МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система аутентифікації пристроїв інтернету речей»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Москаленко В.В.**

Студента групи ІК.м-91 **Лобін Є.О.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕЛІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«Інформаційно-комунікаційні технології»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Лобіну Євгенію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система аутентифікації пристроїв інтернету речей*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми. Постановка задачі дослідження 2) Огляд існуючих рішень 3) Вибір методу рішення 4) Інформаційне та програмне забезпечення системи
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Огляд існуючих рішень*		
3.	*Вибір методу рішення*		
4.	*Інформаційне та програмне забезпечення системи*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 52 стор., 20 рис., 2 табл., 3 додатка, 18 джерел.

Об’єкт дослідження — система аутентифікації інтернету речей.

Мета роботи — аналіз методів машинного навчання, для розпізнавання приладів.

Методи дослідження — метод розпізнавання приладів на основі інтернет трафіку.

Результати — проаналізовано способи аутентифікації інтернету речей. Запропоновано алгоритм аутентифікації, що оснований на машинному навчанні, який буде використовувати перехоплений мережевий трафік як вхідні данні. Було протестована та обрано найкращий алгоритм машинного навчання, для виконання поставленої задачі. В результат, кращий результат точності показала модель Gradient Boosting.

МАШИННЕ НАВЧАННЯ, МЕТОДИ АУТЕНТІФІКАЦІЇ,
ІНТЕРНЕТ РЕЧЕЙ, АЛГОРИТМИ МАШИИНОГО НАВЧАННЯ,
МЕТОДИ ОПТИМІЗАЦІЇ ГІПЕРПАРАМЕТРІВ.

ЗМІСТ

ВСТУП

Інтернет речей (IoT) - це безліч взаємопов’язаних пристроїв наступного покоління, що включає датчики, сканери, виконавчі механізми тощо, і які можуть надавати персоналізовані послуги, такі як охорона здоров’я, безпека та спостереження. Простіше кажучи, це концепція підключення будь-якого пристрою з перемикачем увімкнення та вимкнення до Інтернету (та / або один до одного). Сюди входить все - від мобільних телефонів, кавоварок, пральних машин, навушників, ламп, пристроїв, що носяться, і майже всього іншого, що ви можете подумати. IoT покращує якість нашого щоденного життя за допомогою всеосяжних обчислень та спілкування.

Щодня до програм IoT підключається незліченна кількість пристроїв. Незважаючи на те, що якість нашого життя покращує IoT, але існує ризик того, що зловмисники можуть отримати контроль над технікою інтернета речей і спричинити серйозні проблеми її користувачам. Серед проблем, які IoT ставить перед організаціями, це питання безпеки та управління, пов'язані з розповсюдженням таких пристроїв та постійним збільшенням кількості активів організації, що підтримують IoT. У майбутньому організації можуть точно не знати, які пристрої IoT підключені до їх мережі.

Пристрої IoT часто зламують віддалено, підключаючи хакера, який намагається увійти в пристрій за допомогою підключення до Інтернету. Якщо IoT-пристрою дозволено взаємодіяти лише з автентифікованим сервером, будь-які зовнішні спроби спілкування будуть ігноруватися.

Згідно з повідомленням про загрози Symantec за 2018 рік[], кількість атак IoT зросла на 600 відсотків між 2016 і 2017 роками, з 6 000 до 50 000 атак відповідно.

Тому, коли пристрої IoT реалізуються в корпоративних мережах, безпеці потрібно приділяти набагато більше уваги. Для вирішення цієї проблеми для стандартизації безпечного зв'язку між машинами необхідно використовувати потужні, але ефективні криптографічні рішення.

Однак важко вибрати правильну модель автентифікації IoT для роботи. Перш ніж вирішити, яка модель архітектури в кінцевому підсумку є найкращою автентифікацією IoT, слід врахувати кілька факторів, таких як енергетичні ресурси, потужність обладнання, фінансовий бюджет, експертиза безпеки, вимоги до безпеки та підключення.

У цій роботі вирішується проблема ідентифікації пристроїв IoT у мережі, шляхом аналізу та класифікації даних трафіку від таких пристроїв. Навіть якщо префікси MAC-адреса можуть бути використані для ідентифікації виробника певного пристрою, не існує стандарту для ідентифікації брендів або типів пристроїв. Пропонується використовувати такі особливості мережі для ідентифікації пристроїв IoT.

Данний підхід є загальним і досить гнучким, щоб бути застосованим у швидкозмінюваному ландшафті Інтернету речей і в той же час достатньо націленим на ефективну ідентифікацію пристроїв та аналіз мережевого трафіку.

Основним завданням цій роботи є дослідженн нового методу класифікації пристроїв, підключених до мережі організації, на основі виключно аналізу мережевого трафіку. Більш конкретно, основа увага буде зосереджена на таких питаннях:

а) Чи можливо точно розрізнити пристрої IoT та пристрої, що не належать до IoT?

б) Для конкретного пристрою IoT (наприклад, смарт-телевізор, IP-камера) чи можливо точно моделювати поведінку мережі та виявляти такий пристрій у мережевому трафіку

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Подібно до індустрії програмного забезпечення, де безпека при дизайні заохочується при розробці програмного забезпечення, IoT повинен включати якийсь механізм перевірки, щоб гарантувати, що будь-які створені дані надсилаються та отримуються лише залученими сторонами.

Незалежно від того, це пристрій на основі датчика або який використовується для виконання певної функції, усі пристрої можуть бути відкритими для хакерства, якщо не вжити профілактичних заходів. Є кілька прикладів цього, від вимкнення монітора спостереження за турецьким трубопроводом[2] до нападу на такі медичні пристрої, як інсулінові помпи[3] та апарати для МРТ[4]. Ці атаки потенційно загрожують життю і вказують на те, що деякі хакери не мають ніяких сумнівів, коли йдеться про вибір цілей або отримання прав на хвастощі перед іншими кіберзлочинцями.

Очевидно, що підключені пристрої містять ризик, який може загрожувати життю або поставити під загрозу добробут вашої компанії чи сім'ї. Щоб зрозуміти ризики, нам спочатку потрібно зрозуміти, як пристрої взаємодіють.

1.1 Принцип роботи системи IoT

IoT - це набагато більше, ніж підключені пристрої, і в ідеалі вимагає організованої та виділеної IoT-інфраструктури, яка, як правило, розглядається як така, що має чотири різні етапи, що відображають шлях, що проходить від пристроїв IoT до остаточного аналізу. Обробка даних може відбуватися на кожному з цих чотирьох етапів.

1. Датчик або привід - Наприклад, датчик може збирати дані для контролю температури води, тоді як привід буде виконувати фізичну функцію, таку як закриття або відкриття клапана при досягненні заданої температури.

2. Інтернет-шлюзи - дані аналогового датчика збираються та перетворюються на цифрові, а потім передаються через обраний вами протокол, будь то Wi-Fi, дротова локальна мережа або Інтернет. Це дозволяє

надсилати всі дані на обробку, оскільки аналіз у режимі реального часу вимагає великих обсягів даних і може сповільнити вашу мережу.

3. Edge IT - проміжний етап, який виконує додатковий аналіз перед відправкою даних до центру обробки даних. Знову ж таки, це зменшення трафіку до центру обробки даних та гарантування того, що пропускна здатність мережі не перевищується в центрі обробки даних. Наприклад, вам не знадобляться всі дані з усіх пристроїв, а лише дані, які відповідають визначеним критеріям для подальших дій.

4. Центр обробки даних або хмара - можливий детальний аналіз решти даних, а створені звіти надсилаються в локальну мережу. Коли обробка та аналіз даних відбувається за межами робочої зони, IT-командам не потрібно турбуватися про відсутність пропускної здатності мережі на місці.

Способи реалізації кожного з цих етапів залежатимуть від кількості датчиків і пристроїв IoT, обсягу генерованих даних та способу обробки цих даних. Ефективна екосистема IoT повинна враховувати безпеку та аутентифікацію - це один із способів досягнення цієї мети, незалежно від того, задіяний вона в Індустріальному Інтернеті або просто використовуючи переваги пристроїв IoT, які доповнюють операційні процеси.

Аутентифікація IoT - це модель для формування довіри до ідентичності машин та пристроїв IoT для захисту даних та контролю доступу, коли інформація подорожує через незахищену мережу, таку як Інтернет.

Аутентифікація також допомагає запобігти спробам зловмисників видавати себе за пристрої Інтернету речей в надії отримати достуg до даних на серверах, таких як записані розмови, зображення та інша потенційно конфіденційна інформація.

Існує декілька методів, за допомогою яких ми можемо досягти надійної аутентифікації для захисту зв'язку між пристроями IoT:

1. Одностороння аутентифікація: у випадку, коли дві сторони бажають спілкуватися одна з одною, лише одна сторона аутентифікується перед іншою, тоді як інша сторона не буде аутентифікована.

2. Двостороння аутентифікація: також називається взаємною аутентифікацією, коли обидві сутності аутентифікують одна одну.

3. Тристороння аутентифікація: це місце, де центральний орган аутентифікує дві сторони та допомагає їм аутентифікувати одна одну.

4. Розподілений: використання розподіленого методу прямої аутентифікації між учасниками зв'язку.

5. Централізоване: використання централізованого сервера або довіреної третьої сторони для розповсюдження та управління використовуваними сертифікатами аутентифікації.

Авторизація IoT - це інструмент, що використовується для перевірки ідентичності кожної кінцевої точки в системі IoT. Процес сертифікації налаштовується під час реєстрації та інформує постачальника послуг про метод, який слід використовувати при перевірці ідентичності системи під час реєстрації.

1.2 Сертифікати X.509

Протокол X.509 (IETF RFC 5280) забезпечує найбільш безпечний тип аутентифікації цифрових ідентифікаційних даних і базується на моделі ланцюга сертифікатів довіри. Використання сертифікатів X.509 як механізму сертифікації є прекрасним способом масштабування виробництва та спрощення постачання обладнання.

Інфраструктура відкритих ключів (PKI) складається з деревоподібної структури серверів та пристроїв, що підтримують список надійних кореневих сертифікатів. Кожен сертифікат містить відкритий ключ пристрою та підписаний приватним ключем CA. Унікальний «відбиток пальця» забезпечує унікальну ідентичність, яку можна перевірити, запустивши крипто-алгоритм, такий як RSA.

Цифрові сертифікати, як правило, розташовані в ланцюжку сертифікатів на рис. 1.1[6] в якому кожен сертифікат підписаний приватним ключем іншого довіреного сертифіката, і ланцюжок повинен повернутися до глобально довіреного кореневого сертифіката. Ця домовленість встановлює делегований ланцюжок довіри від довіреного центру кореневих сертифікатів (CA) до остаточного сертифіката “листочка”, встановленого на пристрої через кожну проміжну CA.

