НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено

Завідувач кафедри

_____ **Дмитро ЛАНДЕ**

(підпис)

«_____»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: Модель фільтрації VEC листів з використанням машинного навчання

Виконав: здобувач вищої освіти **IV** курсу, групи ФБ-91
(шифр групи)

Зібаров Дмитро Владиславович
(прізвище, ім'я, по батькові) (підпис)

Керівник Козленко Олег Віталійович
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Здобувач вищої освіти _____
(підпис)

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти
Зібаров Дмитро Владиславович

(прізвище, ім’я, по батькові)

1. Тема роботи

“Модель фільтрації VEC листів з використанням машинного навчання”

,

керівник роботи

Козленко Олег Віталійович, асистент

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 26 » травня 2023 р. № 2028-С

2. Термін подання здобувачем вищої освіти роботи « 9 » червня 2023 р.

3. Вихідні дані до роботи Набір даних з прикладами VEC листів

4. Зміст роботи Теоретичні відомості про vendor email compromise листи, методи попередньої обробки текстових даних, методи машинного навчання, метрики для оцінки моделей машинного навчання, аналіз впливу різних методів на результат роботи моделей, виявлення найбільш успішного підходу та пошук можливих варіантів покращення роботи моделі.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Дата видачі завдання 15.01.2023_____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	15.02.2023	виконано
2	Огляд та опрацювання інформаційних джерел	15.02.2023 - 05.03.2023	виконано
3	Збір даних для створення набору даних	06.03.2023 - 15.03.2023	виконано
4	Підготовка даних	15.03.2023 - 18.03.2023	виконано
5	Планування підходу до вирішення задачі	18.03.2023 - 12.04.2023	виконано
6	Написання програмного коду для попередньої обробки текстових даних	12.04.2023 - 20.04.2023	виконано
7	Написання програмного коду для дослідження впливу попередньої обробки тексту на результати роботи моделей машинного навчання	20.04.2023 - 05.05.2023	виконано
8	Вивчення матеріалів для покращення найбільш результативного підходу	05.05.2023 - 18.05.2023	виконано
9	Перевірка різних варіантів покращення моделі	19.05.2023 - 31.05.2023	виконано
10	Оформлення дипломної роботи	01.06.2023 - 09.06.2023	виконано

Здобувач вищої освіти _____ Зібаров ДМИТРО
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Олег КОЗЛЕНКО
(підпис)

(**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Обсяг роботи 44 сторінок, 19 ілюстрацій, 6 таблиць, 4 додатки, 7 джерел літератури. В роботі проведено аналіз різних методів попередньої обробки тексту та їхній вплив на різні методи машинного навчання, виявлено найбільш оптимальний підхід для класифікації Vendor Email Compromise (VEC) листів а також запропоновано рішення для покращення результатів найефективнішого варіанту.

Метою даної роботи є виявлення та реалізація оптимального підходу для виявлення VEC листів з найменшою кількістю хибно-позитивних (FP) класифікацій.

Об'єктом дослідження даної роботи є методи попередньої обробки тексту, методи машинного навчання для виявлення VEC листів.

Предметом дослідження даної роботи є вплив попередньої обробки тексту на ефективність класифікації VEC листів, виявлення найбільш ефективного підходу для виявлення VEC листів та варіанти покращення результатів.

Ключові слова: Business email compromise, BEC, Vendor email compromise, VEC, машинне навчання, стемінг, лематизація, видалення стоп-слів, багатошаровий перцептрон, логістична регресія, випадковий ліс, k найближчих сусідів, метод опорних векторів, крос-валідація.

ABSTRACT

Work summary 44 pages, 19 illustrations, 6 tables, 4 appendices, 7 sources of literature. The work analyzes various text preprocessing methods and their impact on various machine learning methods, reveals the most optimal approach for the classification of Vendor Email Compromise (VEC) letters, and also proposes a solution to improve the results of the most effective option.

The purpose of this work is to identify and implement an optimal approach for detecting VEC sheets with the least number of false-positive (FP) classifications.

The object of research of this work is text preprocessing methods, machine learning methods for detecting VEC letters.

The subject of research in this paper is the impact of text preprocessing on the efficiency of VEC sheet classification, the identification of the most effective approach for VEC sheet detection, and options for improving the results.

Keywords: Business email compromise, BEC, Vendor email compromise, VEC, machine learning, stemming, lemmatization, stop word removal, multilayer perceptron, logistic regression, random forest, k nearest neighbors, support vector method, cross-validation.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ML – Machine Learning (машинне навчання)

VEC – Vendor Email Compromise (компрометація електронної пошти постачальника послуг)

BEC – Business Email Compromise (компрометація бізнес-електронної пошти)

MLP – MultiLayer Perceptron (багатошаровий перцептрон)

LBFGS – Limited-memory алгоритм Бройдена-Флетчера-Гольдфарба-Шанно

SVM – Support Vector Machine (метод опорних векторів)

KNN – k-nearest neighbors (k найближчих сусідів)

TP – True Positive (істинно позитивний)

FP – False Positive (хибно позитивний)

TN – True Negative (істинно негативний)

FN – False Negative (хибно негативний)

ВСТУП

VEC – це вид інтернет-шахрайства, який полягає в тому, що зловмисник видає себе за третю сторону (постачальника послуг) з метою ввести клієнтів цього постачальника в оману. Зловмисники користуються відкритими джерелами для збору інформації: імена співробітників постачальника, їхні електронні пошти, логотипи, клієнти, адреси клієнтів, а потім, користуючись зібраними даними представляються автоматизованою розсилкою або співробітником компанії. Зазвичай на перший погляд такі листи виглядають правдоподібно, але при більш детальному розгляді одразу помічаються недоліки такого простого підходу: помилки в тексті; неспівпадаючий домен відправника та компанії, частиною якої він представляється; підозрілий текст повідомлення.

Виявлення VEC методами машинного навчання (ML) є дуже актуальним, оскільки традиційні інструменти безпеки, такі як фільтри спаму або антивіруси, не можуть ефективно розпізнавати такого роду атаки, які часто не мають шкідливих URL-адрес або вкладень. ML може допомогти аналізувати та розпізнавати типи фішингових електронних листів, які використовуються у BEC, та попереджати IT-відділи та користувачів про небезпеку. Крім того, ML може допомогти виявляти аномалії у поведінці користувачів електронної пошти, такі як зміна мови, стилю або частоти листування.

Таким чином, використання ML для виявлення BEC є необхідним для захисту організацій від таких шкодочинних атак, які можуть призвести до великих фінансових та репутаційних збитків.

