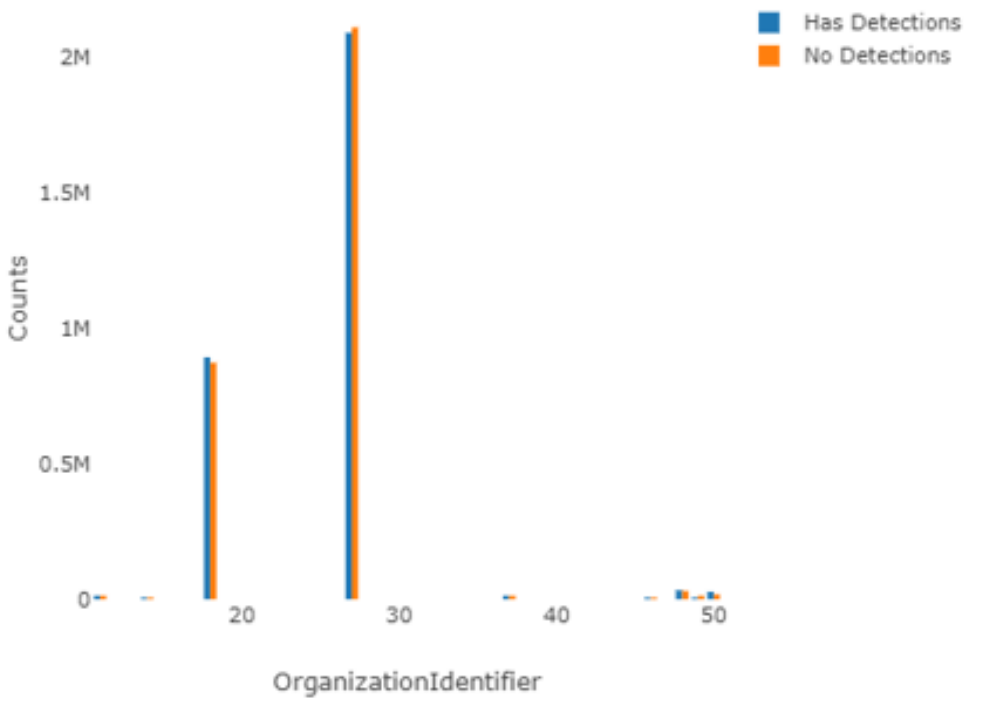


Рисунок 2.19 – Топ 10 категорій «OrganizationIdentifier»

Дві організації містять 66% усіх комп`ютерів, тому агрегуємо усі інші значення (рис 2.20).

```
train.loc[train['OrganizationIdentifier'].isin([27, 18]) == False, 'OrganizationIdentifier'] = 48
test.loc[test['OrganizationIdentifier'].isin([27, 18]) == False, 'OrganizationIdentifier'] = 48
```

Рисунок 2.20 – Агрегування категорії «OrganizationIdentifier»

Тепер побудуємо графіки, окремо для сенсорних (рис. 2.21) та звичайних пристроїв (рис. 2.22).

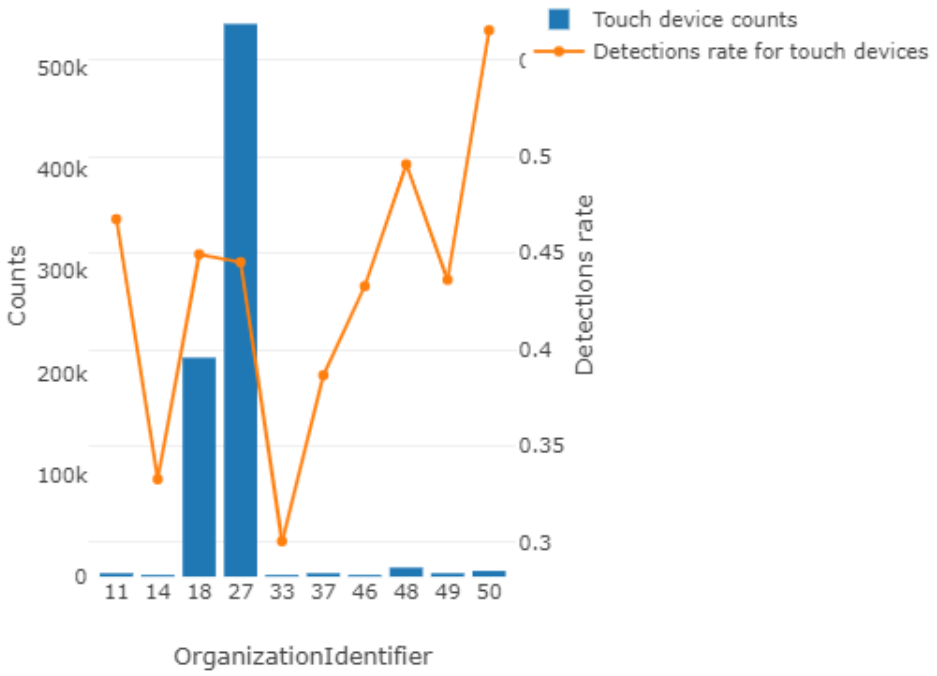


Рисунок 2.21 – Топ 10 категорій «OrganizationIdentifier» для сенсорних пристроїв

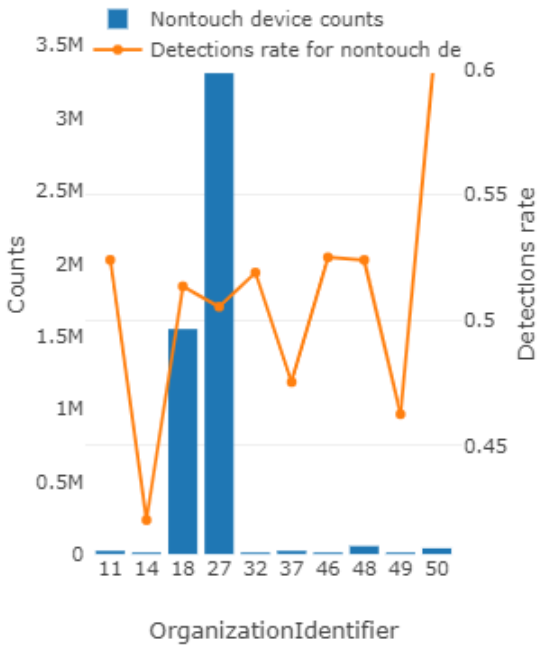


Рисунок 2.22 – Топ 10 категорій «OrganizationIdentifier» для звичайних пристроїв

Графік за полем «OsPlatformSubRelease», окремо для сенсорних (рис. 2.23) та звичайних пристроїв (рис. 2.24).

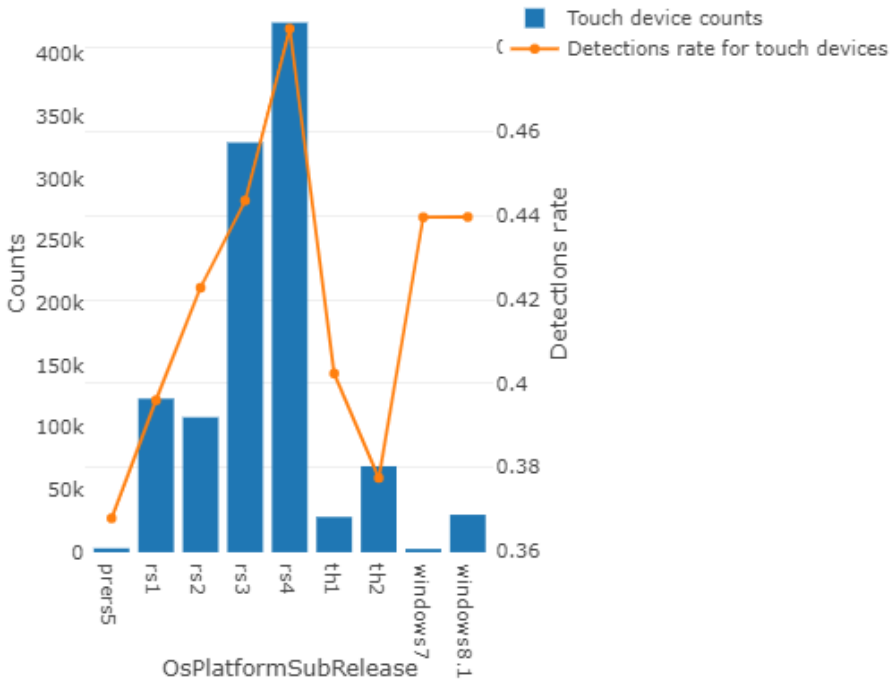


Рисунок 2.23 – Топ 10 категорій «OsPlatformSubRelease» для сенсорних пристроїв

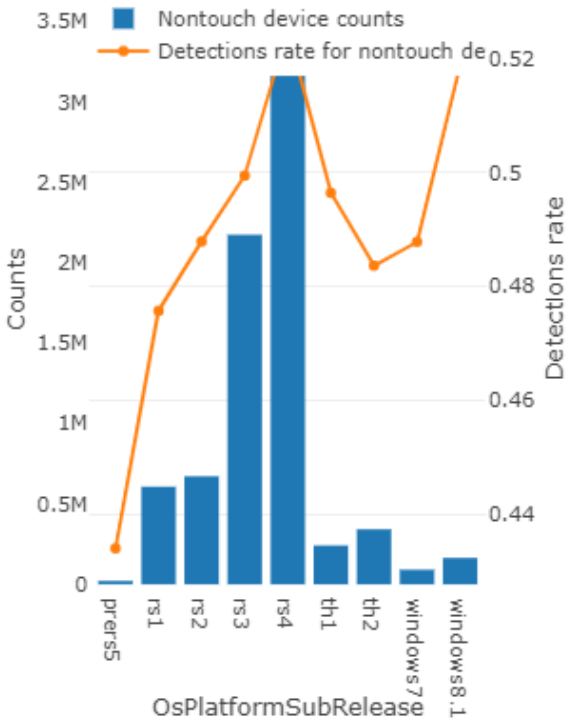


Рисунок 2.24 – Топ 10 категорій «OsPlatformSubRelease» для звичайних пристроїв

Більшість комп`ютерів мають операційну систему Windows 10. Графік за полем «Census_ProcessorCoreCount», окремо для сенсорних (рис. 2.25) та звичайних пристроїв (рис. 2.26).

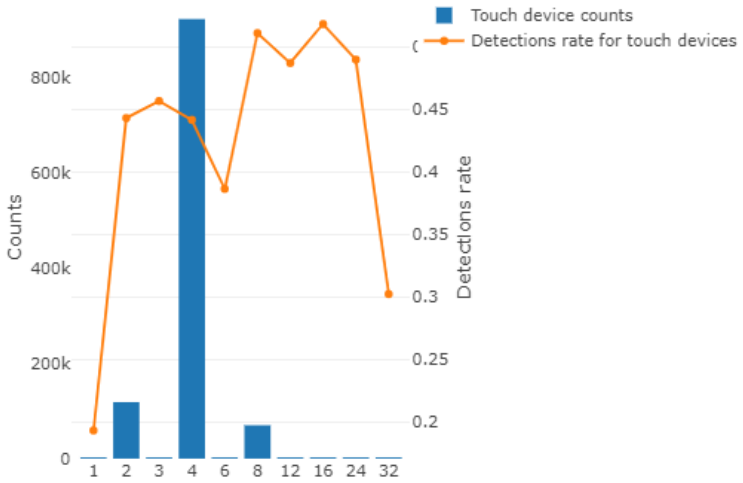


Рисунок 2.25 – Топ 10 категорій «Census_ProcessorCoreCount» для сенсорних пристроїв

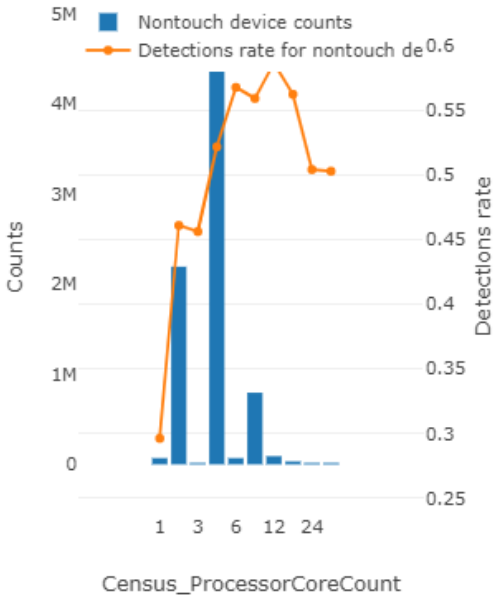


Рисунок 2.26 – Топ 10 категорій «Census_ProcessorCoreCount» для звичайних пристроїв

З цих графіків видно, що більшість комп'ютерів мають 2, 4 або 8 ядер. Для сенсорних пристроїв найбільш популярною є конфігурація з 4 ядрами. Ці три варіанти значень містяться у 95% усіх записів.