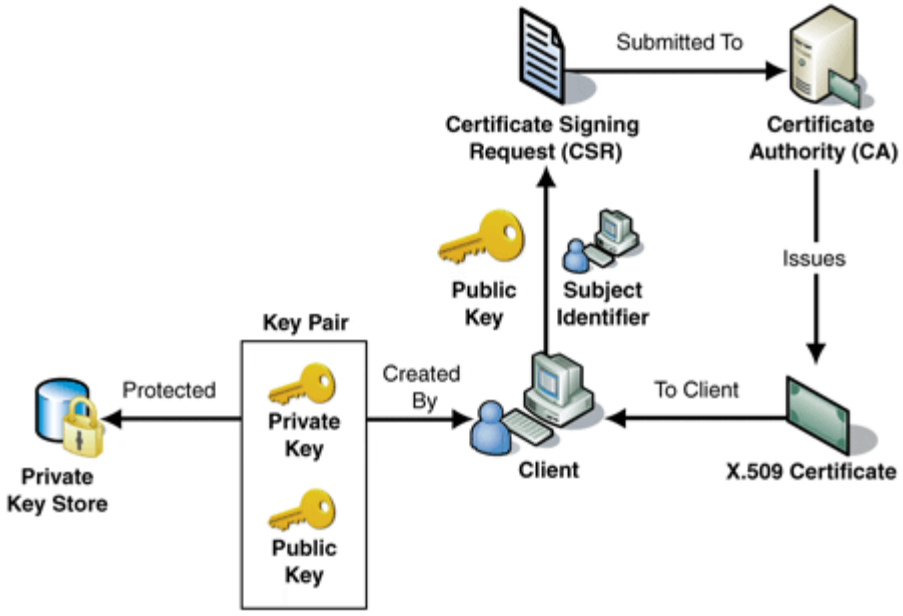

Рисунок 1.1 – Принцип роботи сертифікатів X.509

Це вимагає великого контролю з боку керівництва, але існує безліч варіантів постачальників.

Однак управління життєвим циклом сертифіката X.509 може бути складним завданням через логістичні складності та має свою ціну, що додає до загальної вартості рішення. З цієї причини багато клієнтів покладаються на зовнішніх постачальників для отримання сертифікатів та автоматизації життєвого циклу.

1.3 Модуль апаратного захисту (HSM)

Модуль апаратної безпеки (HSM) використовується для безпечного, секретного зберігання пристроїв на основі апаратних засобів і є найбезпечнішою формою секретного зберігання. І сертифікат X.509, і маркер SAS можуть зберігатися в HSM як на рис.1.2[9]. HSM можуть

використовуватися з двома механізмами атестації, що підтримуються службою надання послуг.

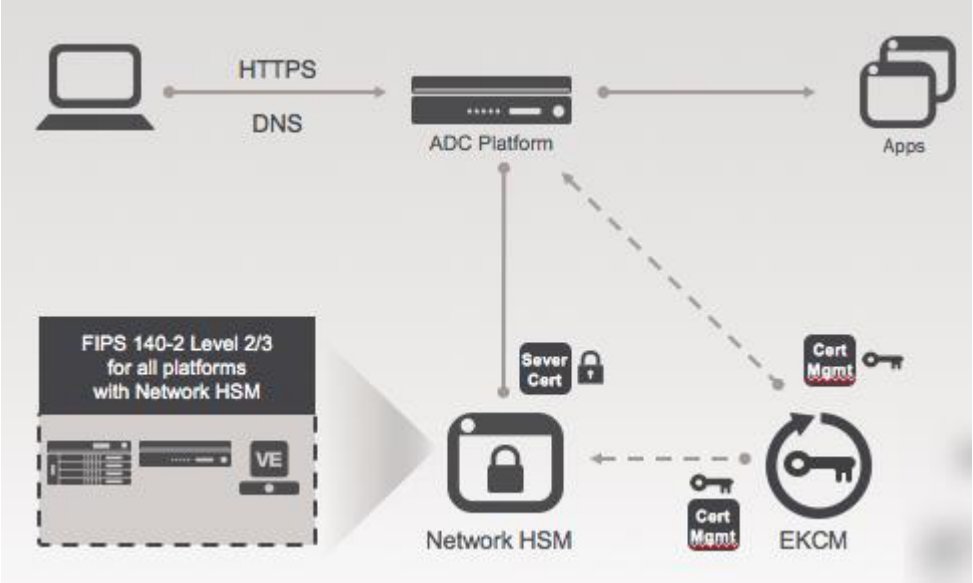

Рисунок 1.2 – Принцип роботи сертифікатів HSM

Крім того, секрети пристрою також можуть зберігатися в програмному забезпеченні (пам’яті), але це менш безпечна форма зберігання порівняно з HSM.

1.4 Модуль довіреної платформи (TPM)

Важливо перевірити ідентифікацію пристрою, який зв’язується із шлюзом обміну повідомленнями в розгортаннях автентифікації IoT. Звичайним методом є створення пар ключів для пристроїв, які потім використовуються для автентифікації та шифрування трафіку. Однак дискові пари ключів схильні до фальсифікацій.

TPM бувають різних форм, зокрема:

- Стримані апаратні пристрої
- Вбудоване апаратне обладнання
- Впровадження прошивки
- Впровадження програмного забезпечення

Хоча типовий TPM має кілька криптографічних можливостей, три ключові функції мають значення для автентифікації IoT:

- Безпечне завантаження
- Встановлення кореня довіри (RoT)
- Ідентифікація пристрою

Виробники пристроїв не завжди можуть повною мірою довіряти всім суб'єктам у своєму ланцюзі поставок. Проте вони не можуть відмовитись від економічних вигод від використання недорогих постачальників та обладнання. TPM можна використовувати в різних точках ланцюга постачання, щоб перевірити, чи не було неправильно змінено пристрій.

TPM має можливість надійно зберігати ключі в захищеному від несанкціонованого обладнання середовищі. Ключі генеруються в самому TPM і тому захищені від отримання зовнішніми програмами. Навіть не використовуючи можливості надійного апаратного забезпечення та безпечного завантаження, TPM настільки ж цінний, як сховище апаратних ключів. Приватні ключі захищені апаратним забезпеченням і забезпечують набагато кращий захист, ніж програмний ключ.

1.5 Симетричні ключі

Сертифікація симетричного ключа - це простий підхід до автентифікації пристрою за допомогою екземпляра Служби надання послуг. Цей метод сертифікації є досвідом «Hello World» для розробників. Атестація пристрою з використанням TPM або сертифіката X.509 є більш безпечною і повинна використовуватися для більш жорстких вимог безпеки.

Реєстрація симетричних ключів також забезпечує чудовий спосіб для застарілих пристроїв з обмеженими функціями безпеки завантажуватися в хмару через Azure IoT.

Симетрична атестація ключів за допомогою Служби надання пристроїв здійснюється за допомогою тих самих маркерів безпеки, що підтримуються

концентраторами IoT для ідентифікації пристроїв. Ці маркери безпеки є маркерами SAS (Shared Access Signature).

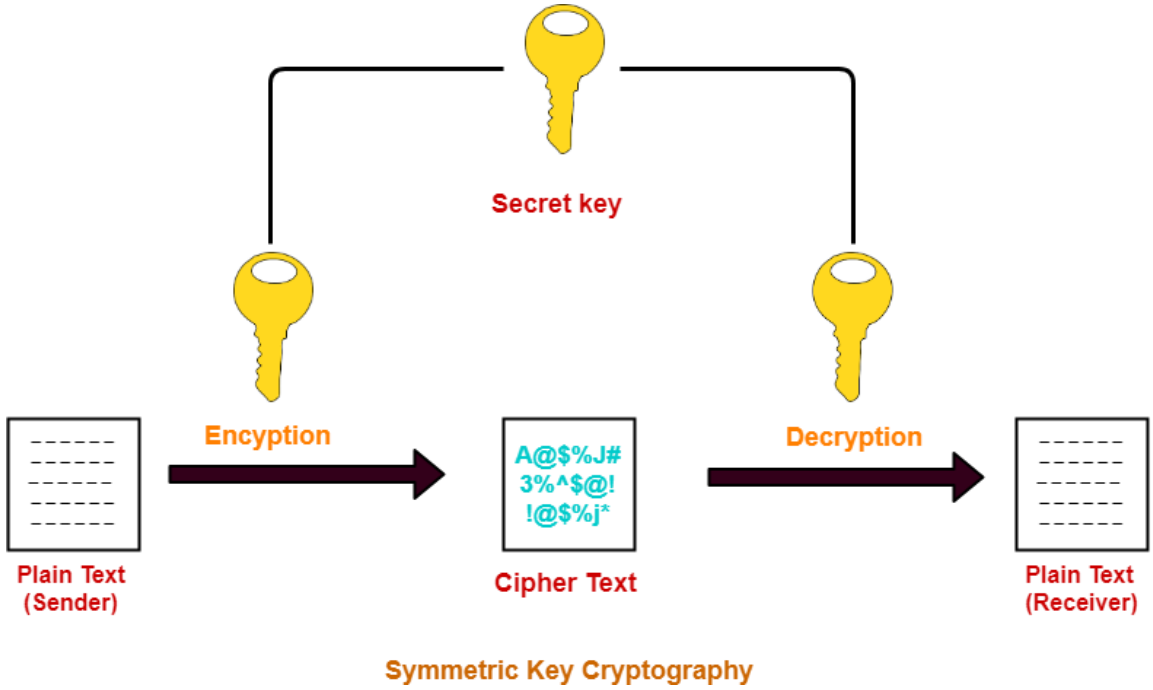

Рисунок 1.3 – Криптографія симетричних ключів

Токени SAS мають хешований підпис, створений за допомогою симетричного ключа. Підпис відтворюється Службою надання пристроїв, щоб перевірити, чи справжній маркер безпеки, представлений під час сертифікації.

Коли пристрій засвідчує індивідуальну реєстрацію, пристрій використовує симетричний ключ, визначений в індивідуальному записі реєстрації, щоб створити хешований підпис для маркера SAS.

Спільні симетричні ключі можуть бути менш захищеними, ніж сертифікати X.509 або TPM, оскільки один і той же ключ використовується пристроєм та хмарою, а це означає, що ключ потрібно захищати у двох місцях. Дизайнери, які використовують симетричні клавіші, іноді жорстко кодують чисті (незашифровані) клавіші на пристрої, роблячи ключі вразливими, що не є рекомендовано практикою.

Правильна реалізація автентифікації IoT має багато корисних наслідків для безпеки IoT. Однак вибір правильного методу може бути складним завданням, а неправильний вибір може в десять разів збільшити ризики.

Деякі ризики можна пом'якшити, надійно зберігаючи симетричний ключ на пристрої та дотримуючись найкращих практик щодо зберігання ключів. Це не неможливо, але коли симетричні ключі використовуються виключно, вони можуть бути менш безпечними, ніж реалізації HSM, TPM та X.509.

У випадку сертифікатів, HSM, TPM та додатків X.509 основна проблема полягає у доведенні ключа, не розкриваючи приватну частину ключа.

1.6 Постановка задачі

Необхідно реалізувати механізм розпізнавання пристроів інтернету речей, представленний у вігляді масиву данних, який створенний за допомгою перехоплення інтернет трафіку у вигляді .pcap файлів, та його аналізу.