В даній роботі будуть розглянуті декілька різних методів для класифікації електронних листів, а також застосовані та перевірені на ефективність різні методи попередньої обробки тексту. Також, буде запропоновано рішення проблеми недостатньої ефективності моделі, що дозволить адаптувати систему до нових видів шкідливих листів та зменшити кількість FP та FN реакцій.

Мета і завдання дослідження:

Об'єкт дослідження – Vendor Email Compromise (компрометація електронної пошти постачальника послуг).

Предмет дослідження – методи попередньої обробки природної мови та методи машинного навчання для класифікації VEC листів.

Мета роботи – виявлення найбільш ефективної архітектури для попередньої обробки даних, тренування та застосування моделі машинного навчання.

Завдання дослідження:

1. Вивчення природи VEC листів, виявлення індикаторів, що вказують на небезпеку листа та існуючих методів для виявлення листів цього типу.
2. Збір даних для тренування моделі та підготовка цих даних для навчання моделі.
3. Вивчення існуючих методів попередньої обробки даних.
4. Розробка програмного забезпечення для попередньої обробки даних та класифікації електронних листів.
5. Аналіз метрик точності класифікаційних моделей з різними підходами до попередньої обробки даних, виявлення найкращих моделей та методів для цього набору даних.
6. Розробка механізму для поліпшення найкращого підходу для класифікації VEC листів.

Методи дослідження

Під час виконання дослідження застосовувалися такі методи, як: літературний огляд (дослідження існуючої інформації про особливості VEC листів та методи боротьби з ними, методи попередньої обробки тексту та методи машинного навчання для класифікації текстових даних), аналіз (виявлення закономірностей в VEC листах та вивчення їх особливих ознак),

моделювання (створення програмного коду для тестування моделей), класифікація (виявлення шкідливих листів типу VEC за допомогою створених моделей),вимірювання (отримання метрик моделей), порівняння (аналіз точностей різних моделей в різних умовах для визначення найбільш ефективного підходу).

Практичне значення одержаних результатів

В процесі створення даної роботи було створено програмний застосунок, що дозволить швидко виявляти підозрілі листи а також покращує свою продуктивність за рахунок допомоги зі сторони аналітика.

Апробація результатів роботи та публікацій

Робота була опублікована у Всеукраїнській науково-практичній конференції «Theoretical and Applied Cybersecurity» (TACS-2023) на базі Фізико-технічного інституту Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського”.

1 ПОНЯТТЯ VEC та BEC, ПОПЕРЕДНЯ ОБРОБКА ДАНИХ, МЕТОДИ МАШИННОГО НАВЧАННЯ, МЕТРИКИ ДЛЯ ОЦІНКИ ЕФЕКТИВНОСТІ МОДЕЛЕЙ

1.1 VEC та BEC

BEC (business email compromise) є одним з найпоширеніших та найнебезпечніших видів кібершахрайства, який полягає у використанні електронної пошти для обману організацій або осіб на переказ коштів або надання конфіденційної інформації [1]. BEC може мати різні форми, такі як підробка адреси електронної пошти керівника, постачальника або партнера, запит на зміну банківських реквізитів, фальшивий рахунок або прохання про допомогу [1]. BEC може призвести до великих фінансових та репутаційних збитків для жертв.

Vendor email compromise (VEC) є видом атаки на ланцюг постачання (або шахрайства на ланцюгу постачання), яка полягає у тому, що кіберзловмисник отримує доступ до електронної пошти постачальника або партнера, який має довіру в організації-цілі, і використовує її для вимагання платежу за фальшивий рахунок або переказу коштів на шахрайський рахунок [2]. VEC є більш складним і цільовим, ніж звичайні атаки business email compromise (BEC), які не обов’язково мають бути адаптовані до конкретної особи. VEC вимагає високого рівня дослідження, імітації та переконання з боку зловмисників, щоб обдурити жертву.

В більшості випадків подібні листи відповідають певному шаблону: терміновість повідомлення, адреса відправника не відповідає його імені, неможливість особистої розмови (лише текстова переписка), прохання змінити платіжні дані або надіслати якусь доволі важливу інформацію (Рисунок 1.1).

Існує декілька підходів для боротьби з цим видом фішингу:

● **Використання чорних списків:** електронні адреси відомих зловмисників блокуються та нові листи від них не надходять. [3]

● **Червоні прапорці:** вміст листа аналізується та шукаються характерні для BEC/VEC слова та фрази, типовими для цього виду фішингу є: “DD update”, “update banking information”, “CEO”, “gift cards”, тощо.

● **Поведінкові характеристики:** для виявлення шкідливих листів аналізуються заголовки листа. Наприклад: час надходження листа, IP-адреса відправника, сервер відправника, репутація адреси та домену відправника, схожість імені або адреси відправника з тими, що є в компанії, тощо. [3]

● **Методи машинного навчання:** виділяються своєю зручністю, так як враховуючи еволюційність підходів зловмисника дуже ймовірно, що не буде необхідності оновлювати систему (на відміну від інших методів), буде достатньо лише додатково навчити модель на нових даних.

● **Навчання співробітників:** у випадку, коли всі попередні методи не спрацювали – ціль зловмисника стає останньою лінією оборони. На такий випадок компанії проводять тренінги своїх робітників. В залежності від підходу та побажань організатора вони можуть проходити в різних форматах: несправжня фішингова розсилка, семінари, онлайн-курси, тощо. Зазвичай в постачальника послуг безпеки електронних поштових скриньок є опція проведення подібних тренувань. [3]