Графік за полем «Census_TotalPhysicalRAM», окремо для сенсорних (рис. 2.27) та звичайних пристроїв (рис. 2.28).

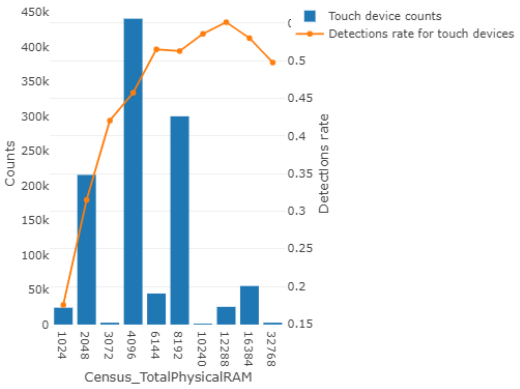


Рисунок 2.27 – Топ 10 категорій «Census_TotalPhysicalRAM» для сенсорних пристроїв

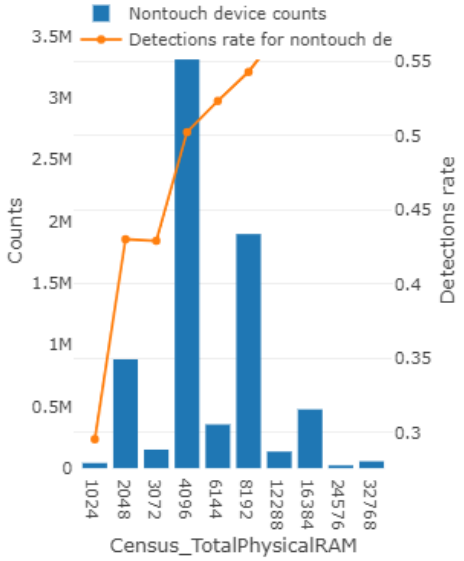


Рисунок 2.28 – Топ 10 категорій «Census_TotalPhysicalRAM» для звичайних пристроїв

Більшість комп`ютерів мають 8 і менше Гб оперативної пам`яті.

Також побудуємо графіки кореляції, за 10 полями в одному графіку (рис 2.29).

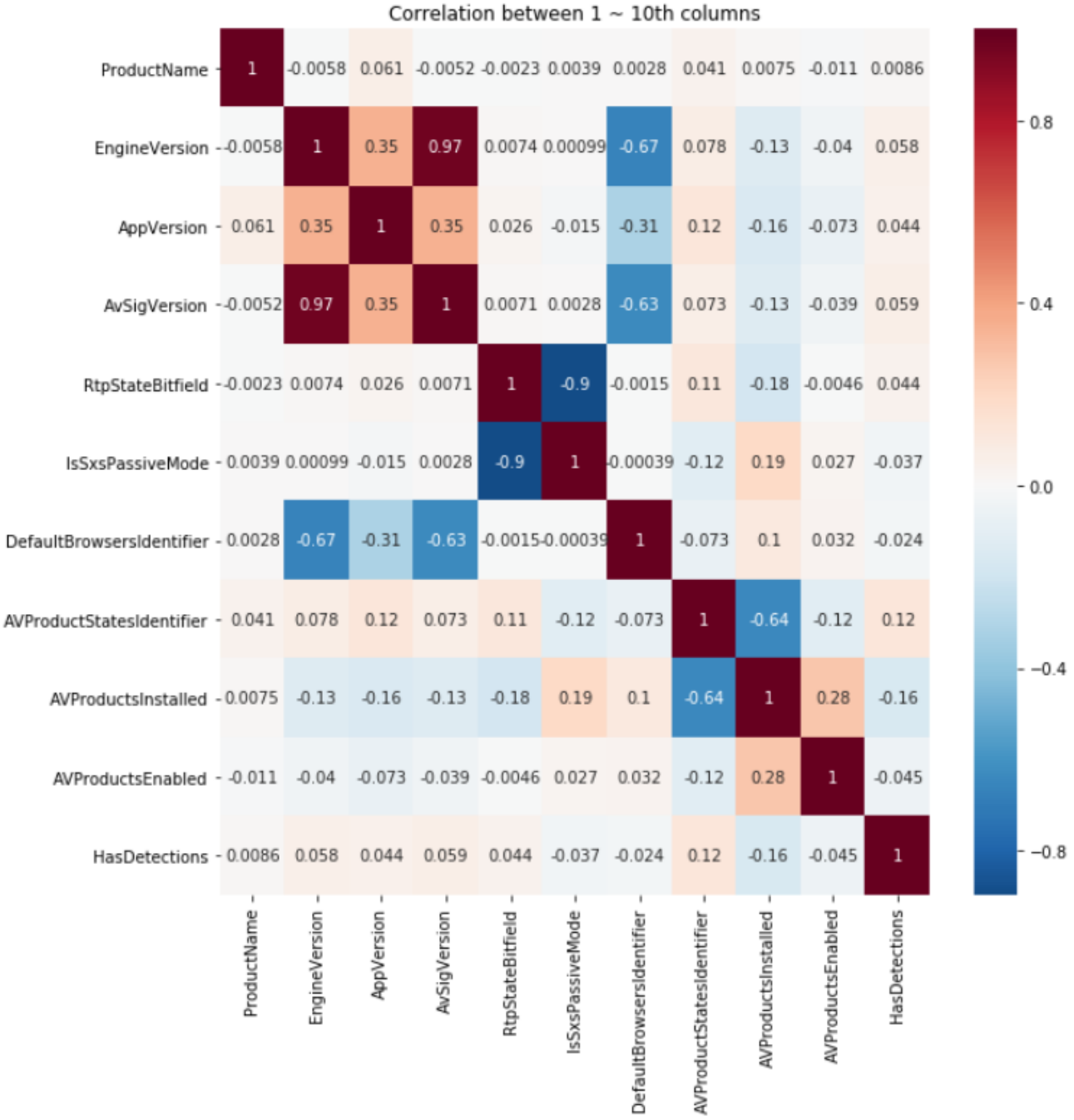


Рисунок 2.29 – Кореляція між 1-10 полями

На рисунку 2.30 продемонстровано графік кореляції наступних 10 полей.

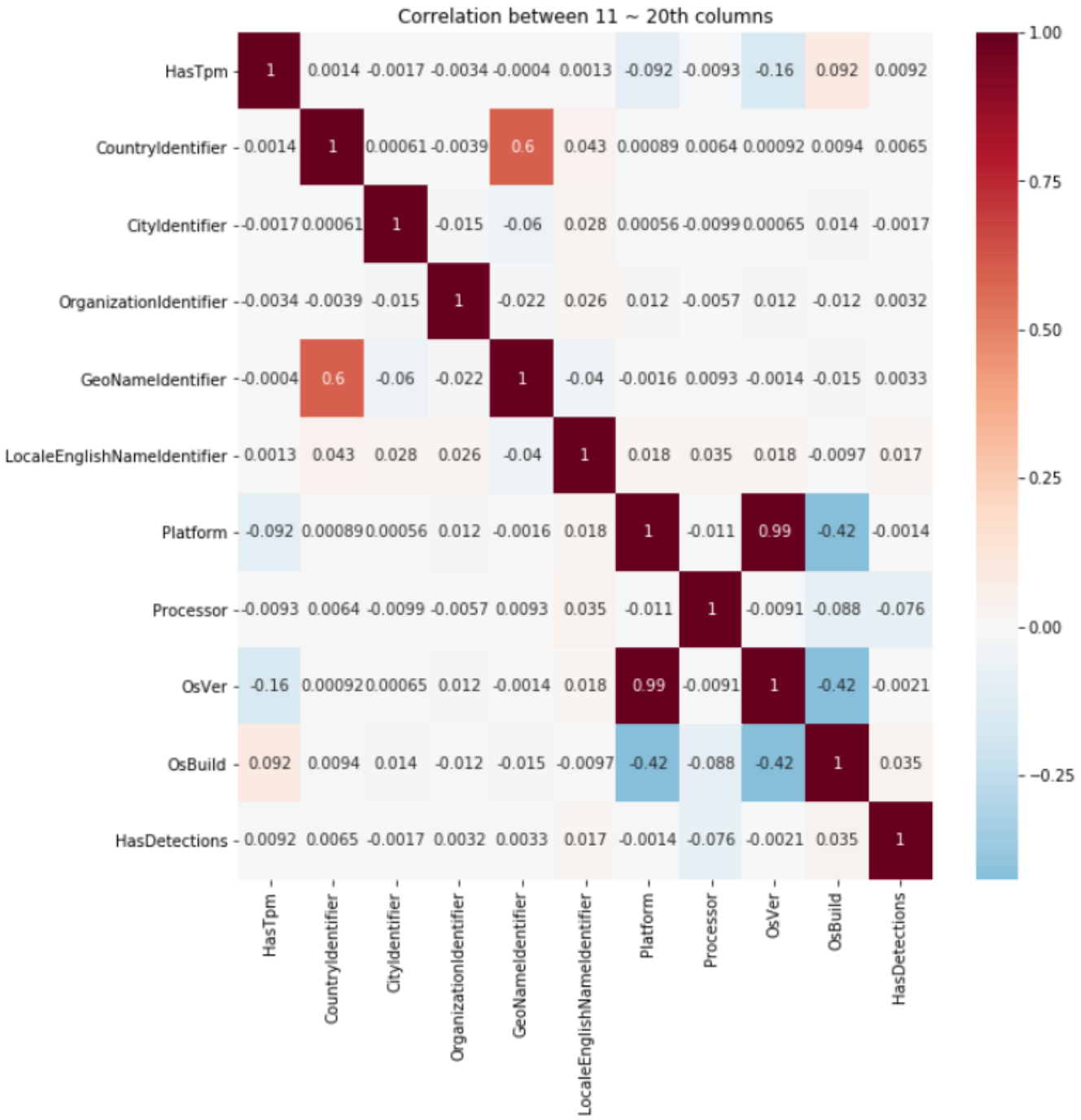


Рисунок 2.30 – Кореляція між 11-20 полями

Поля, що мають високу кореляцію можна видалити.

2.3 Результати аналізу

Проведений аналіз показав, що:

– «PuaMode» та «Census_ProcessorClass» мають майже 100% пропущених значень;

– в колонці «DefaultBrowsersIdentifier» 95% значень відносяться до одної категорії;

– всього 26 колонок, в яких одна категорія містить 90% значень;

– більшість комп`ютерів мають 8 і менше Гб оперативної пам`яті;

– більшість комп'ютерів мають 2, 4 або 8 ядер. Для сенсорних пристроїв найбільш популярною є конфігурація з 4 ядрами;

– більшість комп'ютерів мають операційну систему Windows 10;

– якщо комп'ютер має один антивірус, то можливість зараження зменшується, але якщо два, то збільшується;

– деякі поля мають невірний тип даних;

– набагато більше комп`ютерів, ніж сенсорних пристроїв, але і відсоток заражень вище у сенсорних пристроїв;

– поля, що мають високу кореляцію з іншими та незбалансовані поля можна видалити.

На основі проведеного розвідувального аналізу, побудуємо модель об`єкта.

РОЗДІЛ 3 ПОБУДОВА МОДЕЛІ ПРОГНОЗУВАННЯ ЙМОВІРНОСТІ ЗАРАЖЕННЯ КОМП'ЮТЕРІВ ВІРУСАМИ

3.1 Опис існуючих методів

Існує велика кількість підходів та методів побудови моделей об`єктів, у тому числі з використанням методів машинного навчання:

- дерева рішень;
- нейронні мережі;
- регресії.