Обрати мови програмування, та виділити декілька ансаблевих методів класифікаціїї данних, які будуть використовуватися у машинному навчанні. Оптимізувати гіперпараметри моделей навчання, для отримання кращого кінцевого результату.

Після чого провести аналіз результатів, та обрати метод, який будет краще за всіх підходити для розпізнавання отриманних данних.

2 ВИБІР МЕТОДУ РІШЕННЯ

Так як основною темою роботи є аналіз забранних данних, то в першу чергу необхідно обрати мову програмування, яка буде відповідати за цей процесс.

Наступним шагом

2.1 Вибір мови програмування

Data Science вже давно є великою справою. У сучасному технологічному світі, що швидко розростається, коли люди, як правило, генерують багато даних, надзвичайно важливо, щоб ми знали, як аналізувати, обробляти та використовувати ці дані для подальших знань про бізнес.

В данному розділи буде проаналізовано декілька мов програмування, які використовуются в Data Science, та є найбільш популярними в 2020 році, це будуть Python, R, Scala та SAS.

Об'єктно-орієнтована природа Python полегшує вченим даних виконувати завдання з кращою стабільністю, модульністю та читабельністю коду. Python багата на спеціалізовані бібліотеки глибокого навчання та інших машинних навчальних програм та популярних інструментів, таких як scikit-learn, Keras та TensorFlow. Безсумнівно, Python дозволяє вченим-розробникам даних розробляти складні моделі даних, які можна підключити безпосередньо до виробничої системи.

За результатами опитування розробників Python[12], 84% респондентів використовували Python як свою основну мову, тоді як для 16% це була їхня друга мова.

Для збору даних Python підтримує таблиці CSV, JSON, SQL та обробку веб-сторінок.

Бібліотека аналізу даних для Python, Pandas пропонує найкраще, що ви можете отримати для дослідження даних[18]. Організовані у фрейми даних, Pandas можуть фільтрувати, сортувати та відображати дані з усією легкістю, яку ви можете собі уявити.

Для моделювання даних

1. NumPy - аналіз чисельного моделювання
2. SciPy - наукові обчислення та обчислення
3. scikit-learn - отримати доступ до численних потужних алгоритмів машинного навчання. Він також пропонує інтуїтивно зрозумілий інтерфейс, який дозволяє вченим з обробки даних використати всю потужність машинного навчання без його численних складностей

R - це інструмент з відкритим кодом, який дозволяє вченим даних працювати з багатьма операційними системами на різних платформах. Статистика є основною силою цієї технології. R - це не просто мова, а ціла екосистема сама по собі для проведення статистичних розрахунків. Це полегшує виконання операцій з обробки даних, математичного моделювання, візуалізації даних із вбудованими функціями.

R підтримує формати файлів Excel, CSV, текстові файли, файли у форматі Minitab або SPSS, веб-обмін за допомогою Rvest.

R був побудований для статистичного та чисельного аналізу великих наборів даних, а отже, існує безліч операцій, які можна виконати для дослідження даних - сортування даних, транспонування таблиць, створення графіків, генерація таблиць частот, дані вибірки, розподіл ймовірностей, об'єднувати дані, перетворювати змінні та багато іншого.

R - це надійне середовище, придатне для наукової візуалізації з багатьма пакетами, які спеціалізуються на графічному відображенні результатів для візуалізації даних.

Scala - це поєднання об’єктно-орієнтованого та функціонального програмування в одній короткій мові високого рівня. Ця мова спочатку була розроблена для віртуальної машини Java (JVM), і однією з сильних сторін Scala є те, що вона дозволяє дуже легко взаємодіяти з кодом Java.

Одну з основних причин вивчення Scala для Data Science можна віднести до Apache Spark. Scala, що використовується спільно з Apache Spark для роботи

з великими обсягами даних (Big Data), робить її неоціненною для вчених з питань даних.

Багато високопродуктивних фреймворків даних, побудованих на вершині Hadoop, зазвичай написані та використовують Scala або Java. Причиною використання Scala у цих середовищах є його швидка підтримка одночасності. Оскільки Scala працює на JVM, у поєднанні з Hadoop це майже не заважає.

Як і R, SAS - це інструмент, розроблений для вдосконаленого аналізу даних та складних статистичних операцій. Це власний інструмент із закритим кодом, який пропонує широкий спектр статистичних можливостей для виконання складного моделювання. SAS в основному використовується великими організаціями та професіоналами завдяки високій надійності.

Зауважте, SAS не є інструментом, який найкраще підходить для початківців та незалежних любителів науки про дані, оскільки SAS розроблений спеціально для задоволення потреб бізнесу.

SAS добре виконує статистичне моделювання за допомогою SAS Base - основної мови програмування, що керує середовищем SAS.

Проаналізувавши цей список, так переглянувши доступні середовища розробки основною технологією для аналізу було обрано Python. Середовищем розробки було обрано PyCharm IDEA, яке є комерційним інтегрованим середовищем програмування, але також має безкоштовну Community версію.

2.2 Вибір методів аналізу данних

Ансамблеве навчання - це потужний алгоритм машинного навчання[1], який використовується фахівцями з питань даних у різних галузях. Принадність ансамблевих методів навчання полягає в тому, що вони поєднують у собі передбачення багатьох моделей машинного навчання. Так як основною мовою програмування обрано Python, то методи аналізу данних будуть розглянуті тільки ті, що мають реалізацію для Python.

Приклади ансамблевих методів навчання:

- Bagging

– Boosting
– Stacking
– Blending, etc.

Ці методи навчання ансамблю включають такі популярні алгоритми машинного навчання, як XGBoost, Gradient Boosting, Randomized decision trees та інші.

Спершу розглянемо деіякі з них:

Bagging – основна ідєя полягає у поєднанні результатів багатьох моделей (наприклад, усіх дерев рішень) для отримання узагальненого результату, основною технікою якого є Bootstrapping.

Bootstrapping - це техніка вибірки, при якій створюються підмножини спостережень із вихідного набору даних із заміною. Розмір підмножин такий же, як розмір оригінального набору[7].

Техніка Bagging (або Bootstrap Aggregating) використовує ці підмножини (пакети), щоб отримати чітке уявлення про розподіл (повний комплект). Розмір підмножин, створених для bagging, може бути меншим за початковий набір.

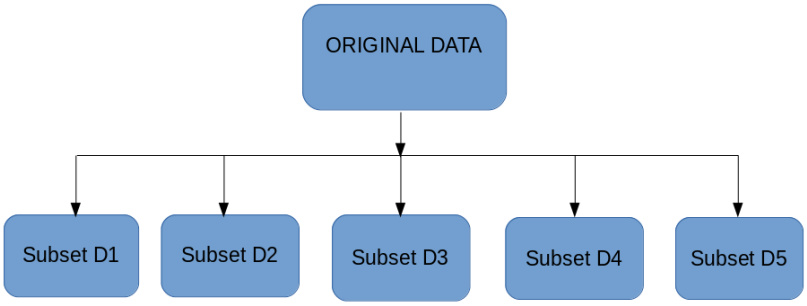

Рисунок 2.1 – Принцип роботи техніки Bagging(ч.1)

1. Кілька підмножин створюються з вихідного набору даних, вибираючи спостереження із заміною, як на рис 2.1[13].
2. На кожну з цих підмножин створюється базова модель (слабка модель).
3. Моделі працюють паралельно і не залежать одна від одної.
4. Остаточні прогнози визначаються поєднанням прогнозів з усіх моделей, як показано на рис. 2.2[13].

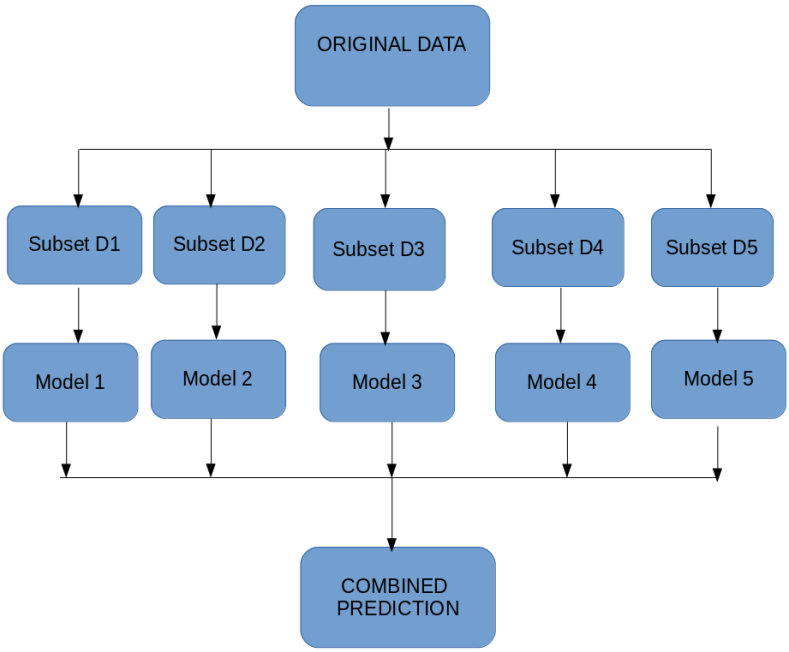

Рисунок 2.2 – Принцип роботи техніки Bagging(ч.2)

Boosting - це послідовний процес, коли кожна наступна модель намагається виправити помилки попередньої моделі. Наступні моделі залежать від попередньої моделі. У наступних кроках описаний алгоритм роботи цієї моделі[11].

1. Підмножина створюється з вихідного набору даних.
2. Спочатку всі точки даних отримують рівні ваги.
3. На цій підмножині створюється базова модель.
4. Ця модель використовується для прогнозування всього набору даних
5. Помилки обчислюються з використанням фактичних значень та передбачених значень.
6. Спостереженням, які неправильно передбачені, надається більша вага. (На рис. 2.3 трьом неправильно класифікованим пунктам "синій плюс" буде надано вищі ваги)

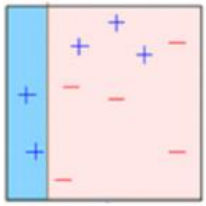

Рисунок 2.3 – Принцип роботи техніки Boosting (ч.1)

7. Створюється інша модель та прогнозуються дані щодо набору даних. (Модель на рис. 2.4 намагається виправити помилки попередньої моделі)

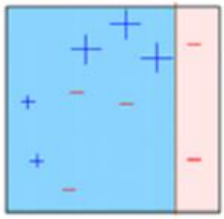

Рисунок 2.4 – Принцип роботи техніки Boosting (ч.2)

8. Подібним чином створюється кілька моделей(рис. 2.5), кожна з яких виправляє помилки попередньої моделі.
9. Кінцева модель (сильний учень) є зваженим середнім значенням усіх моделей (слабкий учень).