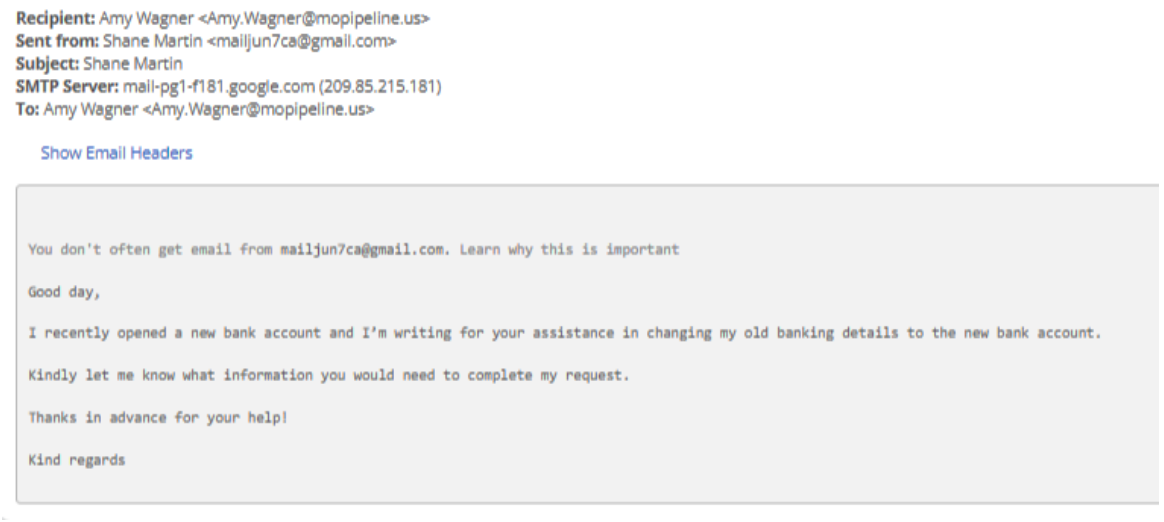

Рисунок 1.1 – На зображенні видно, що зловмисник видає себе за Shane Martin, електронна адреса відправника не відповідає імені відправника (швидше за все це випадковий набір літер і співпадіння перших двох з іменем – це збіг), домен відправника “google.com”, що також є певним індикатором і доволі звичним явищем, зловмисник каже, що змінив банківські дані і хоче, аби співробітник відділу бухгалтерії оновив інформацію в базі.

1.2 Попередня обробка даних

Попередня обробка тексту – це процес, який передбачає перетворення неструктурованого тексту в єдину структуровану форму, що підходить для аналізу та вивчення. Так як більшість текстів пише людина, то не виключено, що в ньому будуть зайві пробіли, знаки пунктуації, літери регістру, що не підходить до речення, тощо.

Попередня обробка тексту допомагає:

- **Зменшити шум і надлишковість даних**. Текстові дані часто містять інформацію, яка не є корисною для аналізу, наприклад, знаки пунктуації, стоп-слова, надлишкові пробіли, тощо. Попередня обробка дозволяє видалити та/або замінити таку інформацію і тим самим зменшити масив даних або його складність. [4]
- **Покращити якість та швидкодію алгоритмів машинного навчання.** Текстові дані майже завжди мають неоднорідний формат, наприклад,

стиль написання, регістри, форми слів, тощо. Попередня обробка дозволяє привести слова до одного формату, видалити все зайве та спростити тест і таким чином буде підвищена швидкість та точність алгоритмів машинного навчання. [4]

● **Видобути корисну інформацію та контекст з текстових даних.** Текстові дані зазвичай містять неявну або не очевидну інформацію, наприклад, семантику, контекст, емоції, тощо. Попередня обробка дозволяє перетворити інформацію в числовий вектор, і таким чином зв’язати слова між собою (відобразити особливості тексту в числах). [4]

1.2.1 Приведення тексту до нижнього регістру, видалення пунктуації та зайвих пробілів.

Приведення тексту до нижнього регістру як і видалення пунктуації та зайвих пробілів, напевно, найлегший спосіб спростити живий текст для обробки, проте, теж мають вплив на ефективність деяких методів машинного навчання. В даній роботі для зведення тексту до нижнього регістру використовується стандартна функція Python “lower()”, а для видалення пунктуації та зайвих пробілів використовуються прості регулярні вирази: `text = re.sub(r'[^\w\s]', '', text)` — для видалення всього, що не є буквами чи цифрами; `text = re.sub(r'\s+', ' ', text)`— для заміни всіх послідовностей пробілів на один пробіл.

Таблиця 1.1 – Порівняння сирого тексту та з приведеним регістром до нижнього, видаленими зайвими пробілами та символами, що не є літерами або цифрами

До	Після
[You don't often get email from brecam495@gmail.com. Learn why this is important at https://aka.ms/LearnAboutSenderIdentification] CAUTION:This email	you dont often get email from brecam495gmailcom learn why this is important at httpsakamslearnaboutsenderidentification cautionthis email originated from outside of the organisation do not click

Кінець таблиці 1.1

originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and know the content is safe. -- Hello Daniel Hope you don't have so much on your plate? Well in case you do, kindly peg it.I want you to handle a confidential task for me quickly as we wish to implement an Employee incentive program. Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you details. Best Regards, Adam H. Couch Sent from my iPad	links or open attachments unless you recognise the sender and know the content is safe hello daniel hope you dont have so much on your plate well i case you do kindly peg iti want you to handle a confidential task for me quickl as we wish to implement an employee incentive program let me know if you are available and also reconfirm with your whatsapp number here so that i ca send you details best regards adam h couch sent from my ipad

1.2.2 Стемінг

Стемінг – це метод попередньої обробки тексту, що заключається у видаленні суфіксів та закінчень в слова, результат часто схожий на зведення до кореня слова, але зазвичай це збіг, так як алгоритм базується на інших принципах. Існує декілька варіантів цього алгоритму, що відрізняються швидкістю та точністю:

- **Пошук за таблицею.** В цьому алгоритмі використовується таблиця усіх можливих варіантів слів та їх зведених форм. Такий метод працює доволі швидко відносно інших, але його можливості доволі обмежені через його специфіку (таблиця має постійно оновлюватись, аби підтримувати максимальну кількість слів).

● **Портерівський алгоритм** (його покращена версія застосовується в роботі). Даний алгоритм використовує набір правил, що послідовно зводять слово до мінімальної форми. Перевагами є його простота, швидкість роботи та універсальність, проте він має недоліки, так як через відсутність врахування структури слова має потенційну низьку точність та високу ймовірність помилок.

● **Алгоритм Ловінса**. Даний алгоритм користується списком з 294 суфіксів для їхнього видалення. Перевагами, є швидкість та простота, а недоліками, як і в портерівського, є низька точність та висока ймовірність помилок через відсутність врахування морфологічної структури тексту та крім цього він не може обробляти слова з префіксами.

В даній роботі застосовується алгоритм **Snowball**, що був розроблений Мартіном Протером і по факту є покращеною версією портерівського алгоритму. Різниця між ними в тому, що Snowball швидший, точніший та швидший за портерівський, проте, має ті самі недоліки, що й оригінал (хоча й в меншій мірі).