Дерево прийняття рішень (також можуть називатися деревами класифікацій або регресійними деревами) – використовується в галузі статистики та аналізу даних для прогнозних моделей. Структура дерева містить такі елементи: «листя» і «гілки». На ребрах («гілках») дерева ухвалення рішення записані атрибути, від яких залежить цільова функція, в «листі» записані значення цільової функції, а в інших вузлах – атрибути, за якими розрізняються випадки. Щоб класифікувати новий випадок, треба спуститися по дереву до листа і видати відповідне значення. Подібні дерева рішень широко використовуються в інтелектуальному аналізі даних. Мета полягає в тому, щоб створити модель, яка прогнозує значення цільової змінної на основі декількох змінних на вході.

Кожен лист являє собою значення цільової змінної, зміненої в ході руху від кореня по листа. Кожен внутрішній вузол відповідає одній з вхідних змінних. Дерево може бути також «вивчено» поділом вихідних наборів змінних на підмножини, що засновані на тестуванні значень атрибутів. Це процес, який повторюється на кожному з отриманих підмножин. Рекурсія завершується тоді, коли підмножина в вузлі має ті ж значення цільової змінної, таким чином, воно не додає цінності для пророкувань. Процес, що йде «згори донизу», індукція дерев рішень (TDIDT), є прикладом поглинаючого «жадібного» алгоритму, і на сьогодні

є найбільш поширеною стратегією дерев рішень для даних, але це не єдина можлива стратегія [13].

Штучні нейронні мережі (ШНМ, англ. artificial neural networks, ANN), або конективістські системи (англ. connectionist systems) – це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють.

ШНМ ґрунтується на сукупності з'єднаних вузлів, що називають штучними нейронами (аналогічно до біологічних нейронів у головному мозку тварин). Кожне з'єднання (аналогічне синапсові) між штучними нейронами може передавати сигнал від одного до іншого. Штучний нейрон, що отримує сигнал, може обробляти його, й потім сигналізувати штучним нейронам, приєднаним до нього [14].

Регресійний аналіз – розділ математичної статистики, присвячений методам аналізу залежності однієї величини від іншої. На відміну від кореляційного аналізу не з'ясовує чи істотний зв'язок, а займається пошуком моделі цього зв'язку, вираженої у функції регресії.

Регресійний аналіз використовується в тому випадку, якщо відношення між змінними можуть бути виражені кількісно у виді деякої комбінації цих змінних. Отримана комбінація використовується для передбачення значення, що може приймати цільова (залежна) змінна, яка обчислюється на заданому наборі значень вхідних (незалежних) змінних. У найпростішому випадку для цього

використовуються стандартні статистичні методи, такі як лінійна регресія. На жаль, більшість реальних моделей не вкладаються в рамки лінійної регресії. Наприклад, розміри продажів чи фондові ціни дуже складні для передбачення, оскільки можуть залежати від комплексу взаємозв'язків множин змінних. Таким чином, необхідні комплексні методи для передбачення майбутніх значень [15].

Оскільки в даному дослідженні стоїть задача пошуку закономірностей для багатовимірного випадку то доцільно використовувати дерева рішень.

У свою чергу, дерева рішень будуються за такими методами:

– бустинг;
– багінг;
– стекінг.

Бустинг – це ансамблевий мета-алгоритм машинного навчання передусім для зменшення зсуву а також і дисперсії у навчанні з учителем, та сімейство алгоритмів машинного навчання, які перетворюють слабких учнів на сильних. Хоча бустинг й не є обмеженим алгоритмічно, більшість алгоритмів бустингу складаються з ітеративного навчання слабких класифікаторів стосовно розподілу та додавання їх до кінцевого сильного класифікатора. Коли вони додаються, вони, як правило, зважуються певним чином, зазвичай пов'язаним з точністю слабких учнів. Після додавання слабкого учня дані перезаважуються: неправильно класифіковані приклади набирають ваги, а класифіковані правильно вагу втрачають (деякі алгоритми підсилювання насправді зменшують вагу повторювано неправильно класифікованих прикладів, наприклад, підсилювання більшістю та BrownBoost). Таким чином, майбутні слабкі учні більше зосереджуються на тих прикладах, які попередні слабкі учні класифікували неправильно [16].

Багінг – це мета-алгогритм композиційного навчання, призначений для поліпшення стабільності і точності алгоритмів машинного навчання, що використовуються в статистичній класифікації та регресії. Алгоритм також зменшує дисперсію і допомагає уникати перенавчання. Хоча він зазвичай

застосовується до методів навчання машин на основі дерев рішень, його можна використовувати з будь-яким видом методу [17].

Стекінг (Stacked Generalization або Stacking) – один з найпопулярніших способів ансамблювання алгоритмів, тобто. використання декількох алгоритмів для вирішення однієї з задач машинного навчання. Він використовується для об'єднання інформації з декількох прогностичних моделей для створення нової моделі. Часто укладені моделі (модель другого рівня) перевершують кожну з окремих моделей завдяки своєму згладжуючому характеру і здатності виділяти кожну базову модель, де вона найкраще виконує роботу, і дискредитувати кожну базову модель, де вона працює погано. З цієї причини стекінг є найбільш ефективним, коли базові моделі значно відрізняються [18].

3.2 Опис обраного методу

Найбільш ефективним з них, як відомо, є бустинг. Для автоматизації бустингу існують дві найбільш потужні бібліотеки (мовою Python):

– LightGBM;
– XGBoost.

LightGBM – це бібліотека градієнтного бустингу, що використовує алгоритми дерев рішень [19]. Переваги бібліотеки:

– висока швидкість навчання та висока ефективність: Light GBM використовує алгоритм на основі гістограми, тобто він збирає безперервні значення функцій у дискретні контейнери, які прискорюють процедуру навчання.

– низьке використання пам`яті: замінює безперервні значення на дискретні контейнери, що призводить до меншого використання пам’яті.

– висока точність: він створює набагато складніші дерева, дотримуючись підходу з розділенням листків, а не підходу за рівнями, який є основним фактором для досягнення вищої точності. Однак іноді це може призвести до переобладнання, якого можна уникнути, встановивши параметр max_depth.

– підтримка паралельного навчання та навчання з використанням графічного процесора;

– сумісність з великими наборами даних: він здатний однаково добре працювати з великими наборами даних зі значним скороченням часу навчання в порівнянні з XGBOOST.

XGBoost – оптимізована бібліотека екстремального градієнтного підсилення, розроблена бути високоефективною, гнучкою та портативною. Вона реалізує алгоритми машинного навчання на основі бібліотеки градієнтного бустингу(Gradient Boosting). XGBoost забезпечує паралельне підсилення дерев (також відоме як GBDT, GBM), яке дозволяє швидко і точно вирішити багато проблем машинного навчання.

LightGBM використовує нову техніку односторонньої вибірки на основі градієнта (GOSS) для фільтрації екземплярів даних для пошуку значення розділення, тоді як XGBoost використовує попередньо відсортований алгоритм і алгоритм на основі гістограми для обчислення найкращого розподілу. Тут прикладами є спостереження/зразки.

Простіше кажучи, алгоритм, заснований на гістограмі, розбиває всі точки даних для об’єкта на дискретні групи і використовує ці бірки для пошуку розділеного значення гістограми. Незважаючи на те, що він є ефективнішим, ніж попередньо відсортований алгоритм у швидкості навчання, який перераховує всі можливі точки розділення на попередньо відсортованих значеннях ознак, він все ще відстає від GOSS з точки зору швидкості. Але що робить цей метод GOSS ефективним?

GOSS (Gradient Based One Side Sampling) — це новий метод вибірки, який зменшує вибірку екземплярів на основі градієнтів. Як ми знаємо, екземпляри з малими градієнтами добре тренуються (невелика помилка навчання), а з великими градієнтами — недостатньо. Наївний підхід до зниження вибірки полягає в тому, щоб відкинути екземпляри з малими градієнтами, зосередившись виключно на екземплярах з великими градієнтами, але це змінило б розподіл даних. У двох

словах, GOSS зберігає екземпляри з великими градієнтами, виконуючи випадкову вибірку для екземплярів з малими градієнтами.

На відміну від порівневого (горизонтального) зростання в XGBoost, LightGBM здійснює полистове (вертикальне) зростання, що призводить до більшого зменшення втрат і, в свою чергу, більшої точності, а також швидкості. Але це також може призвести до переобладнання навчальних даних, які можуть бути оброблені за допомогою параметра max-depth, який вказує, де відбудеться поділ.

Отже, XGBoost здатний створювати більш надійні моделі, ніж LightGBM. Однак XGBoost вимагає багато ресурсів для навчання великих обсягів даних, що робить його доступним варіантом для більшості підприємств, тоді як LightGBM є легким і може використовуватися на скромному обладнанні.

Обидва алгоритми встановили золотий стандарт з точки зору продуктивності вихідної моделі, і користувач повністю залежить від вибору в першу чергу на основі природи категорійних функцій і розміру даних. Враховуючи великий розмір датасету (4.2 Гб), велику кількість ознак (82 шт.) та обмежені обчислювальні можливості (оперативна пам'ять: 8 Гб), пропонується застосовувати бібліотеку LightGBM.

3.3 Побудова алгоритму прогнозування даних

На рисунку 3.1 зображений алгоритм прогнозування даних.

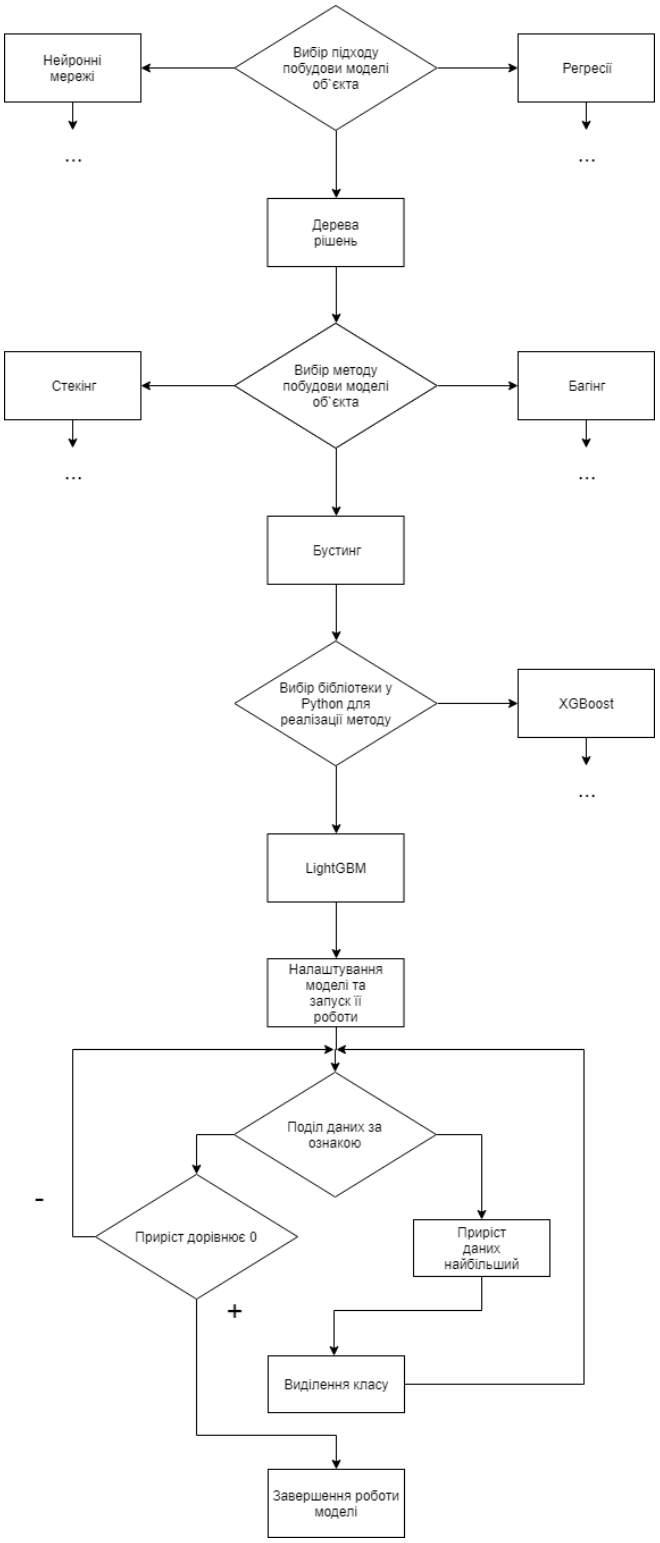


Рисунок 3.1 – Алгоритм прогнозування даних

3.4 Налаштування технічних засобів для навчання моделі

Для початку необхідно встановити бібліотеку LightGBM. За допомогою двох наступних запитів у консоль викликаємо встановлення потрібних програмних засобів:

pip install wheel

pip install lightgbm

Зібрана бібліотека, яка включена у wheel файл, підтримує як GPU, так і CPU версії з коробки. Ця функція є експериментальною і наразі доступна лише для Windows. Щоб використовувати версію GPU, вам потрібно лише встановити бібліотеки OpenCL Runtime. Для графічних процесорів NVIDIA і AMD вони включені в звичайні драйвери для відеокарти, тому не потрібно нічого робити. Якщо потрібно, щоб процесор AMD або Intel працював як графічний процесор (для тестування та налагодження), можна встановити AMD APP SDK.