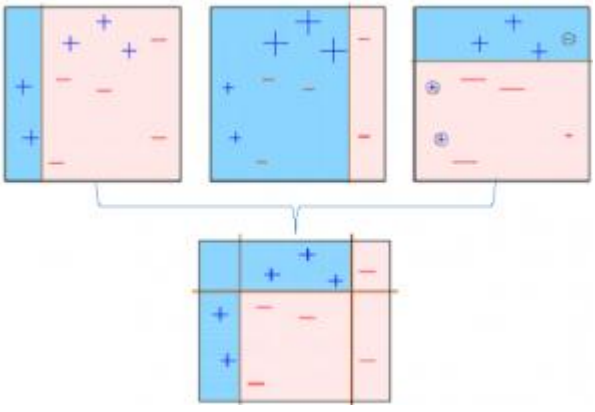

Рисунок 2.5 – Принцип роботи техніки Boosting (ч.3)

Таким чином, алгоритм підсилення поєднує ряд слабких учнів, щоб сформувати сильного учня. Окремі моделі не матимуть високої ефективності для всього набору даних, але вони добре працюють для певної частини набору даних. Таким чином, кожна модель насправді підвищує ефективність ансамблю.

Blending застосовується за тим самим підходом, що і stacking, але для прогнозування використовується лише затримка (перевірка), встановлена з набору кортежів. Іншими словами, на відміну від укладання, прогнози робляться лише на основі затримки. Набір затримки та прогнози використовуються для побудови моделі, яка запускається на тестовому наборі[8]. Опис процесу blending:

1. Комплект кортежів ділиться на навчально та перевірочний комплект, як це показано на рис. 2.6.

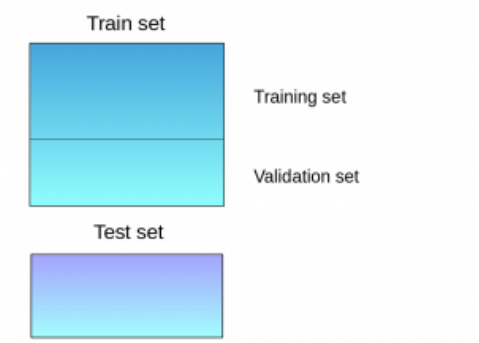

Рисунок 2.6 – Принцип роботи техніки blending (ч.1)

2. Моделі встановлені на навчальному наборі.
3. Прогнози робляться на наборі перевірки та наборі тестів(рис.2.7).

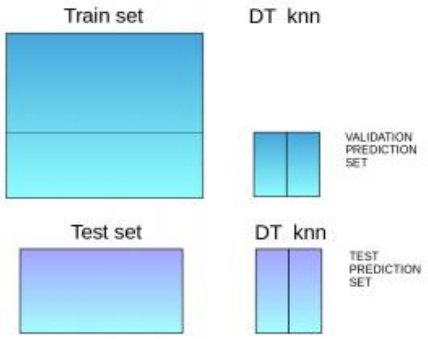

Рисунок 2.7 – Принцип роботи техніки blending (ч.2)

4. Набір перевірки та його прогнози використовуються як функції для побудови нової моделі.
5. Ця модель використовується для остаточного прогнозування тесту та мета-функцій.

Stacking - це техніка навчання ансамблю, яка використовує прогнози з декількох моделей (наприклад, дерево рішень, knn або svm) для побудови нової моделі. Ця модель використовується для прогнозування на тестовому наборі. Нижче наведено поетапне пояснення простого stacked ансамблю:

1. Комплект кортежів розділений на 10 частин (рис. 2.8).

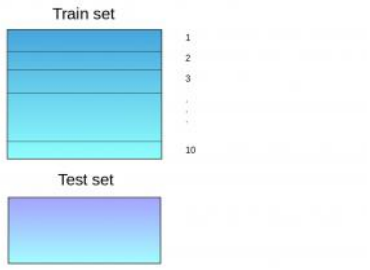

Рисунок 2.8 – Принцип роботи техніки Stacking (ч.1)

2. Базова модель (припустимо, дерево прийняття рішень) складається з 9 частин, а прогнози зроблені для 10-ї частини (рис. 2.9). Це робиться для кожної частини комплекту кортежів.

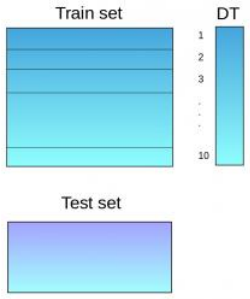

Рисунок 2.9 – Принцип роботи техніки Stacking (ч.2)

3. Потім базова модель (в даному випадку дерево рішень) встановлюється на весь набір даних про кортеж.
4. За допомогою цієї моделі (рис. 2.10) прогнози робляться на тестовому наборі.

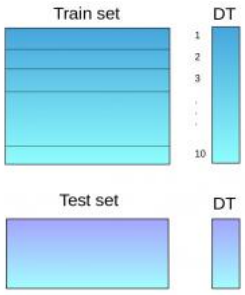

Рисунок 2.10 – Принцип роботи техніки Stacking (ч.3)

5. Етапи 2 - 4 повторюються для іншої базової моделі (скажімо, knn), що призводить до іншого набору прогнозів для складу кортежів та набору тестів (рис. 2.11).

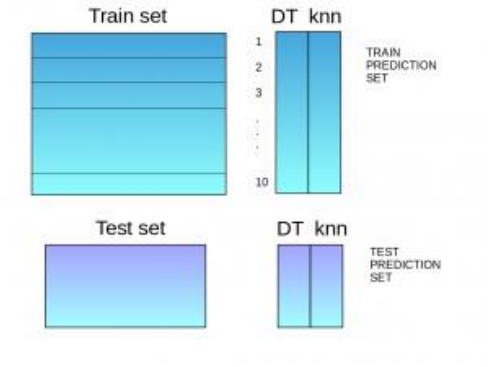

Рисунок 2.11 – Принцип роботи техніки Stacking (ч.4)

6. Прогнози з набору кортежів використовуються як особливості для побудови нової моделі.

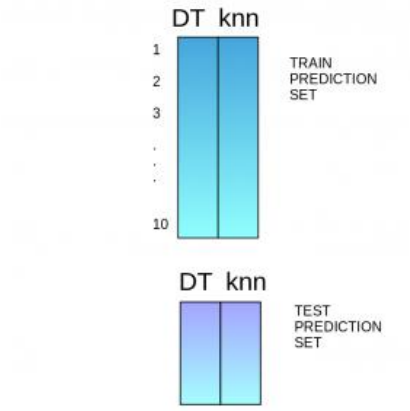

Рисунок 2.12 – Принцип роботи техніки Stacking (ч.5)

7. Ця модель (рис. 2.12) використовується для остаточного прогнозування набору тестових прогнозів.

Bagging та Boosting - дві найпоширеніші прийоми машинного навчання. Отже обирату моделі будем на основі цих двох технік. Нижче наведені реалізації алгоритмів, які використовую ці техніки:

Bagging алгоритми :

- Bagging meta-estimator
- Random forest

Boosting алгоритми:

- AdaBoost
- GBM

Bagging meta-estimator - це алгоритм складання, який використовується як для класифікації (BaggingClassifier), так і для регресії (BaggingRegressor). Для прогнозування застосовується типова техніка bagging-гу. Нижче наведено кроки для алгоритму bagging meta-estimator:

1. Випадкові підмножини створюються з вихідного набору даних (Bootstrapping).
2. Підмножина набору даних включає всі функції.
3. Вказаний користувачем базовий оцінювач встановлюється на кожному з цих менших наборів.
4. Прогнози від кожної моделі поєднуються для отримання кінцевого результату.

Приклад реалізації:

```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.75135135135135134
```

Приклад коду для проблеми регресії:

```
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor(tree.DecisionTreeRegressor(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

Параметри, що використовуються в алгоритмах:

- base_estimator:
  - Він визначає базовий оцінювач, який підходить для випадкових підмножин набору даних.
  - Коли нічого не вказано, базовий оцінювач є деревом рішень.
- n_estimators:

- Це кількість базових оцінювачів, які потрібно створити.
- Кількість оцінювачів слід ретельно налаштовувати, оскільки велика кількість зайняла б дуже багато часу, тоді як дуже мала кількість може не дати найкращих результатів.

- max_samples:
  - Цей параметр контролює розмір підмножин.
  - Це максимальна кількість вибірок для навчання кожного базового оцінювача.
- max_features:
  - Керує кількістю функцій, які слід витягувати з цілого набору даних.
  - Він визначає максимальну кількість функцій, необхідних для навчання кожного базового оцінювача.
- n_jobs:
  - Кількість завдань, які потрібно виконувати паралельно.
  - Якщо -1, кількість завдань встановлюється на кількість ядер.

- random_state:
  - Він визначає метод випадкового розбиття. Коли випадкове значення стану однакове для двох моделей, випадковий вибір однаковий для обох моделей.
  - Цей параметр корисний, коли потрібно порівняти різні моделі.

Random Forest - ще один алгоритм машинного навчання ансамблю, який відповідає техніці bagging. Базовими оцінювачами у random forest є дерева рішень. На відміну від Bagging meta-estimator, random forest випадковим чином вибирає набір ознак, які використовуються для визначення найкращого розбиття на кожному вузлі дерева рішень.

Дивлячись на це поетапно, ось що робить модель random forest:

1. Випадкові підмножини створюються з вихідного набору даних (завантаження).
2. На кожному вузлі у дереві рішень враховується лише випадковий набір ознак, що визначають найкращий поділ.
3. Модель дерева рішень підходить для кожного з підмножин.
4. Остаточний прогноз обчислюється шляхом усереднення прогнозів з усіх дерев рішень.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')
train_x = train_data.drop(columns=['Survived'],axis=1)
train_y = train_data['Survived']
test_x = test_data.drop(columns=['Survived'],axis=1)
test_y = test_data['Survived']
model = RandomForestClassifier()
model.fit(train_x,train_y)
predict_train = model.predict(train_x)
accuracy_train = accuracy_score(train_y,predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train)
predict_test = model.predict(test_x)
print('\nTarget on test data',predict_test)
accuracy_test = accuracy_score(test_y,predict_test)
print('\naccuracy_score on test dataset : ', accuracy_test)
```

Параметри:

- n_estimators:
  - Він визначає кількість дерев рішень, які потрібно створити у random forest.
  - Як правило, більша кількість робить прогнози сильнішими та стабільнішими, але дуже велика кількість може призвести до збільшення часу тренувань.
- criterion:
  - Він визначає функцію, яку слід використовувати для розбиття.

- Функція вимірює якість розділення для кожної функції та вибирає найкращий розділ.

- max_features:
  - Він визначає максимальну кількість функцій, дозволених для поділу в кожному дереві рішень.
  - Збільшення максимальних можливостей зазвичай покращує продуктивність, але дуже велика кількість може зменшити різноманітність кожного дерева.
- max_depth:
  - Random forest має кілька дерев рішень. Цей параметр визначає максимальну глибину дерев.
- min_samples_split:
  - Використовується для визначення мінімальної кількості зразків, необхідних у листовому вузлі перед спробою розділення.
  - Якщо кількість вибірок менше необхідної кількості, вузол не розбивається.
- min_samples_leaf:
  - Це визначає мінімальну кількість зразків, необхідних для розміщення на листовому вузлі.
  - Менший розмір листка робить модель більш схильною до фіксації шуму в даних поїздів.
- max_leaf_nodes:
  - Цей параметр визначає максимальну кількість листових вузлів для кожного дерева.
  - Дерево припиняє розщеплення, коли кількість листових вузлів стає рівним максимальному листовому вузлу.
- n_jobs:

- Це вказує на кількість завдань, які потрібно виконувати паралельно.
- Якщо -1, кількість завдань встановлюється на кількість ядер.