Таблиця 1.2 – Порівняння сирого тексту та після стемінгу

До	Після
[You don't often get email from brecam495@gmail.com. Learn why this is important at https://aka.ms/LearnAboutSenderIdentification] CAUTION:This email originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and know the content is safe. -- Hello Daniel Hope you don't have so much on your plate? Well in case you do, kindly	you do n't often get email from brecam495 @ gmail.com . learn whi this is import at https : //aka.ms/learnaboutsenderidentif] caution : this email origin from outsid of the organis . do not click link or open attach unless you recognis the sender and know the content is safe . -- hello daniel hope you do n't have so much on your plate ? well in case you do , kind peg it.i want you to handl a confidenti task for me quick as we wish to implement an employe incent program . let me know if you are avail and also re-confirm with your whatsapp number here so that i can send you detail . best regard , adam h.

Кінець таблиці 1.2

peg it.I want you to handle a confidential task for me quickly as we wish to implement an Employee incentive program. Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you details. Best Regards, Adam H. Couch Sent from my iPad	couch sent from my ipad Backend TkAgg is interactive backend. Turning interactive mode on.

1.2.3 Лематизація

Лематизація - це алгоритмічний процес встановлення леми слова на основі його очікуваного значення. На відміну від стематизації, лематизація залежить від коректного визначення частини мови та значення слова в реченні, а також в ширшому контексті, який оточує це речення. Існує декілька варіантів алгоритму лематизації:

- **Пошук за таблицею** - це алгоритм лематизації, який використовує принцип пошуку за таблицею, в якій збережені всі можливі форми слова та їх леми. Перевагами цього методу є простота, швидкість та зручність обробки винятків з мовних правил. До недоліків слід віднести великий обсяг таблиці, яка має бути попередньо створена та оновлювана, а також низьку гнучкість та адаптивність до нових слів.
- **Правила заміни**. Цей алгоритм використовує принцип заміни частин слова за допомогою набору правил, що по черзі застосовуються до слова для його зведення. Перевагами цього методу є простота, швидкість та універсальність, а недоліками є низька точність та висока ймовірність помилок, оскільки він не враховує морфологічну структуру слова та контекст.
- **Морфологічний аналіз**. Цей алгоритм використовує принцип розбору слова на морфеми - найменші значущі частини слова - та визначення їх

ролей та значень. Перевагами цього методу є висока точність та здатність враховувати морфологічну структуру слова та контекст. До недоліків слід віднести складність, повільність та необхідність наявності морфолог.

В роботі використовується лематизатор WordNetLemmatizer з бібліотеки nltk, що використовує морфологічну інформацію з WordNet для зведення слів до базової форми.

Таблиця 1.3 – Порівняння сирого тексту та після лематизації

До	Після
[You don't often get email from brecam495@gmail.com. Learn why this i important at https://aka.ms/LearnAboutSenderIdentification] CAUTION:This email originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and know the conter is safe. -- Hello Daniel Hope you don't have so much on your plate? Well in case you do, kindly peg it.I want you to handle a confidential task for me quickly as we wish to implement an Employee incentiv program. Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you details. Best Regards, Adam H. Couch Sent from my iPad	You do n't often get email from brecam495 @ gmail.com . Learn why thi is important at http : //aka.ms/LearnAboutSenderIdentification] CAUTION : This email originated from outside of the organisation . Do not click link or open attachment unless you recognise the sender and know the conter is safe . -- Hello Daniel Hope you do n't have so much on your plate ? Well in cas you do , kindly peg it.I want you to handl a confidential task for me quickly a we wish to implement an Employee incentiv program . Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you detail . Best Regards , Adam H. Couch Sent from my iPad Backend TkAgg is interactive backend. Turning interactive mode on.

1.2.4 Видалення стоп-слів

Стоп-слова - це фрази, слова або словоформи, які не несуть ніякого смислового навантаження і можуть бути видалені з тексту (для англійської мови прикладами можуть бути: “the”, “and”, “a”, “an”, тощо.) Таким чином розмір

тексту зменшується і підвищується простота аналізу та його швидкість. Існує кілька алгоритмів для їх видалення:

- **Алгоритм Робертса–** використовується для видалення стоп-слів з англомовного тексту. Використовує заздалегідь визначений список для виявлення цілей.
- **Алгоритм Кунта і Макклелланда –** використовується для видалення стоп-слів з німецькомовного тексту. Використовує частотний аналіз для виявлення цілей.
- **Алгоритм Хантера** – використовується для видалення стоп-слів з французьких текстів. Використовує частотний аналіз для виявлення цілей.

Як можна побачити, алгоритми працюють дуже схоже і найчастіше відрізняються цільовими мовами. В даній роботі використовується метод з використанням бібліотеки англомовних стоп-слів, після чого вони просто видаляються з тексту.

Таблиця 1.4 – Порівняння сирого тексту та після видалення стоп-слів

До	Після
[You don't often get email from brecam495@gmail.com. Learn why this is important at https://aka.ms/LearnAboutSenderIdentification] CAUTION:This email originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and know the content is safe. -- Hello Daniel	You n't often get email brecam495 @ gmail.com . Learn important https : //aka.ms/LearnAboutSenderIdentification] CAUTION : This email originated outside organisation . Do click links open attachments unless recognise sender know content safe . -- Hello Daniel Hope n't much plate ? Well case , kindly peg it.I want handle confidential task quickly wish implement Employee incentive

Закінчення таблиці 1.4

Hope you don't have so much on your plate? Well in case you do, kindly peg it.I want you to handle a confidential task for me quickly as we wish to implement an Employee incentive program. Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you details. Best Regards, Adam H. Couch Sent from my iPad	program . Let know available also re-confirm WhatsApp number send details . Best Regards , Adam H. Couch Sent iPad Backend TkAgg is interactive backend. Turning interactive mode on.

1.2.5 CountVectorizer та OneHotEncoder

CountVectorizer - це клас з бібліотеки scikit-learn, який використовується для перетворення тексту у вектори лічильників токенів [5]. Це означає, що він розбиває текст на окремі слова (токени) та підраховує кількість входжень кожного слова у текст. Результатом є матриця, де кожен рядок представляє документ, а кожен стовпець представляє унікальне слово у корпусі.

OneHotEncoder - це клас з бібліотеки scikit-learn, який використовується для кодування категоріальних змінних у форматі "one-hot" [6]. Це означає, що для кожної категорії створюється новий бінарний стовпець, де значення 1 означає наявність категорії, а 0 - її відсутність. Наприклад, якщо ми маємо категоріальну змінну “колір” з можливими значеннями “червоний”, “зелений” та “синій”, то OneHotEncoder створить три нових стовпця: “колір_червоний”, “колір_зелений” та “колір_синій”.

Варто зазначити, що методи машинного навчання не можуть працювати з абсолютно сирими даними, тому в розділі з аналізом впливу попередньої обробки ці методи будуть присутні в усіх дослідженнях за замовчуванням.