Далі необхідно зібрати бібліотеку у бінарні файли з вихідних файлів. Для цього потрібно викликати команду:

pip install --no-binary :all: lightgbm

Для отримання найкращої оптимальної моделі потрібно налаштувати параметри бібліотеки LightGBM перед початком роботи з неї. Щоб отримати найкращу підгонку, необхідно налаштувати наступні параметри:

– num_leaves: оскільки LightGBM росте по листу, це значення має бути менше *2^(max_depth)*, щоб уникнути сценарію переобладнання

– min_data_in_leaf: для великих наборів даних його значення має бути встановлено від сотень до тисяч

– max_depth: ключовий параметр, значення якого має бути встановлено відповідним чином, щоб уникнути переобладнання

– max_bin: високе значення матиме великий вплив на точність, але в кінцевому підсумку призведе до переобладнання

Розглянемо також і інші корисні параметри, які надає нам бібліотека LightGBM:

– min_child_samples: мінімальна кількість зразків (даних) для групування в листок. Цей параметр може значно допомогти при переобладнанні: більші розміри вибірки на лист зменшить переобладнання (але може призвести до недостатнього підгонки)

Налаштування на незбалансовані дані. Найпростіший спосіб врахувати незбалансовані або спотворені дані - додати ваги до позитивних прикладів класу:

– scale_pos_weight: вагу можна розрахувати на основі кількості негативних і позитивних прикладів: sample_pos_weight = кількість негативних зразків / кількість позитивних зразків.

Тюнінг для переобладнання. На додаток до параметрів, згаданих вище, для контролю переобладнання можна використовувати такі параметри:

– min_child_weight: мінімальна сума гессена для листа. У поєднанні з min_child_samples більші значення зменшують переобладнання

– bagging_fraction і bagging_freq: вмикає пакетування (підвибірку) даних навчання. Обидва значення мають бути встановлені для використання пакетів. Частота контролює, як часто (ітераційно) використовується пакетування. Менші фракції та частоти зменшують переобладнання

– feature_fraction: контролює підвибірку функцій, які використовуються для навчання (на відміну від підвибірки фактичних даних навчання у випадку пакетування). Менші фракції зменшують переобладнання

– lambda_l1 і lambda_l2: контролює регуляризацію L1 і L2

Точність можна підвищити, налаштувавши такі параметри:

– learning_rate: для підвищення точності можна використовувати меншу швидкість навчання та більшу кількість ітерацій

– num_leaves: більша кількість листків підвищує точність з високим ризиком переобладнання

Для навчання моделі були встановлені наступні параметри:

– num_leaves: 256

– min_data_in_leaf: 42

- objective: binary
- max_depth: 5
- learning_rate: 0.05
- boosting: gbdt
- feature_fraction: 0.8
- bagging_freq: 5
- bagging_fraction: 0.8
- bagging_seed: 11
- lambda_l1: 0.15
- lambda_l2: 0.15
- random_state: 42
- verbosity: -1

3.5 Результат прогнозування даних

Розроблена програма, яка розв’язує задачу прогнозування зараження комп'ютерів шкідливими програмами, мовою Python. Фрагменти коду цієї програми наведено у додатку А.

Приклад побудованого дерева рішень продемонстровано на рисунку 3.3

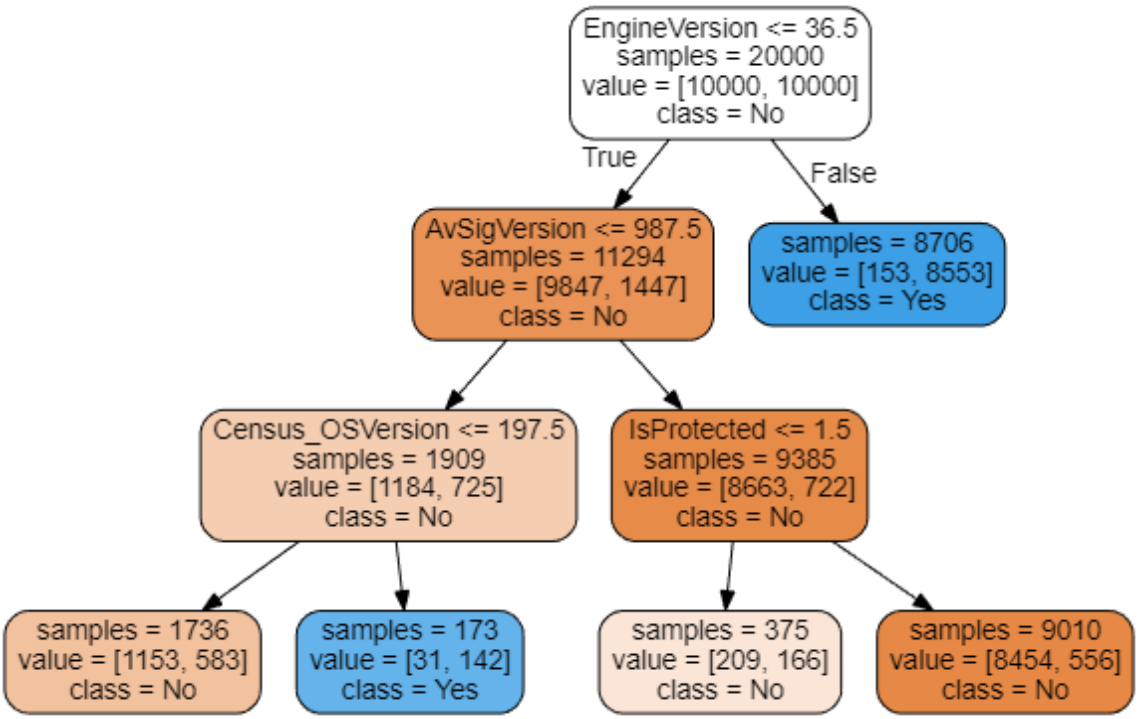


Рисунок 3.2 – Дерево рішень

Побудовано діаграму важливості ознак (рис. 3.3).

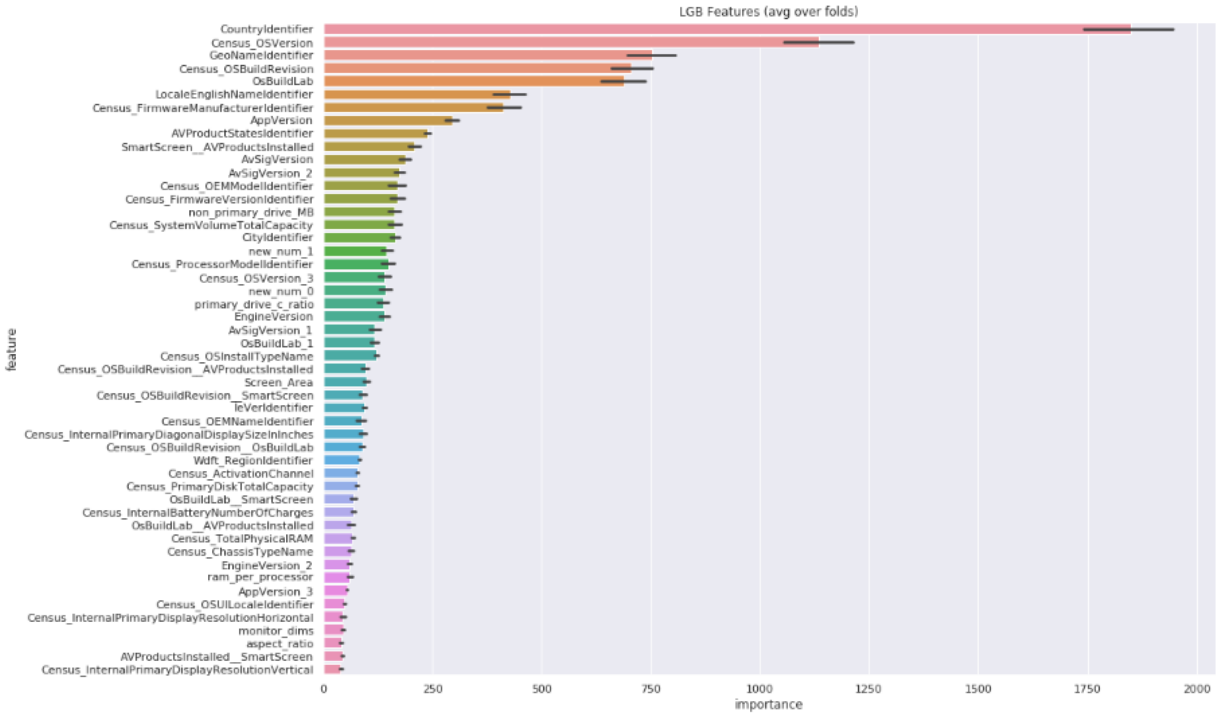


Рисунок 3.3 – Діаграма важливості ознак

Аналізуючи діаграму важливості ознак, можна зробити висновок, що найбільш впливовими ознаками, при побудові дерева рішень були:

- «CityIdentifier» – код міста, в якому розташований комп'ютер;
- «Census_OSVersion» – версія операційної системи;
- «GeoNameIdentifier» – код географічного регіону, в якому розташований комп`ютер;
- «Census_OSBuildRevision» – версія збірки операційної системи;
- «OsBuildLab» – лабораторія збірок, що розробила операційну систему.

Після того, як ми закінчили навчання нашої моделі, ми просто не можемо припустити, що вона буде добре працювати з даними, які вона раніше не бачила. Іншими словами, ми не можемо бути впевнені, що модель матиме бажану точність і дисперсію у виробничому середовищі. Нам потрібна певна впевненість у точності прогнозів, які дає наша модель. Для цього нам потрібно перевірити нашу модель. Цей процес прийняття рішення про те, чи чисельні результати, які кількісно визначають гіпотетичні зв’язки між змінними, прийнятні як опис даних, відомий як валідація.

Щоб оцінити продуктивність будь-якої моделі машинного навчання, нам потрібно перевірити її на деяких невидимих даних. Виходячи з продуктивності моделей на невидимих даних, ми можемо сказати, що наша модель недостатньо/надмірна/добре узагальнена.

Перехресна перевірка (Cross validation) — це одна з технік, яка використовується для перевірки ефективності моделей машинного навчання, а також процедура повторної вибірки, яка використовується для оцінки моделі, якщо у нас обмежені дані. Для виконання CV нам потрібно залишити в стороні зразок/частину даних, які не використовуються для навчання моделі, пізніше використати цей зразок для тестування/валідації.

Процедура має один параметр під назвою k, який відноситься до кількості груп, на які потрібно розділити вибірку даних. Тому цю процедуру часто

називають перехресною перевіркою k-кратів. Коли вибрано конкретне значення для k, воно може використовуватися замість k у посиланні на модель, наприклад, k=10 стає 10-кратною перехресною перевіркою.