- random_state:
  - Цей параметр використовується для визначення випадкового вибору.
  - Він використовується для порівняння між різними моделями.

AdaBoost - один з найпростіших алгоритмів підсилення. Зазвичай для моделювання використовують дерева рішень. Створюється кілька послідовних моделей, кожна з яких виправляє помилки з останньої моделі. AdaBoost призначає ваги спостереженням, які були неправильно передбачені, і наступна модель працює, щоб правильно передбачити ці значення.

Нижче наведені кроки для виконання алгоритму AdaBoost:

1. Спочатку всі спостереження в наборі даних мають однакові ваги.
2. Модель побудована на підмножині даних.
3. За допомогою цієї моделі прогнози робляться для всього набору даних.
4. Помилки обчислюються шляхом порівняння прогнозів та фактичних значень.
5. Під час створення наступної моделі вищі ваги надаються точкам даних, які були передбачені неправильно.
6. Ваги можна визначити, використовуючи значення помилки. Наприклад, чим більше похибка, тим більше вага, присвоєна спостереженню.
7. Цей процес повторюється доти, доки функція помилки не зміниться або не буде досягнуто максимальне обмеження кількості оцінювачів.

Приклад реалізації:

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
```

```
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.81081081081081086
```

Приклад коду для проблеми регресії:

```
from sklearn.ensemble import AdaBoostRegressor
model = AdaBoostRegressor()
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

Параметри:

- base_estimators:
  - Це допомагає вказати тип базового оцінювача, тобто алгоритм машинного навчання, який використовуватиметься як базовий навчальний засіб.
- n_estimators:
  - Він визначає кількість базових оцінювачів.
  - Значення за замовчуванням - 10.
- learning_rate:
  - Цей параметр контролює внесок оцінювачів у кінцеву комбінацію.
  - Існує компроміс між learning_rate та n_estimators.
- max_dept:
  - Визначає максимальну глибину індивідуального оцінювача.
- n_jobs
  - Вказує кількість процесорів, яку дозволено використовувати.
  - Встановіть -1 для максимально дозволених процесорів.
- random_state:
  - Ціле значення, щоб вказати випадковий розподіл даних.
  - Визначене значення random_state завжди даватиме однакові результати, якщо буде надано однакові параметри та дані навчання.

Gradient Boosting або GBM - ще один алгоритм машинного навчання ансамблю, який працює як для регресії, так і для проблем класифікації. GBM використовує техніку boosting, поєднуючи низку слабких учнів, щоб сформувати сильного учня. Кожне наступне дерево регресії послідовно будується на основі помилок, обчислених попереднім деревом.

Приклад реалізації:

```
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.81621621621621621
```

Приклад коду для проблеми регресії:

```
from sklearn.ensemble import GradientBoostingRegressor
model= GradientBoostingRegressor()
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

Параметри

- min_samples_split
  - Визначає мінімальну кількість зразків (або спостережень), які потрібні у вузлі для розгляду.
  - Використовується для контролю надмірного припасування. Більш високі значення заважають моделі вивчати відносини, які можуть бути дуже специфічними для конкретної вибірки, обраної для дерева.
- min_samples_leaf
  - Визначає мінімальні зразки, необхідні для термінального або листового вузла.
  - Як правило, нижчі значення слід вибирати для дисбалансованих класових проблем, оскільки регіони, в яких клас меншин буде більшістю, будуть дуже малі.
- min_weight_fraction_leaf

        - Подібно до min_samples_leaf, але визначається як частка від загальної кількості спостережень замість цілого числа.
    - max_depth
        - Максимальна глибина дерева.
        - Використовується для контролю надмірного припасування, оскільки більша глибина дозволить моделі вивчити відносини, дуже специфічні для конкретного зразка.
        - Потрібно налаштувати за допомогою резюме.
    - max_leaf_nodes
        - Максимальна кількість кінцевих вузлів або листків у дереві.
        - Можна визначити замість max_depth. Оскільки створюються бінарні дерева, глибина «n» дасть максимум 2 ^ n листків.
        - Якщо це визначено, GBM буде ігнорувати max_depth.
    - max_features
        - Кількість функцій, які слід враховувати під час пошуку найкращого розділення. Вони будуть вибрані випадковим чином.
        - Більш високі значення можуть призвести до надмірного пристосування, але це, як правило, залежить від конкретного випадку.

Так как темою дослідження є класифікація моделей на основі заданих параметрів, алгоритми для навчання будуть наступними:

- GradientBoostingClassifier
- RandomForestClassifier
- AdaBoostClassifier
- BaggingClassifier

Після проведення навчання та оптимізації параметрів, буде обрано найкращий алгортим для рішення поставленої задачі.

2.3 Вибір методів оптимізації гіперпараметрів моделей

Вибір правильних гіперпараметрів для моделей машинного - це один із найкращих способів покращити результати моделей.

Налаштування гіперпараметрів - це процес визначення правильної комбінації гіперпараметрів, що дозволяє моделі максимізувати свою продуктивність. Встановлення правильної комбінації гіперпараметрів - єдиний спосіб отримати максимальну продуктивність із моделей.

Вибір правильної комбінації гіперпараметрів - завдання непросте. Є два способи їх встановити.

Ручне налаштування гіперпараметрів: У цьому методі різні комбінації гіперпараметрів встановлюються (і експериментуються) вручну. Це виснажливий процес і не може бути практичним у випадках, коли є багато гіперпараметрів, які потрібно спробувати.

Автоматизоване налаштування гіперпараметрів: У цьому методі оптимальні гіперпараметри знаходять за допомогою алгоритму, який автоматизує та оптимізує процес.

Розглянемо декілька популярних методів оптимизації гіперпараметрів на сьогодні:

Random Search – у цьому методі створюється сітка можливих значень для гіперпараметрів[14]. Кожна ітерація пробує випадкову комбінацію гіперпараметрів із цієї сітки, реєструє продуктивність і, нарешті, повертає комбінацію гіперпараметрів, яка забезпечила найкращу продуктивність.

У методі Grid Search створюється сітка можливих значень для гіперпараметрів. Кожна ітерація пробує комбінацію гіперпараметрів у певному порядку. Він підходить для моделі на кожній можливій комбінації гіперпараметра та реєструє продуктивність моделі. Нарешті, він повертає найкращу модель з найкращими гіперпараметрами.

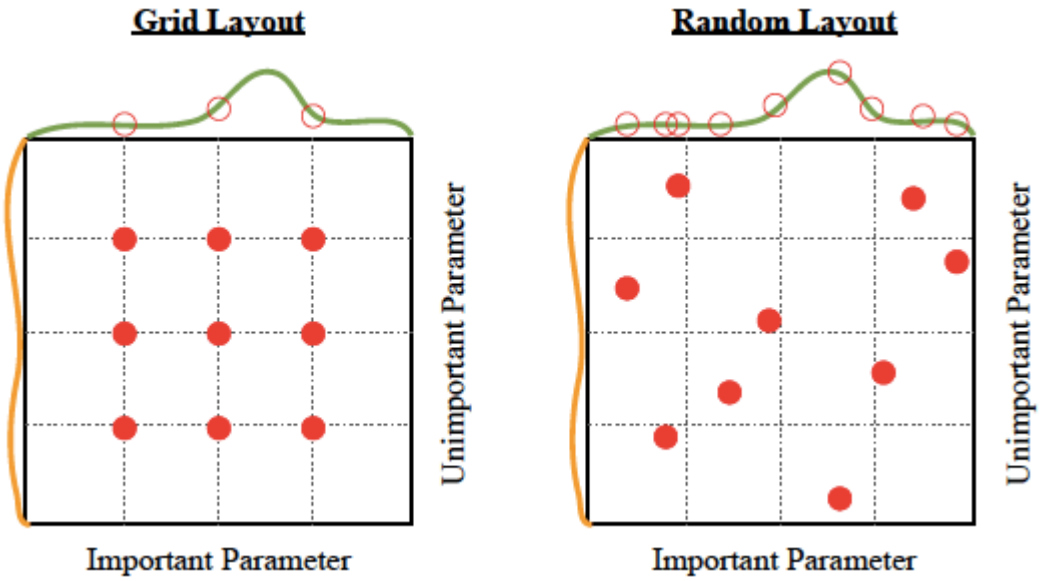

Рисунок 2.13 – Метод оптимізації Grid Search

Scikit-learn буде використнний як основний інструмент, так як він має реалізації для grid search та random search.

Для обох цих методів scikit-learn навчає та оцінює модель у k-кратному перехресному підтвердженні за різними варіантами вибору параметрів та повертає найкращу модель.

Зокрема:

Random search: за допомогою randomsearchcv[15] виконує пошук за деякою кількістю випадкових комбінацій параметрів

Grid Search: gridsearchcv[16] виконує пошук за всіма наборами параметрів у сітці.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

Для реалізції та тестування використовувалась мова Python, версії 3.8.0. Як середовища розробки, була використана безкоштовна версія PyCharm 2020.2.3 від компаніїї JetBrains. Реалізація алогритмів для тестування, таких як RandomForest, Bagging meta-estimator, Ada Boosting, GBM була взята з пакету scikit-learn, версії 0.23.2.

Для порівняння результатів, використовувалась метрика точності(accuracy_score з бібліотеки sklearn.metrics), вона повертає дробне число, яке обчислюється за формулою – (кількість правильно класифіцированих спостоережень)/(загальний обсяг спосторежень).

Для гріфачного відображення результатів для кожного типу пристрою, була використана матриця помилок(confusion_matrix)

3.1 Підготовка вхідних данних

Набір даних, використаний для цього дослідження був взятий з github[17], в основному він зібраний з 10 різних пристроїв IOT: монітор дитини, ліхтарі, датчик руху, камера безпеки, детектор диму, розетка, термостат, телевізор, годинник та датчик води.

Набір даних містить інформацію про мережевий трафік цих пристроїв, зібраний протягом тривалого періоду часу. Кожен екземпляр у наборі даних представляє сеанс з'єднання (TCP-з'єднання з пакета SYN у пакет FIN). Залежною змінною є класифікація пристрою відповідно до його типу.

Набір навчальних матеріалів містить приблизно 400 000 екземплярів та майже 300 функцій.

Після аналізу данних було виявлено, що не всі дані були доступні для всіх заданих сеансів у наборі даних. Досить часто можна зустріти набір даних, в якому не всі дані доступні і можуть бути використані для навчання. Існують різні підходи до того, як поводитися з цими відсутніми даними, і підхід, якій був застосований, - це видалення екземплярів із відсутніми даними. Всі відсутні

дані у вихідному наборі даних представлені знаком питання, отже для їх видалення був написаний наступний код:

```
import warnings
import pandas as pd

warnings.simplefilter(action='ignore', category=FutureWarning)
data_path = "./data/train.csv"
data = pd.read_csv(data_path, low_memory=False, delimiter=',')
for col in data.columns:
    if data[col].unique().__contains__("?"):
        data = data.drop(data[data[col] == "?"].index)

data.reset_index()
data.to_csv("input_data.csv", index=False)
```

Використання warnings.simplefilter було необхідним, так як при читанні файлу train.csv падали помилки Pandas FutureWarning, цей код був доданий для їх ігнорування.

Після того як записи з некоректними данними були видалені, необхідно було виконати pandas.reset_index(), для того щоб обновити нумерації записів в датафреймі.

Останім кроком було збереження датафрему для файлу на диску, для того щоб кожного разу не виконувати видалення.

Далі необхідно створити дві группи данних, одна з яких буде використоувавиться для навчання моделі, іншя для тестування. Ці группи були створені випадковим чином, за допомогою наступно кода:

```
import pandas as pd
import numpy as np

df = pd.read_csv("./data/input_data.csv")
df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75
train, test = df[df['is_train'] == True], df[df['is_train'] == False]

del train ['is_train']
```

```
del test['is_train']
train.to_csv("./data/train_data.csv", index=False)
test.to_csv("./data/test_data.csv", index=False)
```

Отже, як результат, 75% данних(is_train==true) – це данні для навчання, решта данні для тестування. Після видаллення колонки is_train, обидва датасети були збереженні до відповідних файлів.

Одним з аспектів, який слід враховувати у багатьох задачах машинного навчання, є вибір функцій. Поняття "прокляття розмірності" добре відоме і може призвести до того, що модель переоснащується або працює погано. Для боротьби з нею була введена концепція вибору об’єкта. Деякі моделі зазвичай не потребують вибору функцій, таких як Decision Trees та Random Forest. Причина полягає в тому, що процес вибору ознак виконується на льоту завдяки способу навчання цих моделей («найкраща» характеристика вибирається при кожному зрізі дерева).

Однак для досягнення кращих результатів для деяких моделей може знадобитися вибір функцій. У цій роботі помітно, що співвідношення зразків до особливостей є справді високим (у навчальному наборі приблизно 400 000 спостережень та приблизно 300 ознак), отже, «прокляття розмірності» не повинно мати великого впливу.

3.2 Навчання моделі Random Forest з параметрами за змовчуванням

Першою моделью для дослідження став Random Forest, який використовує прийоми Bagging.

За допомогою наступного коду, було навчено модель RandomForestClassifier(вхідними параметрами було встановлено лише random_state=32, та n_jobs=-1 для підвищення скорості роботи коду).

Так як обрані класифікатори працюють тільки з цілочисельними або булевими данними, то потрібно було закодувати колонку з назвою пристроїв. Для цього було використано функції pandas.factorize().

Accuracy score був рівний 0.934, confusion matrix представлена на рис. 3.1. Так-як це данна модель йде першою, її результати будуть взяті як єталонні.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
import sklearn.metrics as metrics
from sklearn.ensemble import RandomForestClassifier

train_data_path = "./data/train_data.csv"
test_data_path = "./data/test_data.csv"

train_data = pd.read_csv(train_data_path)
test_data = pd.read_csv(test_data_path)

features = train_data.columns[:297]
train_label = pd.factorize(train_data['device_category'])[0]
test_label = pd.factorize(test_data['device_category'])[0]

model = RandomForestClassifier(random_state=32, n_jobs=-1)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])
print("Default parameters for RandomForestClassifier")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))

preds = train_data.device_category.unique()[predictions]
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'], colnames=['Передбачувані данні'])
print(confusion_matrix)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```

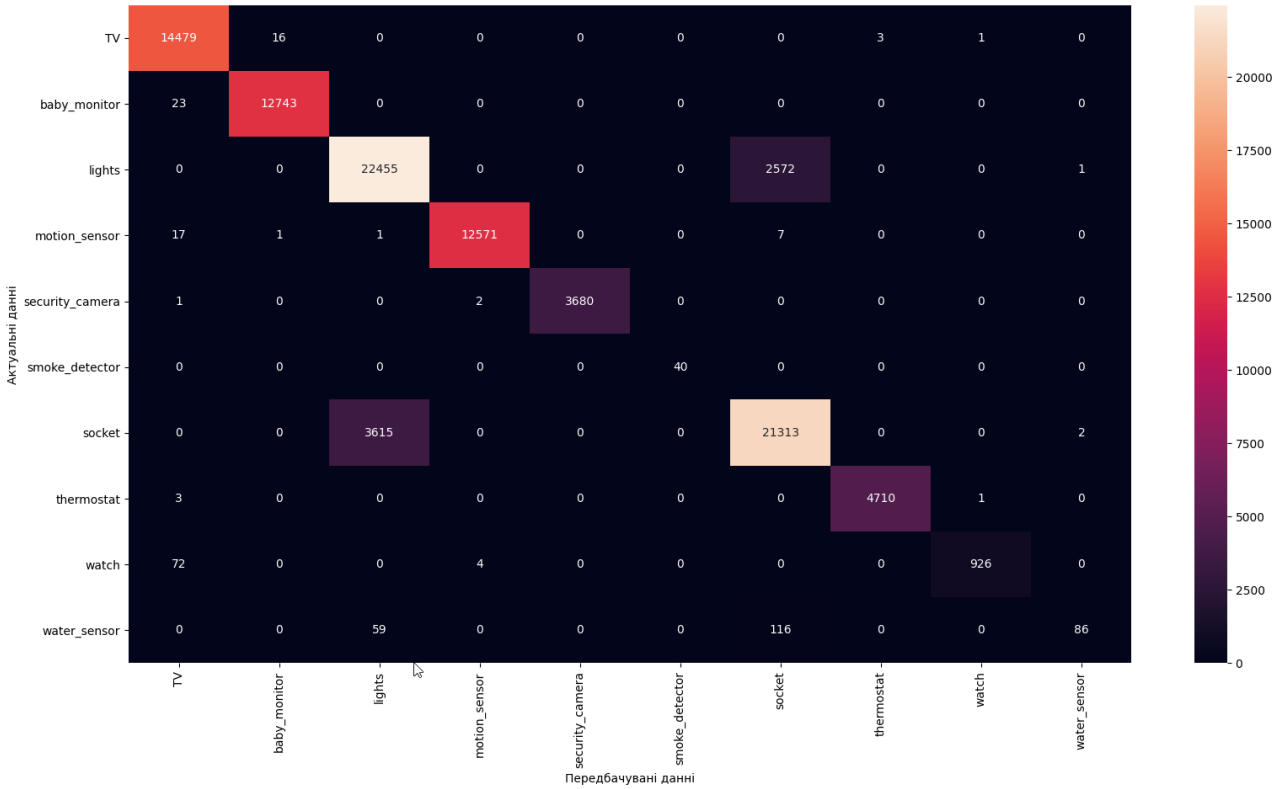

Рисунок 3.1 – Confusion Matrix для Random Forest

3.3 Навчання моделі Bagging meta-estimator з параметрами за змовчуванням

Другою моделью, яка використовую прийоми Bagging, є Bagging meta-estimator.

```
model = BaggingClassifier(random_state=32 , n_jobs=-1)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])

print("Default parameters for Bagging meta-estimator ")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))
preds = train_data.device_category.unique()[predictions]
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'],
                               colnames=['Передбачувані данні'])
print(confusion_matrix)
```

```
sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```

Accuracy score данної моделі рівний 0.9301(що на 0.0039 гірше ніж результат моделі RandomForest), confusion matrix представлена на рис. 3.2.