В даному розділі будуть описані використані в роботі методи машинного навчання, їхні переваги та недоліки.

1.3 Методи машинного навчання

1.3.1 Багатошаровий перцептрон

Multi Layer Perceptron Classifier (MLPClassifier) – це класифікатор, що використовує нейромережу багатошарового перцептрона (MLP), який навчається за алгоритмом зворотного поширення оптимізуючі функцію лог-втрат за допомогою LBFGS (Limited-memory алгоритм Бройдена-Флетчера-Гольдфарба-Шанно) або стохастичного градієнтного спуску. Він є методом машинного навчання з учителем, що в тандемі з попередньою обробкою тексту добре підходить для класифікації тексту.

1.3.2 Логістична регресія

Логістична регресія – це статистичний регресійний метод, що використовується, коли залежна змінна є бінарною (тобто приймає значення 0 або 1).

Однією з переваг цього метода машинного навчання є те, що вона проста для застосування та інтерпретації. Коефіцієнти логістичної регресії можна використати для оцінки впливу незалежних змінних на ймовірність належності до певної категорії. А також даний метод можна застосувати до даних з категоріальними або числовими незалежними змінними.

Недоліком логістичної регресії є те, що вона припускає лінійний зв’язок між незалежними змінними та log-odds (логарифм шансу) належності до певної категорії. Якщо така залежність порушена, то результати моделі можуть бути помилковими.

1.3.3 Випадковий ліс

Випадковий ліс - це ансамблевий метод, який використовується для класифікації, регресії та інших завдань. Він працює шляхом побудови багатьох дерев рішень під час тренування та об’єднання їх результатів. Ого особливість полягає в тому, що він використовує багато дерев рішень, після чого знаходить середнє значення їхніх прогнозів.

Перевагами випадкового лісу є здатність ефективно обробляти великі набори даних а також він менш схильний до перенавчання, ніж окреме дерево рішень,.

Однак, випадковий ліс також має деякі недоліки. Наприклад, моделі можуть бути складними та потребувати багато ресурсів. Також, прогнози, отримані цим методом, можуть бути менш інтерпретованими, ніж прогнози, отримані за допомогою окремого дерева рішень.

1.3.4 k-найближчих сусідів

Алгоритм k-найближчих сусідів (KNN) - це непараметричний метод навчання з учителем, який використовується для класифікації та регресії. Він полягає в знаходженні k найближчих сусідів до нового спостереження та використання їхнього середнього значення або значення класу для встановлення значення нового спостереження або прогнозування класу.

KNN доволі простий для розуміння та реалізації і може бути ефективним, коли межа рішення є складною, проте цей алгоритм дуже повільно працює на великих даних.

1.3.5 Метод опорних векторів

Метод опорних векторів (SVM) - це метод аналізу даних для класифікації та регресійного аналізу, що використовує кероване навчання. SVM знаходить гіперплощину у високовимірному просторі, яка найкраще розділяє дані на класи.

Однією з переваг SVM є те, що вони ефективні у випадках, коли кількість змінних більша, ніж кількість спостережень. SVM також мають гнучкість завдяки використанню різних ядерних функцій, що дозволяє їм моделювати нелінійні межі рішень.

Однак, SVM також мають деякі недоліки. Наприклад, вони доволі чутливі до вибору ядерної функції та її параметрів. Крім того, SVM можуть бути складними для інтерпретації та потребують налаштування регуляризаційного параметру.

1.4 Метрики для оцінки точності моделей машинного навчання

● **Точність (Accuracy)** – відношення кількості правильно класифікованих спостережень до суми всіх спостережень.

$$\text{Accuracy} = \frac{TP + TN}{TP+TN+FP+FN} \quad (1.1)$$

● **Точність (Precision)** – відношення кількості істинно позитивних спостережень до суми істинно позитивних та хибно позитивних спостережень. Тобто, це частина вірно класифікованих позитивних спостережень відносно всіх класифікованих позитивних спостережень.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (1.2)$$

● **Повнота (Recall)** – відношення кількості істинно позитивних спостережень до суми істинно позитивних та хибно негативних спостережень. Тобто, це частина вірно класифікованих позитивних спостережень відносно всіх, котрі мали бути позитивно класифіковані.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (1.3)$$

● **F1-оцінка (F1-score)** – це гармонійне середнє між точністю (Precision) та повнотою. Необхідний для оцінки балансу між точністю (Precision) та повнотою.

$$\text{F1-score} = \frac{2 * (Precision \times Recall)}{Precision + Recall} \quad (1.4)$$

Також, крім класичної оцінки будуть розглянуті крос-валідаційні метрики. Ці метрики використовуються для оцінки якості моделей машинного навчання за допомогою крос-валідації. При такому підході набір даних розділяється на n підмножин (“склади”), після чого модель навчається на n-1 складах і тестується на останньому. Цей процес повторюється до тих пір, доки кожен склад не буде використано як тестовий рівно один раз. В цьому методі оцінки застосовуються ті самі формули, просто вони вираховуються для кожного з k тестів та усереднюється. Даний підхід дозволяє більш точно оцінити ефективність моделі та зменшити вплив похибки на результати. Але так як тренувальні і тестові дані розділені в співвідношенні 3:2, то класичний метод демонструє роботу на більшій вибірці.

Висновки до розділу 1

Отже, компрометація електронних скриньок дуже небезпечний вид фішингу так як використовує методи соціальної інженерії (терміновість, вигідність для отримувача, тощо). Не готовий до такої атаки співробітник може з легкістю пропустити індикатори шкідливості листа та послідувати вказівкам зловмисника.

Існує багато методів для виявлення листів типу BEC/VEC, проте жоден з них не може дати стовідсоткової гарантії виявлення. Тому задачею цієї роботи буде зведення ймовірності помилки до мінімуму. Звісно, в реальних умовах не застосовується лише один метод і, наприклад, тренування співробітників відбувається навіть тоді, коли компанія застосовує автоматизовані рішення для виявлення небажаної пошти.

Попередня обробка тексту є необхідним етапом в оптимізації даних перед навчанням моделей машинного навчання. Існує багато різних підходів, але усі вони націлені на скорочення вхідного тексту та зменшення його різноманітності.

В розділі з аналізом буде окремо досліджено вплив кожного з обраних та всіх разом методів попередньої обробки тексту на результативність обраних методів машинного навчання. На даному етапі я можу припустити, що послідовне примінення усіх методів дасть найкращий результат.