Для оцінки навичок моделі в прикладному машинному навчанні на невидих даних в основному використовується метод перехресної перевірки. Щоб оцінити роботу моделі для прогнозування даних в цілому, використовується обмежене вибірка даних, яка не використовувалась під час навчання моделі. Це популярний метод, оскільки він простий для розуміння і тому, що він, як правило, призводить до менш упередженої або менш оптимістичної оцінки навичок моделі, ніж інші методи.

Загальна процедура виглядає наступним чином:

1. Перемішайте набір даних у випадковому порядку.
2. Розділіть набір даних на k груп
3. Для кожної унікальної групи:
    a. Прийміть групу як набір даних для затримки або тестування
    b. Візьміть решту груп як набір навчальних даних
    c. Встановіть модель на навчальний набір і оцініть її на тестовому наборі
    d. Збережіть оцінку та відкиньте модель
4. Підсумуйте вміння моделі, використовуючи зразок оціночних балів моделі

Важливо, що у вибірці даних кожне спостереження призначається окремій групі й залишається в цій групі протягом процедури. Це означає, що кожен зразок може бути використаний у наборі утримання 1 раз і використаний для навчання моделі k-1 разів.

Значення k потрібно ретельно вибирати для вибірки даних. Погано вибране значення k може призвести до неправильного уявлення про навички моделі. Наприклад, до оцінки з високою дисперсією (яка може сильно змінитися на основі

даних, що використовуються для відповідності моделі), або до високої упередженості , (наприклад, переоцінка майстерності моделі).

Нижче наведено три поширені тактики вибору значення k:

– Репрезентативне: значення k вибирається таким чином, щоб кожна група вибірок даних була достатньо великою, щоб стати статистично репрезентативною для більш широкого набору даних.

– k=10: значення k фіксується на 10, значення, яке було знайдено в результаті експериментів, як правило, призводить до оцінки навичок моделі з низьким зміщенням і помірною дисперсією.

– k=n: значення для k фіксується на n, де n – це розмір набору даних, щоб дати можливість кожному тестовому зразку використовуватися в набірі даних. Цей підхід називається перехресною перевіркою без виключення.

Вибір k зазвичай становить 5 або 10, але формального правила немає. Коли k стає більше, різниця в розмірі між навчальним набором і підмножинами повторної вибірки стає меншою. Оскільки ця різниця зменшується, зміщення техніки стає меншим [20].

Значення k=10 дуже поширене в області прикладного машинного навчання, і його рекомендують, якщо вам важко вибрати значення для свого набору даних.

```
            # feature importance
            fold_importance = pd.DataFrame()
            fold_importance["feature"] = X.columns
            fold_importance["importance"] = model.feature_importance()
            fold_importance["fold"] = fold_n + 1
            feature_importance = pd.concat([feature_importance, fold_importance], axis=0)

    prediction /= n_fold

    print('CV mean score: {0:.4f}, std: {1:.4f}.'.format(np.mean(scores), np.std(scores)))

    if model_type == 'lgb':

        if plot_feature_importance:
            feature_importance["importance"] /= n_fold
            cols = feature_importance[["feature", "importance"]].groupby("feature").mean().sort_
values(
                by="importance", ascending=False)[:50].index
```

Рисунок 3.4 – Перехресна перевірка навченої моделі

```
Scores from each Iteration:  [0.8878555024599691, 0.730675526203351
3, 0.8894245819790283, 0.5964287606247529, 0.717363346638312]
Average K-Fold Score : 0.7643495435810828
```

Рисунок 3.5 - Результат перевірки на адекватність

Побудована модель є досить адекватною.

РОЗДІЛ 4 РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ ПРОГНОЗУВАННЯ ЙМОВІРНОСТІ ЗАРАЖЕНОСТІ КОМП'ЮТЕРІВ

4.1 Вибір інструментів для реалізація інформаційного забезпечення прогнозування ймовірності зараженості комп'ютерів

Отриману навчену модель можна використати для передбачення можливості зараження пристроїв вірусними програмами та виявлення комп'ютерів, які мають найбільш вразливі до шкідливого програмного забезпечення. Для зручного її використання було вирішено створити застосунок, який буде самостійно збирати інформацію про технічні характеристики операційних систем, проводити аналіз та видавати результат прогнозування навченої раніше моделі.

Перед програмною були поставлені наступні технічні вимоги:

- кросплатформеність
- зручний та простий інтерфейс
- функціонал для відображення інформаційної довідки

Кросплатформенний тип програмного забезпечення може працювати на кількох обчислювальних платформах, наприклад, Android, iOS, Windows, Blackberry тощо. Такі програми не вимагають окремого кодування для кожної платформи, скоріше одноразове кодування створить основу для ефективної роботи програми на всіх платформах. Сьогодні це один із найпопулярніших методів у сфері розробки додатків [21].

Отже усі інструменти розробки, які будуть використовуватись для реалізації застосунку, повинні бути кросплатформеними. Нижче наведений загальний план розробки застосунку:

1. Експортування навченої моделі прогнозування
2. Розгортання моделі у новому програмному застосунку
3. Реалізація сбору та обробки даних комп'ютера
4. Створення зручного та простого інтерфейсу

Беручи до уваги зазначені вище критерії та план було вирішено використовувати раніше використані у роботі мову програмування Python (версія 3), фреймворк LightGBM для роботи з моделью, а також фреймворк Qt для розробки інтерфейсу застосунку.

Qt — це кросплатформний фреймворк, який зазвичай використовується як графічний набір інструментів, хоча він також дуже корисний у створенні програм CLI. Він працює на трьох основних настільних ОС, а також на мобільних ОС, таких як Symbian, Nokia Belle, Meego Harmattan, MeeGo або BB10, а також на вбудованих пристроях. Порти для Android (Necessitas) і iOS також знаходяться в розробці [22]. Також фреймворк Qt є кросплатформним набором інструментів реалізації засобів GUI для розробки різноманітних програмних додатків.

Qt позиціонує себе як фреймворк «один раз, компілюйте де завгодно». Розробникам необхідно лише створювати та підтримувати код програми лише на одній платформі, яка використовується для розробки, а вихідний продукт буде успішно збиратися на кожній іншій підтримуваній операційній системі.

Вибираючи ліцензію з відкритим кодом для свого проекту, ви сприяєте розробці безкоштовного програмного забезпечення з відкритим кодом, використовуючи Qt за будь-якою з наступних ліцензій: LGPL версія 3, GPL версія 2 і GPL версія 3 [23].

4.2 Розробка інформаційного забезпечення

Для початку необхідно зберегти навчену модель та завантажити її в новому програмному застосунку. Фреймворк LightGBM надає нам необхідний програмний функціонал, який реалізується у декілька команд. На рис. 4.1 зображено фрагмент відповідного коду:

```
import lightgbm as lgb

# export trained model
lgb.booster_.save_model('lgb_model.txt')

# load from model
bst = lgb.Booster(model_file='lgb_model.txt')
```

Рисунок 4.1 - Збереження та завантаження моделі

Далі необхідно реалізувати функціонал, який надаватиме можливість збирати технічну інформацію комп'ютерів. В мові програмування Python є модуль під назвою *platform*. За допомогою нього можна отримати певні дані, щодо операційної системи комп'ютерного пристрою. Однак він не надає усю необхідну інформацію для прогнозування ймовірності зараження. Тому додатково використаєм модуль *subprocess* [24], який надає можливість виклику консольний операційних команд та отримання результатів видачі. На рис. 4.2 зображено код, який збирає технічну інформацію відповідно до ОС:

```
[ ]  def get_systeminfo():
         # Command to run.
         command = ['systeminfo']

         os = platform.system()
         if os == "Linux":
             cmd = ['sudo lshw']
         elif os == "MacOS":
             cmd = ['system_profiler']

         # Run the commands and get the stdout.
         with subprocess.Popen(command, universal_newlines=True, stdout=subprocess.PIPE) as p:
             stdout, _ = p.communicate()

         return stdout
```

Рисунок 4.2 - Збір технічної інформації операційної системи

Відповідні дані парсятся, обробляються та подаються моделі для прогнозування. Програмну реалізацію наведено у Додатку В.

Для роботи з графічним інтерфейсом необхідно встановити фреймворк Qt п’ятої версії для мови програмування Python. Це легко робиться за допомогою виклику в консолі наступної команди: *pip install pyqt5*

Для дизайну та побудови графічного інтерфейсу був використано інтегроване середовище розробки Qt Creator. Воно надає інтегрований візуальний редактор, який проектує програми на основі віджетів у режимі Design. Інтеграція включає в себе управління проектами та завершення коду.

4.3 Практичне застосування інформаційного забезпечення

Програма складається з головного екрану. На ньому відображені усі комп’ютери під'єднані до однієї мережі, їх імена, статус аналізу кожного пристрою, прогрес бар аналізатора, кнопка для запуску прогнозування моделі, а також кнопка для відображення найвагоміших факторів, які впливають на ймовірність зараження комп’ютерів.

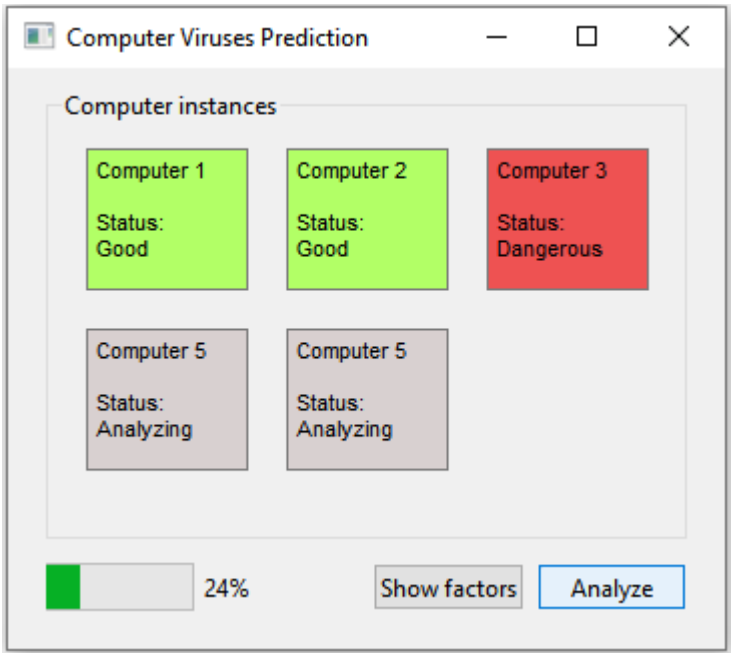


Рисунок 4.3 - Інтерфейс програми

ВИСНОВКИ

Із зростанням кількості користувачів персональних комп’ютерів та різноманітних пристроїв зростає і кількість шкідливого програмного забезпечення. Щороку світова економіка несе фінансові втрати в декілька десятків мільярдів доларів США. Однак з розвитком видів комп’ютерних вірусів галузь боротьби із ними постійно адаптується під нові різновиди.

В ході кваліфікаційної роботи магістра було описано предметну область, а саме проблему зараження комп'ютерів шкідливими програмами, їх типи, способи боротьби з ними. Було розглянуто залежність зараженості комп’ютерів вірусними програмами від технічних характеристик пристроїв. Також проаналізовано літературу в галузі комп’ютерних вірусів, визначено вхідні дані, розроблено алгоритми для обробки даних, проведено розвідувальний аналіз проблеми, з метою виявлення закономірностей та тенденцій, характеру та властивостей даних аналізу.

На основі розглянутих досліджень з прогнозування ймовірності зараження комп'ютерів вірусами, було вирішено застосувати методи машинного навчання, а саме дерева рішень. В роботі виділено найефективніший метод побудови дерев рішень – бустинг та обрано бібліотеку LightGBM у Python для побудови сформованого математичного апарату.