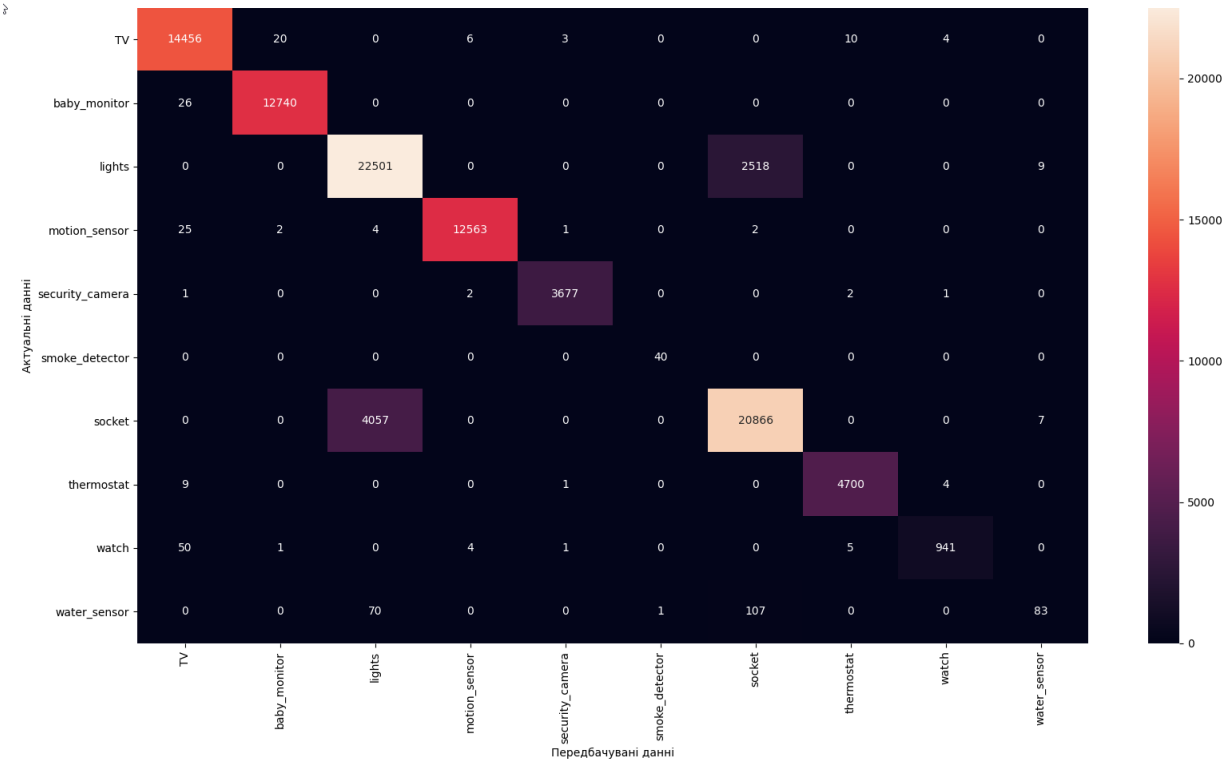

Рисунок 3.2 – Confusion Matrix для Bagging meta-estimator

3.4 Навчання моделі Gradient Boosting з параметрами за змовчуванням

Першою модель зі списку для тестування яка використовує принцип Boosting, є модель GradientBoostingClassifier або GBM. В порівнянні з моделями які використовую принцип Bagging, процесс виконання навчання та передбачення зайняв набагато більше часу(приблизно в 5-7 разів довше).

```
model = GradientBoostingClassifier(random_state=32)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])

print("Default parameters for RandomForestClassifier")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))
preds = train_data.device_category.unique()[predictions]
```

```
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'], colnames=['Передбачувані данні'])
print(confusion_matrix)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```

Accuracy score данної моделі рівний 0.9213 (що на 1.3% гірше, ніж єталон), confusion matrix представлена на рис. 3.3.

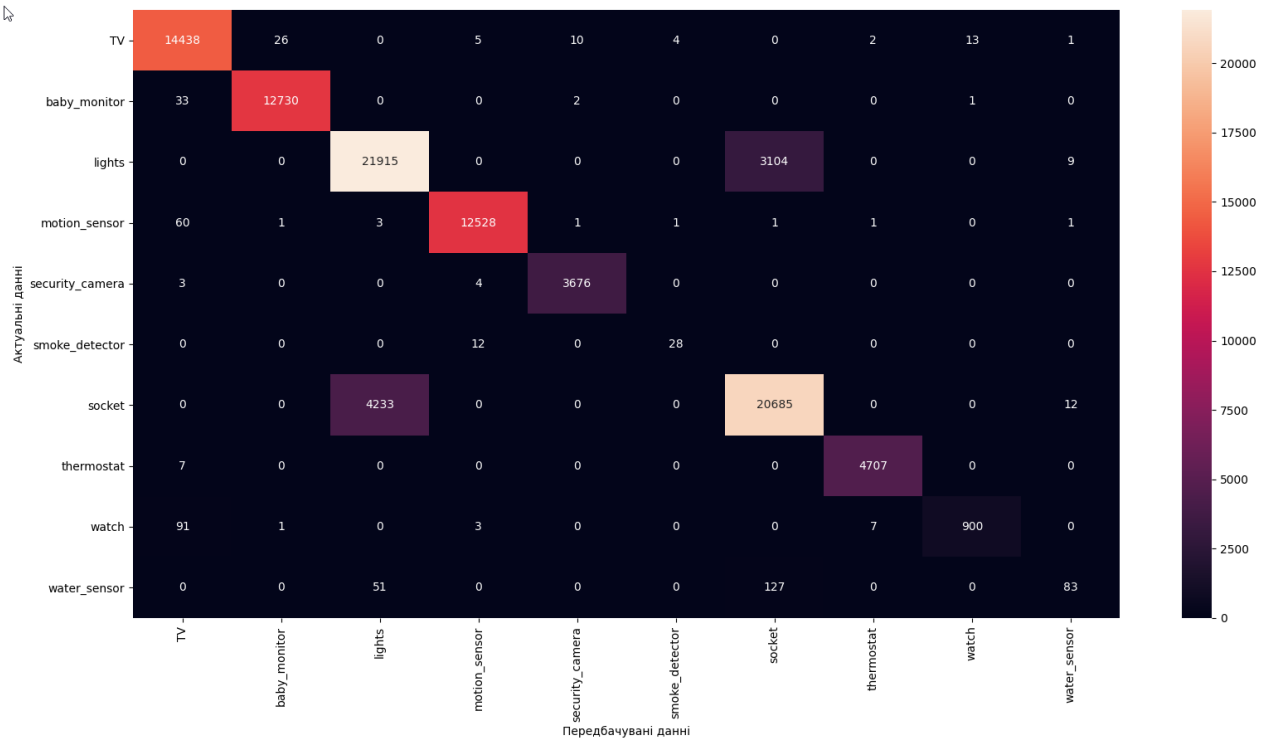

Рисунок 3.3 – Confusion Matrix для GradientBoosting

3.5 Навчання моделі Ada Boosting з параметрами за змовчуванням

Останім алгоритмом для тествування став AdaBoosting, та його реалізація AdaBoostClassifier.

```
model = AdaBoostClassifier(random_state=32)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])

print("Default parameters for AdaBoostClassifier")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))
```

```
preds = train_data.device_category.unique()[predictions]
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'], colnames=['Передбачувані данні'])
print(confusion_matrix)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```

Accuracy score данної моделі рівний 0.49001(що майже на 50% гірше ніж результат єталонної модели), confusion matrix представлена на рис. 3.4. В порівнянні з іншими моделями, данний результат найгірший, і можна зробити висновок, що з параметрами за змовчанням данна модель не підходить для представленного набору данних.

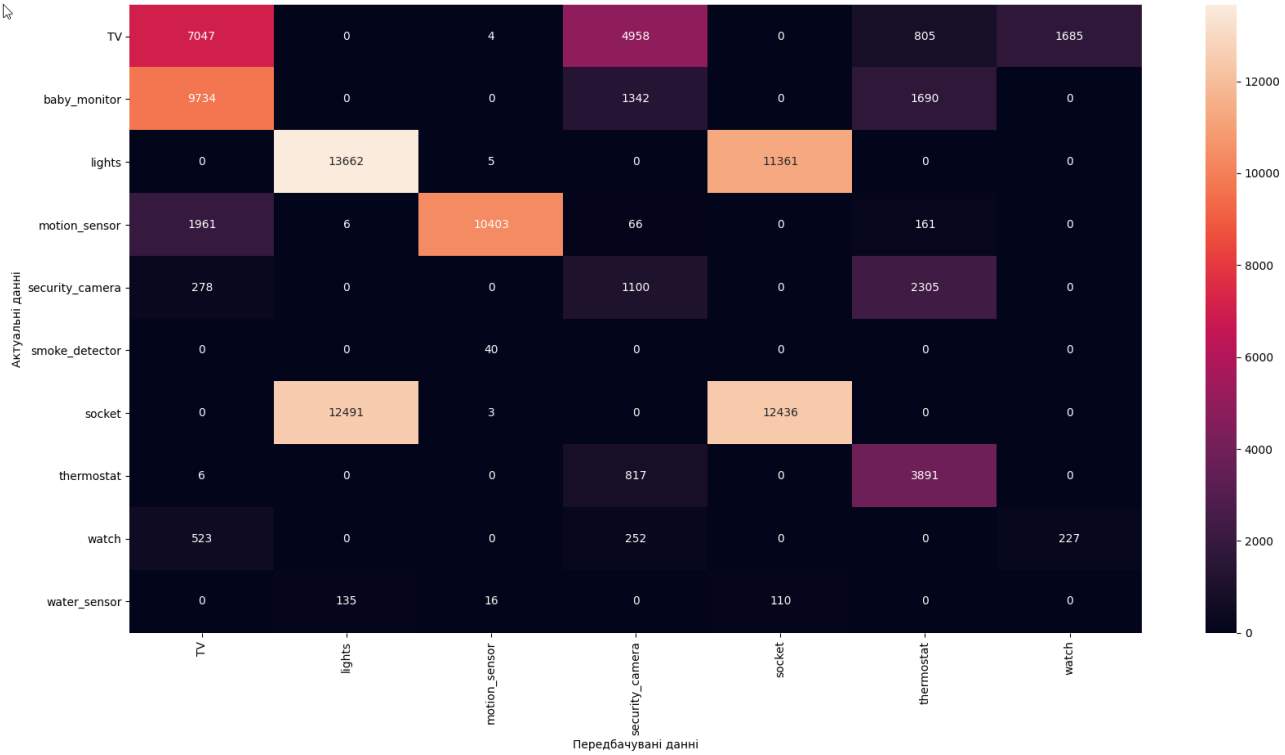

Рисунок 3.4 – Confusion Matrix для Ada Boosting

3.6 Оптимізація гіперпараметрів моделей за допомогою GridSearch

Бібліотека, з якої була взята реалізація GridSearch – sklearn.model_selection.GridSearchCV.

В першу чергу, необхідно визначити перелік параметрів для запуску пошуку, а так як для кожної моделі набір параметрів різний, то в сумі було визначено 4 таблиці параметрів.

Для моделі RandomForest:

```
param_grid = {"n_estimators": [50, 100, 250, 300],
              "criterion": ["gini", "entropy"],
              "max_features": [1, 3, 5, 10],
              "max_depth": [5, 10, 20],
              "min_samples_split": [2, 4, 6],
              "bootstrap": [True, False],
              "min_samples_leaf": [1, 2, 3, 4, 5]
              }
```

Для моделі Bagging meta-estimator:

```
param_grid = {"n_estimators": [50, 100, 250, 300],
              "criterion": ["gini", "entropy"],
              "max_features": [1, 3, 5, 10],
              "max_samples": [5, 10, 20],
              }
```

Для моделі Ada Boosting:

```
param_grid = {"n_estimators": [50, 100, 250, 300],
              "learning_rate": [1, 5, 10, 20],
              }
```

Для моделі Gradient Boosting:

```
param_grid = {"max_depth": [5, 10, 20],
              "min_samples_split": [2, 5, 10],
              "min_samples_leaf": [1, 3, 6],
              "min_weight_fraction_leaf": [0, 2, 5],
              "max_features": [1, 3, 5, 10],
              }
```

Для старту пошуку оптимальних параметрів, необхідно створити єкземляр моделі, та передати його як параметр estimator в код нижче.

```
CV_rfc = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_jobs=-1)
CV_rfc.fit(train_data[features], train_label)
print(CV_rfc.best_params_)
```

Після запуску коду, програма виведе на єкран оптимальні парамери для кожної моделі(необхідно зазначит, що час пошуку оптимальних параметрів для кожної моделі значно відрізнявся, від 3-4 годин, до 24 годин включно, відходячи від цього, обовязково необхідно використовувати параметр n_jobs=-1, для того щоб система виконувала пошук в максильну кількість процессів).

Для кожної моделі набір оптимальних парметрів, повернених від GridSearchCV, значно відрізнявся:

Random Forest - n_estimators - 100, criterion – gini, max_features – 5, max_depth – 10, min_samples_split – 6, bootstrap – True, - min_samples_leaf – 5;

Bagging meta-estimator - n_estimators - 100, criterion - gini, max_features - 10, max_samples – 10;

Ada Boosting - n_estimators - 250, learning_rate – 10;

Gradient Boosting - max_depth - 20, min_samples_split - 5, min_samples_leaf - 6, min_weight_fraction_leaf - 5, max_features – 5;

Після запуску кожної моделі з набором оптимальних параметрів, точність для деяких з них значно покращилась, результат у таблиці 3.1.

Таблиця 3.1. – Результат після оптимізації гіперпарамерів

Назва моделі	**Точність з пар. за змовчуванням**	**Точність з параметрами (GridSearch)**	**% покращення**
Random Forest	0.934	0.9405	0.69
Bagging meta-estimator	0.9301	0.9352	0.54
Ada Boosting	0.49001	0.51	4
Gradient Boosting	0.9213	0.9432	2.37

Висновок: після проведення тестів, найкращий результат відносно точності був показаний для методу Gradient Boosting(на 2.37% краще ніж її перший результат). В порівнянні з результатами передбачення для моделей з параметрами за змовчуванням, модель GBM покращила свій результат, перегнавши результати інших моделей після оптимізації.

Найгірша ж модель - Ada Boosting, покращила свій результат лише на 4% від попереднього, що в цілому добре, але на фоні інших моделей цей результат однозначно найгірший.

3.7 Оптимізація гіперпараметрів моделей за допомогою RandomSearch

Бібліотека, з якої була взята реалізація Random Search

– sklearn.model_selection.RandomizedSearchCV.