Таблиця 1.5 – Порівняння сирого тексту та після застосування усіх методів попередньої обробки тексту

До	Після
[You don't often get email from brecam495@gmail.com. Learn why this is important at https://aka.ms/LearnAboutSenderIdentification] CAUTION:This email originated from outside of the organisation. Do not click links or open attachments unless you recognise the sender and know the content is safe. -- Hello Daniel Hope you don't have so much on your plate? Well in case you do, kindly peg it.I want you to handle a confidential task for me quickly as we wish to implement an Employee incentive program. Let me know if you are available and also re-confirm with your WhatsApp number here so that i can send you details. Best Regards, Adam H. Couch Sent from my iPad	dont often get email brecam495gmailcom learn whi import httpsakamslearnaboutsenderidentif cautionthi email origin outsid organis click link open attach unless recognis sender know content safe hello daniel hope dont much plate well case kind peg iti want handl confidenti task quick wish implement employe incent program let know avail also reconfirm whatsapp number send detail best regard adam h couch sent ipad Backend TkAgg is interactive backend. Turning interactive mode on.

Щодо обраних методів машинного навчання – потенційно всі з обраних методів машинного навчання добре підходять для класифікації мого набору даних, проте результат сильно залежить від специфіки самого набору, тому передбачити результати без практичної перевірки доволі складно. Пізніше буде продемонстровано, що багатошаровий перцептрон покаже найкращий результат, так як здатен до “людиноподібного” аналізу тексту, тобто зможе виявити певні залежності в тексті. На жаль, через особливості роботи цього методу неможливо виявити, який саме фрагмент тексту спровокував правильну класифікацію. Максимум, який можна отримати – найбільш вагомі окремі слова (Додаток Б).

Для оцінки ефективності різних методів попередньої обробки тексту та різних методів попередньої обробки даних будуть застосовані наступні метрики:

- Accuracy
- Precision
- Recall
- F1-score

Дані метрики можуть повною мірою описати результативність окремої моделі та її взаємодію з позитивними та негативними класифікаціями. Для зменшення впливу похибки на результати аналізу будуть застосовані крос-валідаційні варіації цих метрик, що дозволяють на одному набору даних провести декілька тестів з різними тренувальними та тестовими даними.

2 ЗБІР ДАНИХ ДЛЯ ТРЕНУВАННЯ МОДЕЛІ ТА ПІДГОТОВКА НАБОРУ ДАНИХ. АНАЛІЗ ВПЛИВУ МЕТОДІВ ПОПЕРЕДНЬОЇ ОБРОБКИ ТА ВИЗНАЧЕННЯ НАЙБІЛЬШ ЕФЕКТИВНОГО ПІДХОДУ

2.1 Збір даних та підготовка набору даних для тренування та тестування моделі.

В якості шкідливих листів з бази даних компанії було зібрано п’ять сотень листів, що вже були класифіковані комерційною системою як шкідливі з індикатором BEC. Оскільки не існує ідеальних систем, то з’явилась необхідність додатково переглянути усі листи і виправити помилки.

Так як використання реального “чистого” трафіку порушує умови конфіденційності, то для протиставлення шкідливим листам було використано датасет з текстами рекламних розсилок з Kaggle, після чого штучно було його розширено додаванням адреси відправника та адреси отримувача.

Було зібрано близько 500 .json файлів, що містять у собі всі заголовки електронного листа. Для спрощення роботи з ними було написано невеликий застосунок на мові програмування Python (Додаток Г), що робить наступне:

- Знаходить усі файли з розширенням .json у поточній директорії та зберігає їх у списку json_files.
- Створює новий файл output.csv у режимі запису, з кодуванням UTF-8 та без додавання нового рядка після кожного запису.
- Створює об’єкт writer класу csv.writer, що дозволяє записувати дані у файл CSV.
- Записує перший рядок у файл CSV з заголовками стовпців: ‘body’, ‘fullname’ та ‘mail_address’.
- Для кожного файлу JSON у списку json_files:
- Відкриває файл JSON у режимі читання з кодуванням UTF-8.

- Завантажує дані з файлу JSON у змінну data.
- Якщо data є словником, то:
    - Отримує значення для ключа ‘body’ або порожню строку, якщо такого ключа немає.
    - Отримує значення для ключа ‘fullname’ з вкладеного словника за ключем ‘sender_email_address’ або порожню строку, якщо такого ключа немає.
    - Отримує значення для ключа ‘mail_address’ з вкладеного словника за ключем ‘sender_email_address’ або порожню строку, якщо такого ключа немає.
    - Записує отримані значення у наступний рядок файлу CSV.

Звісно, багато листів використовують HTML та CSS у тілі листа, тому було додатково написано функцію для видалення всіх тегів, що належать цим мовам розмітки, для цього достатньо лише за допомогою регулярних виразів видалити усе, що знаходиться поміж трикутних дужок (“<”, “>”).

Для приведення набору з безпечними листами до вигляду набору зі шкідливими було згенеровано ще близько 700 адрес та імен, що були відповідно додані до тіла чистого листа.

Відповідно, для об’єднання двох датасетів, із шкідливих листів було використано тільки текст листа (mail-body), адресу отримувача та адресу відправника. Таким чином фінальний вигляд набору даних можна побачити в Таблиці 2.6, де можна побачити приклад безпечного листа, що в даному випадку містить інформацію з деякого блогу, а також приклад шкідливого листа типу Vendor Email Compromise, де міститься запит на оновлення банківської інформації.

Таблиця 2.6 – демонстрація загального вигляду датасету, з яким працює застосунок

mail-body	user-name	sender-address	label
douglas craig holland (holland@CS.ColoState.EDU) writes: > > With E-Mail, if they can't break your PGP encryption, they'll just > call up one of their TEMPEST trucks and read the electromagnetic > emmisions from your computer or terminal. Note that measures to > protect yourself from TEMPEST surveillance are still classified, as > far as I know. steiner@jupiter.cse.utoledo.edu (Jason 'Think!' Steiner) writes: > are LCD displays vulnerable to tempest? I can see high-voltage type display devices being vulnerable (CRTs, plasma displays, etc.) But Jason beat me to this question. What about EM radiation from low-voltage items like LCD displays? Perhaps the critical element is the driver circuitry? The cabling? What about a portable PC/Mac/etc., where all the "noise" is bunched into one tiny area?	Daniel Farris	info@jp.krannich-solar.com	0
[EXTERNAL EMAIL] This email originated from outside of the organization. DO NOT CLICK links or open attachments unless you recognize the sender and know the content is safe. Hi Deborah, I wish to update my banking information before the next payroll is completed. What information do you need? Regards Noy Vongphachan Treasury Supervisor LURIN	Deborah Jackson	brecam495@gmail.com	1