Перед розробки моделі була проведена попередня обробка вхідних даних, а саме видалення пустих значень та незбалансованих даних, нормалізація та типізація. Побудовано діаграму важливості ознак, основними є такі:

– «CityIdentifier» – код міста, в якому розташований комп'ютер;

– «Census_OSVersion» – версія операційної системи;

– «GeoNameIdentifier» – код географічного регіону, в якому розташований комп`ютер;

– «Census_OSBuildRevision» – версія збірки операційної системи;

– «OsBuildLab» – лабораторія збірок, що розробила операційну систему.

Проведено аналіз існуючих методів аналізу для розв’язання поставленої задачі та обґрунтовано, що оптимальним методом є методи бустингу бібліотеки Lightgbm з використання мови Python. Розроблено та наведено комп’ютерну програму, яка розв’язує поставлену задачі цим методом. Отримано похибку прийнятної точності. Здійснено передбачення ймовірність зараження персональних комп’ютерів із тестового набору заданого датасету. Модель було перевірено на адекватність за допомогою метода перехресної перевірки.

На основі навченої моделі був реалізований кросплатформений застосунок для аналізу комп’ютерів в межах локальної мережі для виявлення найуразливіших до вірусних програм. Додаток розроблявся за допомогою мови програмування Python, фреймворку LightGBM для роботи з моделью та фреймворку Qt для реалізації графічного інтерфейсу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Gerencer T. The Top 10 Worst Computer Viruses in History [Електронний ресурс] / Tom Gerencer // HP Tech. – 2020. – Режим доступу до ресурсу: https://www.hp.com/us-en/shop/tech-takes/top-ten-worst-computer-viruses-in-history.
2. Sausalito C. Cybercrime To Cost The World $10.5 Trillion Annually By 2025 [Електронний ресурс] / Calif Sausalito // Cybercrime Magazine. – 2020. – Режим доступу до ресурсу: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/.
3. Комп'ютерний вірус [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/комп%27ютерний_вірус.
4. Азіза Е. Компютерні віруси. Типи компютерних вірусів. Ознаки зараження вірусом. [Електронний ресурс] / Еліна Азіза. – 2018. – Режим доступу до ресурсу: http://www.myshared.ru/slide/1365037.
5. Rosenthal M. Must-Know Phishing Statistics: Updated 2022 [Електронний ресурс] / Maddie Rosenthal // Tessian. – 2022. – Режим доступу до ресурсу: https://www.tessian.com/blog/phishing-statistics-2020/.
6. Cohen F. Computational aspects of computer viruses / Fred Cohen., 1989. – 325 с.
7. Samuel, Arthur L. Some Studies in Machine Learning Using the Game of Checkers. I. Computer Games / Samuel, Arthur L. – Springer, New York, 1988. – с. 335–365.
8. Antivirus is dead: How AI and machine learning will drive cybersecurity [Електронний ресурс] – режим доступу: https://techbeacon.com/security/antivirus-dead-how-ai-machine-learning-will-drive-cybersecurity.
9. The pros, cons and limitations of AI and machine learning in antivirus software [Електронний ресурс] // Emisoft. – 2020. – Режим доступу до ресурсу:

https://blog.emsisoft.com/en/35668/the-pros-cons-and-limitations-of-ai-and-machine-learning-in-antivirus-software/.

10. Тьюки М. Анализ результатов наблюдений. Разведочный анализ / М. Тьюки.– Москва, 1981. – 697 с.
11. Load the Totality of the Data [Електронний ресурс] – режим доступу: https://www.kaggle.com/theoviel/load-the-totality-of-the-data.
12. Is this Malware? [EDA, FE and lgb] [Електронний ресурс] – режим доступу: https://www.kaggle.com/artgor/is-this-malware-eda-fe-and-lgb-updated.
13. Breiman L. Bagging Predictors. Machine Learning / L. Breiman., 1996. – с. 123–140.
14. McCulloch W. A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics / W. McCulloch, W. Pitts., 1943. – с. 115–133.
15. Freedman D. Statistical Models: Theory and Practice / David A. Freedman., 2009. – 128 с.
16. Zhi-Hua Z. Ensemble Methods: Foundations and Algorithms / Zhou Zhi-Hua., 2012. – 23 с.
17. Preimages for Variation Patterns from Kernel PCA and Bagging / A.Shinde, A. Sahu, D. Apley, G. Runger., 2014. – 46 с.
18. A Kaggler's Guide to Model Stacking in Practice [Електронний ресурс] – режим доступу: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/.
19. LightGBM’s documentation [Електронний ресурс] – режим доступу: https://lightgbm.readthedocs.io/en/latest/.
20. Browlee J. A Gentle Introduction to k-fold Cross-Validation [Електронний ресурс] / Jason Browlee. – 2018. – Режим доступу до ресурсу: https://machinelearningmastery.com/k-fold-cross-validation/.

21. Srivastava S. Cross-platform app development [Електронний ресурс] / Sudeep Srivastava. – 2022. – Режим доступу до ресурсу: https://appinventiv.com/blog/cross-platform-app-frameworks/.
22. Introduction to Qt [Електронний ресурс] // Qt Wiki – Режим доступу до ресурсу: https://wiki.qt.io/Qt_for_Beginners#Introduction_to_Qt.
23. Qt for Open Source Development [Електронний ресурс] // Qt Wiki – Режим доступу до ресурсу: https://www.qt.io/download-open-source
24. David M. How To Use subprocess to Run External Programs in Python 3 [Електронний ресурс] / Muller David // Dgigital ocean. – 2020. – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/how-to-use-subprocess-to-run-external-programs-in-python-3.

ДОДАТКИ

Додаток А

Код програми

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
import lightgbm as lgb
import xgboost as xgb
import time
import datetime
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold, KFold, TimeSeriesSplit
from sklearn.metrics import mean_squared_error, roc_auc_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
import gc
from tqdm import tqdm_notebook
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
```

```
import plotly.tools as tls

import warnings

warnings.filterwarnings("ignore")

import logging

logging.basicConfig(filename='log.txt',level=logging.DEBUG, format='%(asctime)s
%(message)s')

pd.set_option('max_colwidth', 500)

pd.set_option('max_columns', 500)

pd.set_option('max_rows', 100)

import os

print(os.listdir("../input/"))


def reduce_mem_usage(df, verbose=True):

    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']

numerical_columns = [c for c,v in dtypes.items() if v in numerics]

categorical_columns = [c for c,v in dtypes.items() if v not in numerics]

train = pd.read_csv('../input/train.csv', dtype=dtypes)

stats = []

for col in train.columns:

    stats.append((col, train[col].nunique(), train[col].isnull().sum() * 100 / train.shape[0],
train[col].value_counts(normalize=True, dropna=False).values[0] * 100,
train[col].dtype))

stats_df = pd.DataFrame(stats, columns=['Feature', 'Unique_values', 'Percentage of
missing values', 'Percentage of values in the biggest category', 'type'])

stats_df.sort_values('Percentage of missing values', ascending=False)

good_cols = list(train.columns)
```

```
for col in train.columns:
    rate = train[col].value_counts(normalize=True, dropna=False).values[0]
    if rate > 0.9:
        good_cols.remove(col)
train = train[good_cols]
test_dtypes = {k: v for k, v in dtypes.items() if k in good_cols}
test = pd.read_csv('../input/test.csv', dtype=test_dtypes, usecols=good_cols[:-1])
test.loc[6529507, 'OsBuildLab'] = '17134.1.amd64fre.rs4_release.180410-1804'
test['OsBuildLab'] =
test['OsBuildLab'].fillna('17134.1.amd64fre.rs4_release.180410-1804')
test = reduce_mem_usage(test)
def plot_categorical_feature(col, only_bars=False, top_n=10, by_touch=False):
    top_n = top_n if train[col].nunique() > top_n else train[col].nunique()
    print(f"{col} has {train[col].nunique()} unique values and type: {train[col].dtype}.")
    print(train[col].value_counts(normalize=True, dropna=False).head())
    if not by_touch:
        if not only_bars:
            df = train.groupby([col]).agg({'HasDetections': ['count', 'mean']})
            df = df.sort_values(('HasDetections', 'count'),
ascending=False).head(top_n).sort_index()
            data = [go.Bar(x=df.index, y=df['HasDetections']['count'].values,
name='counts'),
                    go.Scatter(x=df.index, y=df['HasDetections']['mean'], name='Detections
rate', yaxis='y2')]
            layout = go.Layout(dict(title = f"Counts of {col} by top-{top_n} categories and
mean target value",
                                    xaxis = dict(title = f'{col}',
```

```
                    showgrid=False,
                    zeroline=False,
                    showline=False,),
                yaxis = dict(title = 'Counts',
                    showgrid=False,
                    zeroline=False,
                    showline=False,),
                yaxis2=dict(title='Detections rate', overlaying='y', side='right')),
            legend=dict(orientation="v"))
    else:
        top_cat = list(train[col].value_counts(dropna=False).index[:top_n])
        df0 = train.loc[(train[col].isin(top_cat)) & (train['HasDetections'] == 1), col].value_counts().head(10).sort_index()
        df1 = train.loc[(train[col].isin(top_cat)) & (train['HasDetections'] == 0), col].value_counts().head(10).sort_index()
        data = [go.Bar(x=df0.index, y=df0.values, name='Has Detections'),
            go.Bar(x=df1.index, y=df1.values, name='No Detections')]
        layout = go.Layout(dict(title = f"Counts of {col} by top-{top_n} categories",
                xaxis = dict(title = f'{col}',
                    showgrid=False,
                    zeroline=False,
                    showline=False,),
                yaxis = dict(title = 'Counts',
                    showgrid=False,
                    zeroline=False,
                    showline=False,),
```

```
                ),
            legend=dict(orientation="v"), barmode='group')
    py.iplot(dict(data=data, layout=layout))
else:
    top_n = 10
    top_cat = list(train[col].value_counts(dropna=False).index[:top_n])
    df = train.loc[train[col].isin(top_cat)]
    df1 = train.loc[train['Census_IsTouchEnabled'] == 1]
    df0 = train.loc[train['Census_IsTouchEnabled'] == 0]
    df0_ = df0.groupby([col]).agg({'HasDetections': ['count', 'mean']})
    df0_ = df0_.sort_values(('HasDetections', 'count'),
ascending=False).head(top_n).sort_index()
    df1_ = df1.groupby([col]).agg({'HasDetections': ['count', 'mean']})
    df1_ = df1_.sort_values(('HasDetections', 'count'),
ascending=False).head(top_n).sort_index()
    data1 = [go.Bar(x=df0_.index, y=df0_['HasDetections']['count'].values,
name='Nontouch device counts'),
        go.Scatter(x=df0_.index, y=df0_['HasDetections']['mean'], name='Detections
rate for nontouch devices', yaxis='y2')]
    data2 = [go.Bar(x=df1_.index, y=df1_['HasDetections']['count'].values,
name='Touch device counts'),
        go.Scatter(x=df1_.index, y=df1_['HasDetections']['mean'], name='Detections
rate for touch devices', yaxis='y2')]
    layout = go.Layout(dict(title = f"Counts of {col} by top-{top_n} categories for
nontouch devices",
                xaxis = dict(title = f'{col}',
                    showgrid=False,
                    zeroline=False,
```

```
                    showline=False,
                    type='category'),
                yaxis = dict(title = 'Counts',
                    showgrid=False,
                    zeroline=False,
                    showline=False,),
                    yaxis2=dict(title='Detections rate', overlaying='y', side='right'),
                ),
                legend=dict(orientation="v"), barmode='group')
        py.iplot(dict(data=data1, layout=layout))
        layout['title'] = f"Counts of {col} by top-{top_n} categories for touch devices"
        py.iplot(dict(data=data2, layout=layout))
train['HasDetections'].value_counts()
train['OsBuildLab'] = train['OsBuildLab'].cat.add_categories(['0.0.0.0.0-0'])
train['OsBuildLab'] = train['OsBuildLab'].fillna('0.0.0.0.0-0')
test['OsBuildLab'] = test['OsBuildLab'].cat.add_categories(['0.0.0.0.0-0'])
test['OsBuildLab'] = test['OsBuildLab'].fillna('0.0.0.0.0-0')
def fe(df):
    df['EngineVersion_2'] = df['EngineVersion'].apply(lambda x:
x.split('.')[2]).astype('category')
    df['EngineVersion_3'] = df['EngineVersion'].apply(lambda x:
x.split('.')[3]).astype('category')
    df['AppVersion_1'] = df['AppVersion'].apply(lambda x:
x.split('.')[1]).astype('category')
    df['AppVersion_2'] = df['AppVersion'].apply(lambda x:
x.split('.')[2]).astype('category')
```