Для проведення пошуку оптимальних гіперпараметрів необхідно визначити перелік параметрів, з яких модель будет брати параметри для пошуку.

Параметри для моделей будут обрані випадковим чином из заданих інтервалів, після чого модель RandomizedSearchCV протестує кожен з них, і поверне найкращий результат.

```
n_estimators = [int(x) for x in numpy.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in numpy.linspace(10, 110, num=11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
rf = RandomForestClassifier()
random_search_cv = RandomizedSearchCV(estimator=rf,
param_distributions=random_grid, n_iter=100, cv=3, verbose=2,
                                      random_state=42, n_jobs=-1)
random_search_cv.fit(train_features, train_labels)
print(random_search_cv.best_params_)
```

Наступни гіперпараметри були обрані RandomizedSearchCV, як найкращі з данного діпазону.

Random Forest - n_estimators - 400, max_features – `sqrt`, max_depth – 40, min_samples_split – 5, bootstrap – True, - min_samples_leaf – 2;

Bagging meta-estimator - n_estimators - 200, max_features - 4, max_samples – 6;

Ada Boosting - n_estimators - 600, learning_rate – 20;

Gradient Boosting - max_depth - 10, min_samples_split - 6, min_samples_leaf - 6, min_weight_fraction_leaf - 4, max_features – 10;

Після проведення повторонго запуску для кожної моделі, результат був представленний в таблиці 3.2 нижче.

Таблиця 3.2. – Результат після оптимізації гіперпарамерів

Назва моделі	**Точність з пар. за змовчуванням**	**Точність з параметрами (RandomizedSearch)**	**% покращення**
Random Forest	0.934	0.9369	0.3
Bagging meta-estimator	0.9301	0.9158	-1.5
Ada Boosting	0.49001	0.48959	-0.08
Gradient Boosting	0.9213	0.9354	1.5

Висновок: при тестування моделей з новими параметрами, для деяких з них було помічено погіршення в результатах(Bagging meta-estimator на 1.5% та Ada Boosting на 0.08%). Модель Gradient Boosting покращила свій результат, у порівнянні з запуском без параметрів, а всього на 1.5%, що є кращим результатом для оптимізації за допогомою Random Search, але не кращим загалом.

3.8 Результат

Після проведення трох ітерацій навчання та тестування для 4 різних моделей (Random Forest, Bagging meta-estimator, Ada Boosting Gradient Boosting), було виявленно наступне:

1. При тестуванні без параметрів, 3 з 4 моделей показали приблизно однаковий результат(Random Forest з яких показала найкращий, з точність. 0.934);

2. При тестуванні після оптимізаціїї гіперапараметрів за допомогою GridSearch модель Gradient Boosting значно покращила свій результат, повернувши найбільше значення точності з усіх ітерацій тестів;
3. При тестуванні після оптимізаціїї гіперапараметрів за допомогою RandomizedSearch, 2 з 4 моделей погіршили свій результат, Bagging meta-estimator та Ada Boosting, на 1.5% и 0.08% відповідно;
4. За весь час тестування, модель Ada Boosting не показала задовільного результату (максимально 0.51 точності), вона однозначно не підходить для рішення поставленої задачі;
5. Модель Gradient Boosting, яка показала найкращий результат після оптимізації, однозначно є фаворитом. Припускается, що точність можливо довести до значения 1, якщо вхідні данні, будуть точніше зібрана, та на знаходження оптимальних параметрів буде витрачено не одна тисяча ітераций пошуку.

ВИСНОВКИ

У цій роботі було досліджено питання аутентифікації пристроїв інтернету речей. Проаналізовані способи аутентифікації, такі як одностороння аутентифікація або двостороння аутентифікація. Також розібрані більш конкретні механізми аутентифікації, та розібрані їх плюси і мінуси:

- Модуль довіреної платформи (TPM)
- Симетричні ключі
- Модуль апаратного захисту (HSM)
- Сертифікати X.509

В рамках роботи було запропоновано алгортим аутентифікації, який оснований на машинному навчанні, який буде використовувати перехопленний мережевий трафік як вхідні данні.

Було проаналізовано мови програмування, які найкраще підходять для Data Science, серед них були – Python, R, Scala та SAS.

Серед поставлених задач, було протестовано та обрано найкращий алогритм машиного навчання, для виконання поставленої задачі. Для оптимізації гіперпараметрів для алгоритму, було обрано 2 метду – Random Search та Grid Search.

Досліджено 4 алгоритми машинного навчання:

- Random Forest
- Bagging meta-estimator
- Ada Boosting
- Gradient Boosting

з використанням параметрів за змовчуванням, та параметрів після оптимізаціїї за допогомою та Random Search та Grid Search.

В результат, кращий результат точності показала модель Gradient Boosting, найгірший - Ada Boosting, яка однозначно не рекомендуеться для рішення поставленої задачі.

СПИСОК ЛІТЕРАТУРИ

1. The Algorithm Design Manual by Steve S. Skiena / Steve S. Skiena. - Springer; 2nd edition (April 5, 2009). - 730 p. - ASIN : B00B8139Z8
2. Homeland Security News Wire, моделі даних [Електронний ресурс] / Режим доступу: http://www.homelandsecuritynewswire.com/dr20141217-2008-turkish-oil-pipeline-explosion-may-have-been-stuxnet-precursor
3. U.S. Food and Drug Administration, моделі даних [Електронний ресурс] / Режим доступу: https://www.fda.gov/news-events/press-announcements/fda-outlines-cybersecurity-recommendations-medical-device-manufacturers
4. Computerworld, моделі даних [Електронний ресурс] / Режим доступу: https://www.computerworld.com/article/2932371/medjack-hackers-hijacking-medical-devices-to-create-backdoors-in-hospital-networks.html
5. Broadcom, моделі даних [Електронний ресурс] / Режим доступу: https://docs.broadcom.com/doc/istr-23-2018-executive-summary-en-aa
6. Techblognow, моделі даних [Електронний ресурс] / Режим доступу: https://techblognow.wordpress.com/2015/02/20/x-509-certificates-explained/
7. Introduction to Algorithms by Thomas H. Cormen / Thomas H. Cormen. - MIT Press; 3rd edition (September 1, 2009). - 1292p. - ISBN-10 : 9780262033848
8. Algorithms by Robert Sedgewick & Kevin Wayne / Robert Sedgewick, Kevin Wayne. - Addison-Wesley Professional; 4th edition (April 3, 2011). - 976p. - ISBN-10 : 032157351X
9. DevCentral, моделі даних [Електронний ресурс] / Режим доступу: https://devcentral.f5.com/s/articles/hardware-security-modules-17848
10. Gatevidyalay, моделі даних [Електронний ресурс] / Режим доступу: https://www.gatevidyalay.com/cryptography-symmetric-key-cryptography/
11. Python Data Science Handbook: Essential Tools for Working with Data 1st Edition / Jake VanderPlas. - O'Reilly Media; 1st edition (December 13, 2016). - 548p. - ISBN-10 : 1491912057

12. JetBrains, моделі даних [Електронний ресурс] / Режим доступу: https://www.jetbrains.com/research/python-developers-survey-2018/
13. AnalyticsVidhya, моделі даних [Електронний ресурс] / Режим доступу: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/
14. GitHub, моделі даних [Зображення] / Режим доступу: https://srdas.github.io/DLBook/DL_images/HPO1.png
15. Scikit-learn, моделі даних [Електронний ресурс] / Режим доступу: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
16. Scikit-learn, моделі даних [Електронний ресурс] / Режим доступу: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
17. GitHub, моделі даних [Архів данних] / Режим доступу: https://github.com/Mosseridan/IoT-device-type-identification/blob/master/data.7z
18. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming / Eric Matthes. -No Starch Press; Illustrated edition (May 3, 2019). - 544p. - ISBN-10 : 1593279280

ДОДАТОК А. ЛІСТИНГ КОДУ РЕАЛІЗАЦІЇ ГЕНЕРАЦІХ ВХІДНИХ ДАННИХ

Лістинг файлів generate_data.py та formate_input_data.py, які використовуються для створення та форматування вхідних данних:

```
# coding=utf-8
# generate_data
import warnings
import pandas as pd

warnings.simplefilter(action='ignore', category=FutureWarning)
data_path = "C:/Users/lobin/Desktop/diploma/data/train.csv"
data = pd.read_csv(data_path, low_memory=False, delimiter=',')
for col in data.columns:
    if data[col].unique().__contains__("?"):
        data = data.drop(data[data[col] == "?"].index)
data.reset_index()
data.to_csv("input_data.csv", index=False)

# coding=utf-8
# formate_input_data
import pandas as pd
import numpy as np

input_data = "input_data.csv"
df = pd.read_csv(input_data)
df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75
train, test = df[df['is_train'] == True], df[df['is_train'] == False]
del train ['is_train']
del test['is_train']
train.to_csv("./data/train_data.csv", index=False)
test.to_csv("./data/test_data.csv", index=False)
```

ДОДАТОК Б. ЛІСТИНГ КОДУ РЕАЛІЗАЦІЇ МОДЕЛІ RANDOM FOREST

Лістинг файлу random_forest.py, за допомгою якого навчається та тестується модель Random Forest.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
import sklearn.metrics as metrics
from sklearn.ensemble import RandomForestClassifier

train_data_path = "./data/train_data.csv"
test_data_path = "./data/test_data.csv"

train_data = pd.read_csv(train_data_path)
test_data = pd.read_csv(test_data_path)

features = train_data.columns[:297]
train_label = pd.factorize(train_data['device_category'])[0]
test_label = pd.factorize(test_data['device_category'])[0]

model = RandomForestClassifier(random_state=32, n_jobs=-1)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])

print("Default parameters for RandomForestClassifier")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))
preds = train_data.device_category.unique()[predictions]
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'],
                               colnames=['Передбачувані данні'])
print(confusion_matrix)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```

ДОДАТОК В. ЛІСТИНГ КОДУ РЕАЛІЗАЦІЇ МОДЕЛІ GRADIENT BOOSTING

Лістинг файлу gradient_boosting.py, за допомгою якого навчається та тестується модель Gradient Boosting.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sn
import sklearn.metrics as metrics
from sklearn.ensemble import GradientBoostingClassifier

train_data_path = "./data/train_data.csv"
test_data_path = "./data/test_data.csv"

train_data = pd.read_csv(train_data_path)
test_data = pd.read_csv(test_data_path)

features = train_data.columns[:297]
train_label = pd.factorize(train_data['device_category'])[0]
test_label = pd.factorize(test_data['device_category'])[0]

model = GradientBoostingClassifier(random_state=32)
x = train_data[features]
y = train_label
model.fit(x, y)

predictions = model.predict(test_data[features])

print("Default parameters for GradientBoostingClassifier")
print("accuracy_score: ", metrics.accuracy_score(test_label, predictions))
preds = train_data.device_category.unique()[predictions]
confusion_matrix = pd.crosstab(test_data['device_category'], preds,
rownames=['Актуальні данні'],
                               colnames=['Передбачувані данні'])
print(confusion_matrix)

sn.heatmap(confusion_matrix, annot=True, fmt="d")
plt.show()
```