2.2 Аналіз впливу методів попередньої обробки

В даному розділі буде розглянуто та досліджено вплив окремих (і всіх разом) методів попередньої обробки тексту на ефективність усіх обраних

методів машинного навчання. Ефективність буде оцінена за наступними метриками (буде використано поняття: TP - істинно позитивні, TN - істинно негативні, FP - хибно позитивні, FN - хибно негативні):

2.2.1 Аналіз без попередньої обробки

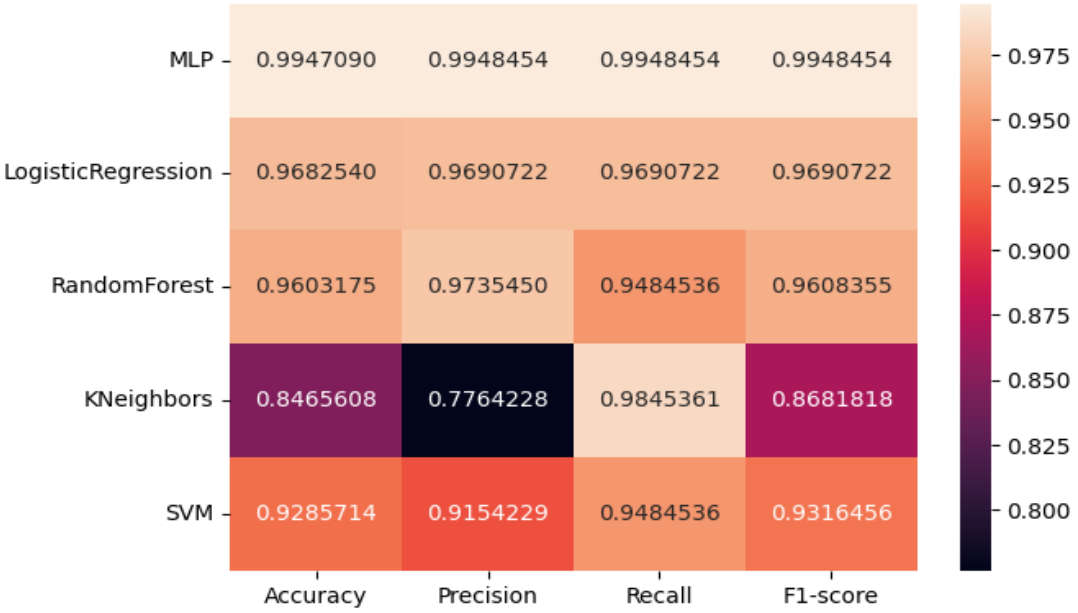

Рисунок 2.1 – Класичні метрики без попередньої обробки

На даному етапі ще зарано робити висновки, так як його ми будемо вважати та точку відліку, але відповідно до рисунка 2.1 та рисунка 2.2 на даний момент можна сказати, що багатошаровий перцептрон показує дуже добрі результати і випереджає інші моделі. Враховуючи доволі високий показник повноти в k-найближчих сусідів можна сказати, що він доволі непогано класифікує шкідливі листи, але в той же час робить багато помилок на чистих. Тобто, ця модель робить забагато позитивних класифікацій.

Рисунок 2.2 – Крос-валідаційні метрики без попередньої обробки

2.2.2 Аналіз зведення регістрів, видалення зайвих символів та пробілів

Відповідно до порівняння Рисунка 2.4 та Рисунка 2.2 можемо зробити висновки:

- MLP: всі характеристики трохи (порядка десятих відсотка) покращились, ймовірно це погрішність.
- Логістична регресія: схоже, кількість TN класифікацій підвищилась, а TP зменшилась.
- Випадковий ліс: усі метрики погіршились на 1-2 відсотки.
- k найближчих сусідів: усі метрики покращились в середньому на 1.5 відсотків.
- Опорні вектори: усі метрики погіршились на 3-4 відсотки.

Рисунок 2.3 – Класичні метрики зі зведенням регістрів, видаленням зайвих символів та пробілів

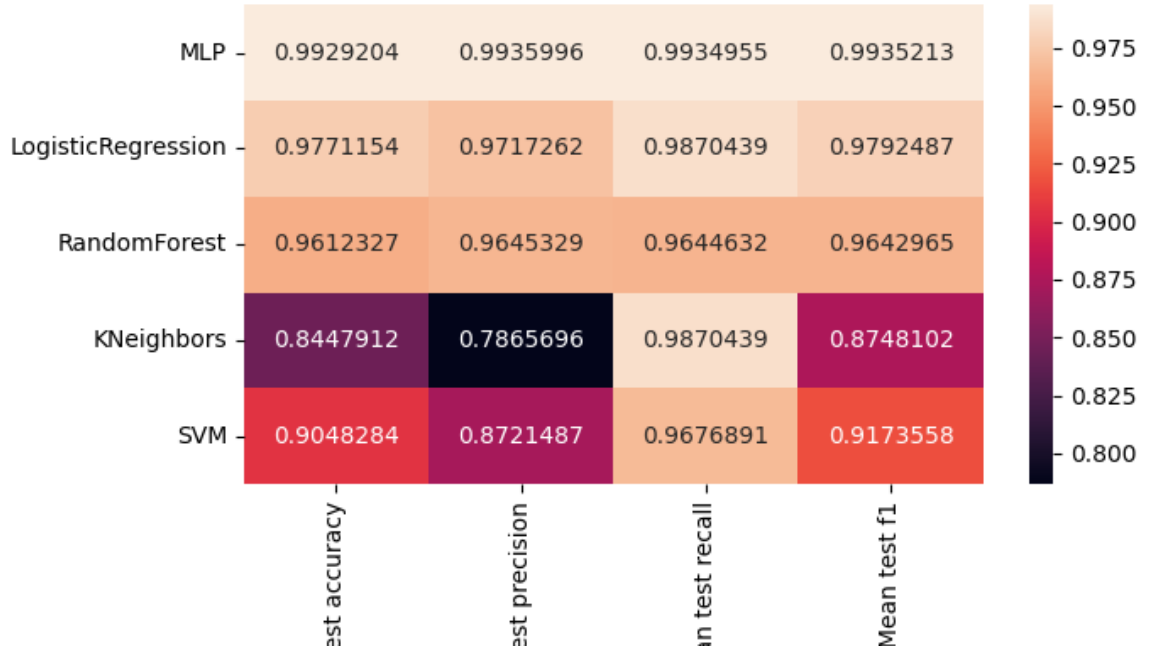

Рисунок 2.4 – Крос-валідаційні метрики зі зведенням регістрів, видаленням зайвих символів та пробілів