```
    df['AppVersion_3'] = df['AppVersion'].apply(lambda x:
x.split('.')[3]).astype('category')

    df['AvSigVersion_0'] = df['AvSigVersion'].apply(lambda x:
x.split('.')[0]).astype('category')

    df['AvSigVersion_1'] = df['AvSigVersion'].apply(lambda x:
x.split('.')[1]).astype('category')

    df['AvSigVersion_2'] = df['AvSigVersion'].apply(lambda x:
x.split('.')[2]).astype('category')

    df['OsBuildLab_0'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[0]).astype('category')

    df['OsBuildLab_1'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[1]).astype('category')

    df['OsBuildLab_2'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[2]).astype('category')

    df['OsBuildLab_3'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[3]).astype('category')

    # df['OsBuildLab_40'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[-1].split('-')[0]).astype('category')

    # df['OsBuildLab_41'] = df['OsBuildLab'].apply(lambda x:
x.split('.')[-1].split('-')[1]).astype('category')

    df['Census_OSVersion_0'] = df['Census_OSVersion'].apply(lambda x:
x.split('.')[0]).astype('category')

    df['Census_OSVersion_1'] = df['Census_OSVersion'].apply(lambda x:
x.split('.')[1]).astype('category')

    df['Census_OSVersion_2'] = df['Census_OSVersion'].apply(lambda x:
x.split('.')[2]).astype('category')

    df['Census_OSVersion_3'] = df['Census_OSVersion'].apply(lambda x:
x.split('.')[3]).astype('category')

    df['primary_drive_c_ratio'] = df['Census_SystemVolumeTotalCapacity']/
df['Census_PrimaryDiskTotalCapacity']
```

```
    df['non_primary_drive_MB'] = df['Census_PrimaryDiskTotalCapacity'] -
df['Census_SystemVolumeTotalCapacity']


    df['aspect_ratio'] = df['Census_InternalPrimaryDisplayResolutionHorizontal']/
df['Census_InternalPrimaryDisplayResolutionVertical']

    df['monitor_dims'] =
df['Census_InternalPrimaryDisplayResolutionHorizontal'].astype(str) + '*' +
df['Census_InternalPrimaryDisplayResolutionVertical'].astype('str')

    df['monitor_dims'] = df['monitor_dims'].astype('category')

    df['dpi'] = ((df['Census_InternalPrimaryDisplayResolutionHorizontal']**2 +
df['Census_InternalPrimaryDisplayResolutionVertical']**2)**.5)/(df['Census_InternalPr
imaryDiagonalDisplaySizeInInches'])

    df['dpi_square'] = df['dpi'] ** 2

    df['MegaPixels'] = (df['Census_InternalPrimaryDisplayResolutionHorizontal'] *
df['Census_InternalPrimaryDisplayResolutionVertical'])/1e6

    df['Screen_Area'] = (df['aspect_ratio']*
(df['Census_InternalPrimaryDiagonalDisplaySizeInInches']**2))/(df['aspect_ratio']**2
+ 1)

    df['ram_per_processor'] = df['Census_TotalPhysicalRAM']/
df['Census_ProcessorCoreCount']

    df['new_num_0'] = df['Census_InternalPrimaryDiagonalDisplaySizeInInches'] /
df['Census_ProcessorCoreCount']

    df['new_num_1'] = df['Census_ProcessorCoreCount'] *
df['Census_InternalPrimaryDiagonalDisplaySizeInInches']

    df['Census_IsFlightingInternal'] = df['Census_IsFlightingInternal'].fillna(1)

    df['Census_ThresholdOptIn'] = df['Census_ThresholdOptIn'].fillna(1)

    df['Census_IsWIMBootEnabled'] = df['Census_IsWIMBootEnabled'].fillna(1)

    df['Wdft_IsGamer'] = df['Wdft_IsGamer'].fillna(0)

    return df
```

```
train = fe(train)
test = fe(test)
more_cat_cols = []
add_cat_feats = [
 'Census_OSBuildRevision',
 'OsBuildLab',
 'SmartScreen',
'AVProductsInstalled']
for col1 in add_cat_feats:
    for col2 in add_cat_feats:
        if col1 != col2:
            train[col1 + '__' + col2] = train[col1].astype(str) + train[col2].astype(str)
            train[col1 + '__' + col2] = train[col1 + '__' + col2].astype('category')
            test[col1 + '__' + col2] = test[col1].astype(str) + test[col2].astype(str)
            test[col1 + '__' + col2] = test[col1 + '__' + col2].astype('category')
            more_cat_cols.append(col1 + '__' + col2)
cat_cols = cat_cols + more_cat_cols
to_encode = []
for col in cat_cols:
    if train[col].nunique() > 1000:
        print(col, train[col].nunique())
        to_encode.append(col)
train = reduce_mem_usage(train)
test = reduce_mem_usage(test)
gc.collect()
```

```
for col in tqdm_notebook(to_encode):
    freq_enc_dict = frequency_encoding(col)
    train[col] = train[col].map(lambda x: freq_enc_dict.get(x, np.nan))
    test[col] = test[col].map(lambda x: freq_enc_dict.get(x, np.nan))
    cat_cols.remove(col)
%%time
indexer = {}
for col in cat_cols:
    # print(col)
    _, indexer[col] = pd.factorize(train[col].astype(str), sort=True)
for col in tqdm_notebook(cat_cols):
    # print(col)
    train[col] = indexer[col].get_indexer(train[col].astype(str))
    test[col] = indexer[col].get_indexer(test[col].astype(str))
    train = reduce_mem_usage(train, verbose=False)
    test = reduce_mem_usage(test, verbose=False)
n_fold = 5
folds = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=15)
# folds = TimeSeriesSplit(n_splits=5)
from numba import jit


def fast_auc(y_true, y_prob):
    y_true = np.asarray(y_true)
    y_true = y_true[np.argsort(y_prob)]
    nfalse = 0
```

```
    auc = 0
    n = len(y_true)
    for i in range(n):
        y_i = y_true[i]
        nfalse += (1 - y_i)
        auc += y_i * nfalse
    auc /= (nfalse * (n - nfalse))
    return auc
def train_model(X=train, X_test=test, y=y, params=None, folds=folds,
model_type='lgb', plot_feature_importance=False, averaging='usual', make_oof=False):
    result_dict = {}
    if make_oof:
        oof = np.zeros(len(X))
    prediction = np.zeros(len(X_test))
    scores = []
    feature_importance = pd.DataFrame()
    for fold_n, (train_index, valid_index) in enumerate(folds.split(X, y)):
        gc.collect()
        print('Fold', fold_n + 1, 'started at', time.ctime())
        X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
        y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]



        if model_type == 'lgb':
            train_data = lgb.Dataset(X_train, label=y_train, categorical_feature = cat_cols)
            valid_data = lgb.Dataset(X_valid, label=y_valid, categorical_feature = cat_cols)
```

```
            model = lgb.train(params,
                              train_data,
                              num_boost_round=2000,
                              valid_sets = [train_data, valid_data],
                              verbose_eval=500,
                              early_stopping_rounds = 200,
                              feval=eval_auc)
            del train_data, valid_data
            y_pred_valid = model.predict(X_valid, num_iteration=model.best_iteration)
            del X_valid
            gc.collect()
            # print('predicting on test')
            # y_pred = model.predict(X_test, num_iteration=model.best_iteration)
            y_pred = predict_chunk(model, X_test)
            # print('predicted')
    if model_type == 'lgb':
        if plot_feature_importance:
            feature_importance["importance"] /= n_fold
            cols = feature_importance[["feature",
"importance"]].groupby("feature").mean().sort_values(
                by="importance", ascending=False)[:50].index
            best_features = feature_importance.loc[feature_importance.feature.isin(cols)]
            logging.info('Top features')
            for f in best_features.sort_values(by="importance",
ascending=False)['feature'].values:
                logging.info(f)
```

```
            plt.figure(figsize=(16, 12));
            sns.barplot(x="importance", y="feature",
data=best_features.sort_values(by="importance", ascending=False));
            plt.title('LGB Features (avg over folds)');


            result_dict['feature_importance'] = feature_importance
    result_dict['prediction'] = prediction
    if make_oof:
        result_dict['oof'] = oof
    return result_dict

params = {'num_leaves': 256,
         'min_data_in_leaf': 42,
         'objective': 'binary',
         'max_depth': 5,
         'learning_rate': 0.05,
         "boosting": "gbdt",
         "feature_fraction": 0.8,
         "bagging_freq": 5,
         "bagging_fraction": 0.8,
         "bagging_seed": 11,
         "lambda_l1": 0.15,
         "lambda_l2": 0.15,
         "random_state": 42,
         "verbosity": -1}
```

```
result_dict1 = train_model(X=train1, X_test=test, y=y1, params=params,
model_type='lgb', plot_feature_importance=True, averaging='rank')

submission = pd.read_csv('../input/sample_submission.csv')
```

Додаток Б

Графічна частина

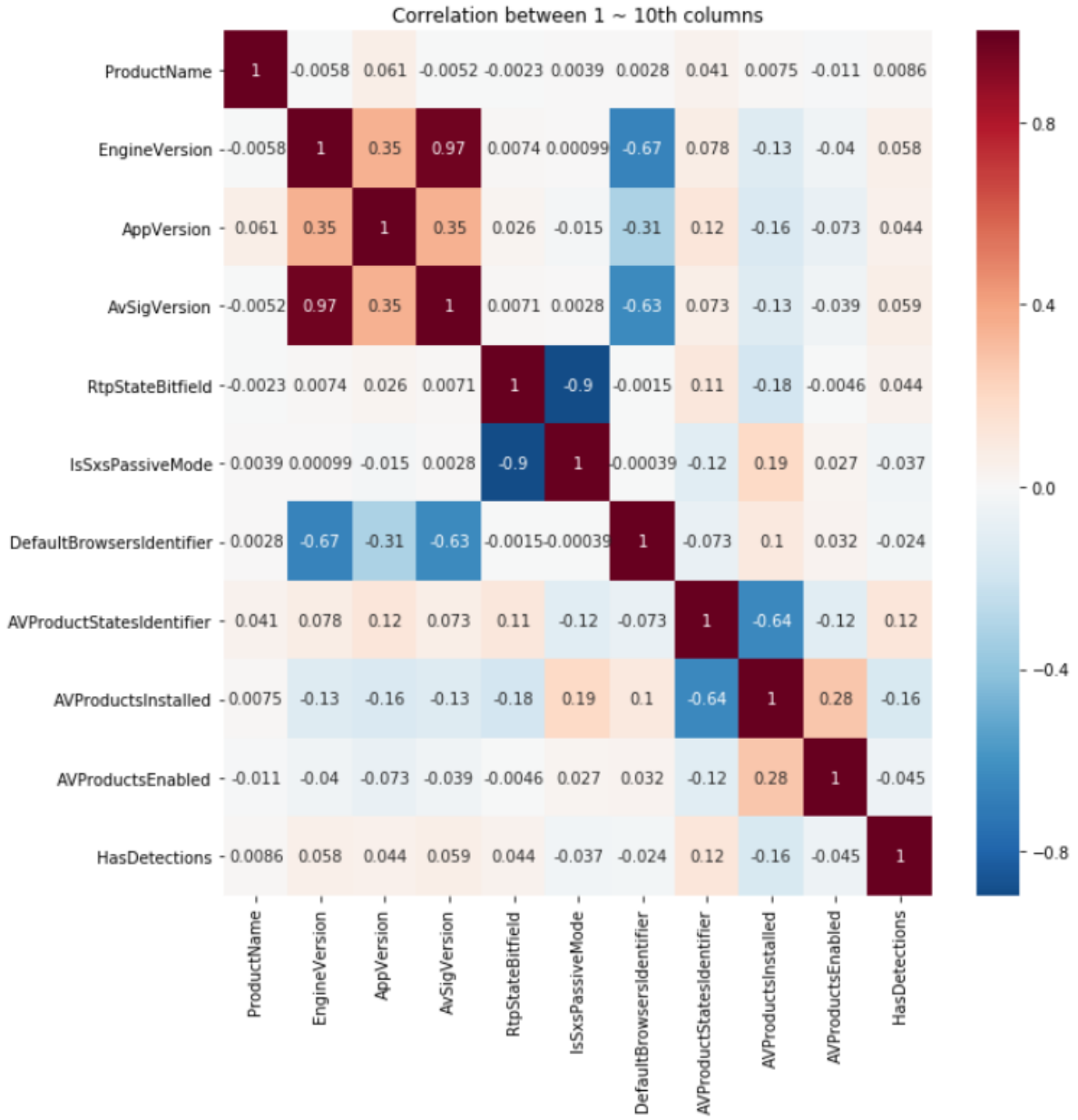


Рисунок Б1. – Кореляція між 1-10 полями

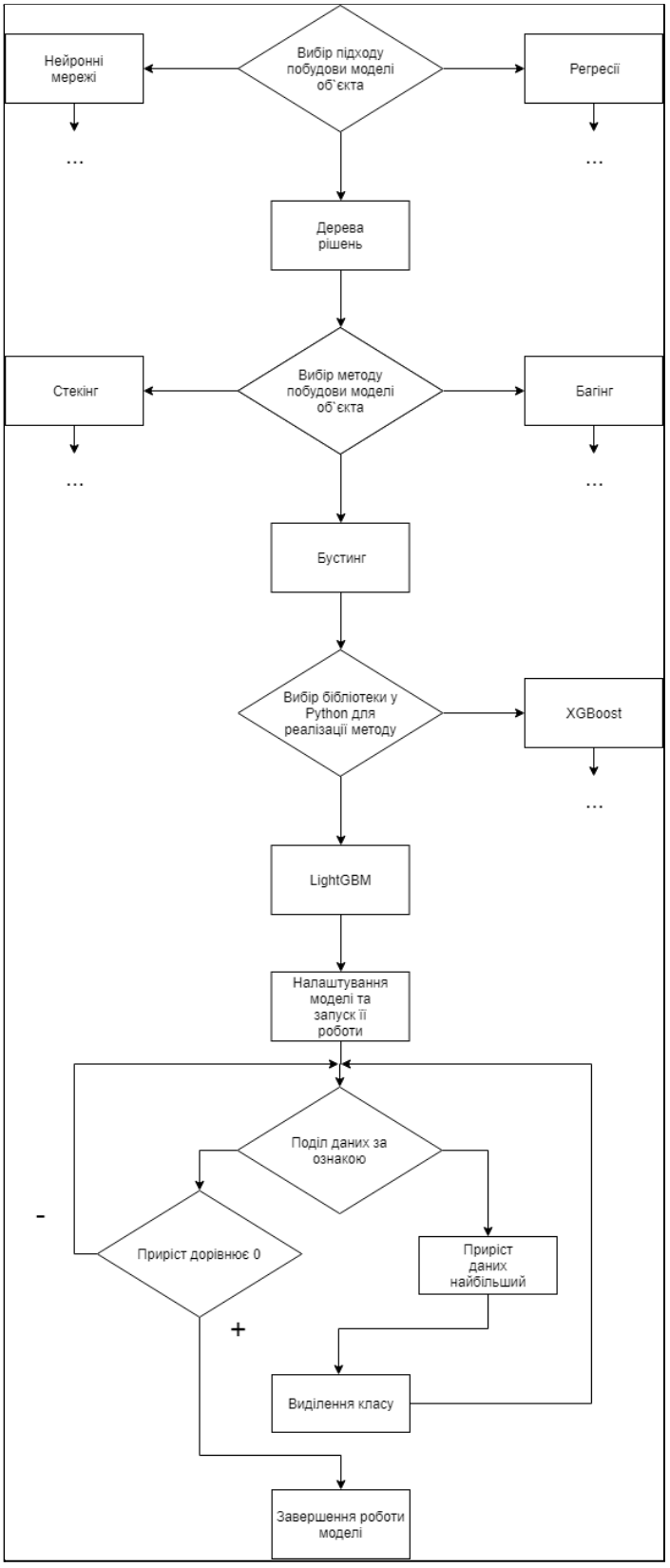


Рисунок Б.2 – Алгоритм прогнозування даних

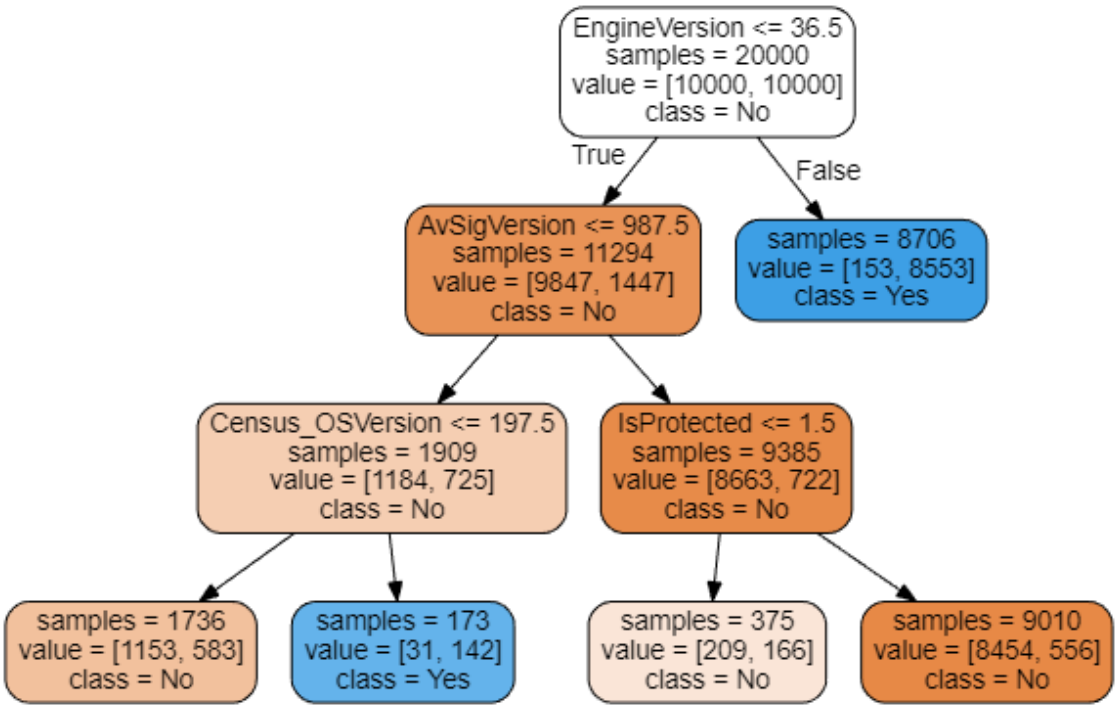


Рисунок Б.3 – Дерево рішень

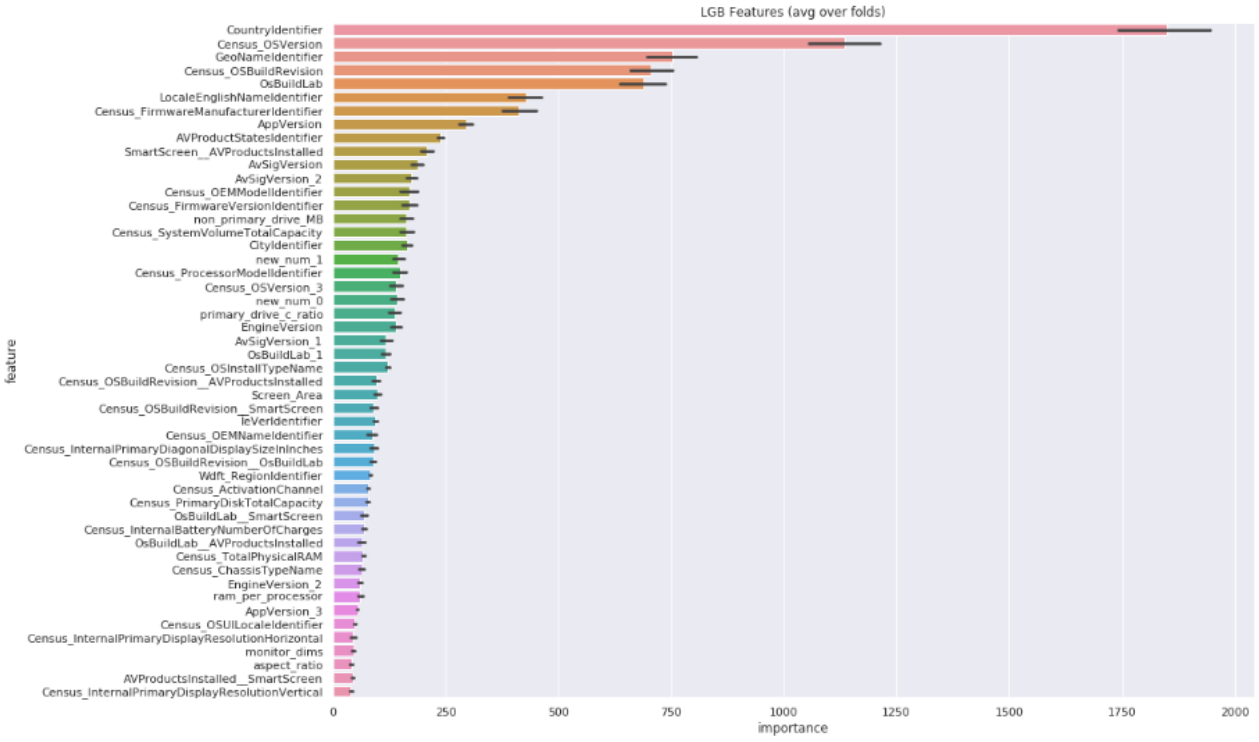


Рисунок Б.4 – Діаграма важливості ознак

Додаток В

Код програми

```
import lightgbm as lgb
import platform
import subprocess
import json

#load from model
bst = lgb.Booster(model_file='lgb_model.txt')
def get_systeminfo():
    # Command to run.
    command = ['systeminfo']
    os = platform.system()
    if os == "Linux":
        cmd = ['sudo lshw']
    elif os == "MacOS":
        cmd = ['system_profiler']

    # Run the commands and get the stdout.
    with subprocess.Popen(command, universal_newlines=True,
stdout=subprocess.PIPE) as p:
        stdout, _ = p.communicate()
    return stdout
```

windows_keys = ['Processor(s)', 'BIOS Version', 'Network Card(s)', 'Region(s)', 'version', 'OS Name', 'OS Version', 'System type', 'System Locale', 'System Model', 'OS Configuration', 'Time Zone']

linux_keys = ['OS Version', 'System type', 'Region', 'version', 'OS Name', , 'System Locale', 'System Model', 'OS Configuration', 'Processor', 'BIOSv', 'Network', 'Time Zone']

macos_keys = ['Networks', 'Region', 'OS Version', 'System type', 'System Locale', 'System Model', 'Configuration', 'Timezone', 'Processor(s)', 'BIOS']

```
def dic_from_systeminfo(stdout):
    # Dic to store all info.
    dic = {}
    # Previous key name saved to reuse for dic values.
    prevkey = ''
    # Loop through each line of stdout and split by colon.
    for line in stdout.splitlines():
        key, sep, value = line.partition(':')
        if sep == ':':
            value = value.strip()
            os = platform.system()
            if os == "Linux":
                keys = linux_keys
            elif os == "MacOS":
                keys = macos_keys
            elif os == "Windows":
                keys = windows_keys
```

```
        if not key.startswith(' '):
            # Assign these keys with values of type dic.
            if key in keys:
                value = dict()
            # Add to dic and save key name.
            dic[key] = value
            prevkey = key
        else:
            # Remove [] characters and then add key and value to the value type dic.
            key = key.strip().replace('[', '').replace(']', '')
            if prevkey and isinstance(dic[prevkey], dict):
                dic[prevkey][key] = value
            else:
                print('Warning:', 'dic[' + prevkey + '] is not a dict value.')
    return dic
```