2.2.3 Аналіз стематизації

Відповідно до порівняння рисунка 2.6 та Рисунка 2.2 можемо зробити висновки:

- MLP: більшість метрик (крім повноти) покращились на 0.1-0.3 відсотки (ймовірно, теж похибка).
- Логістична регресія: усі метрики погіршились на 0.5-1 відсоток.
- Випадковий ліс: враховуючи погіршення значення повноти на 2 відсотки можемо сказати, що модель стала робити значно менше успішних TP класифікацій, але, враховуючи незначне падіння решти метрик, TN класифікації не постраждали.
- k найближчих сусідів: усі метрики крім повноти значно збільшились, але повнота зменшилась – отже, модель стала менш агресивно ставити позитивні класифікації.
- Опорні вектори: всі метрики погіршились на 1-3 відсотки.

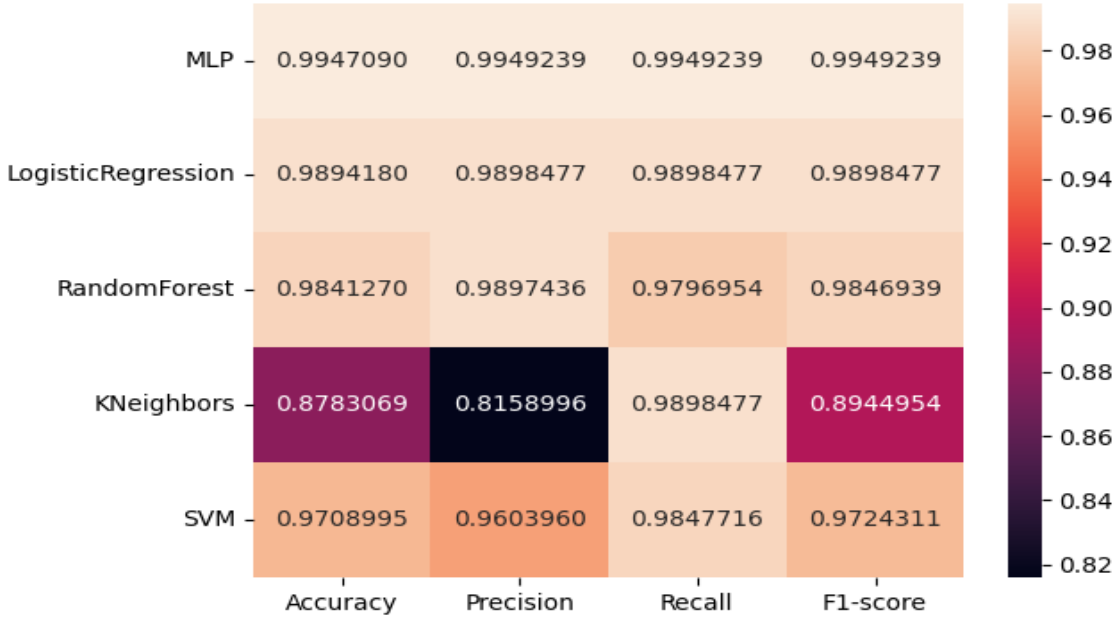

Рисунок 2.5– класичні метрики з використанням стематизації

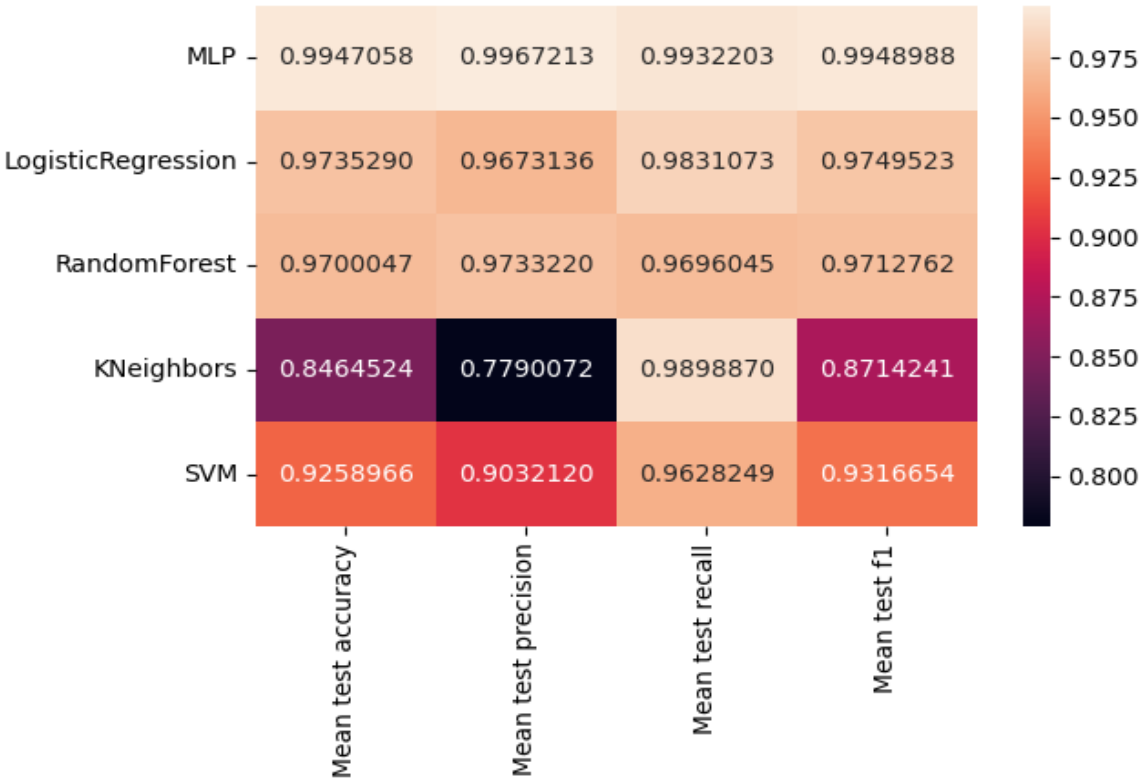

Рисунок 2.6 – крос-валідаційні метрики з використанням стематизації

2.2.4 Аналіз лематизації

Відповідно до порівняння Рисунка 2.8 та Рисунка 2.2 можемо зробити висновки:

- MLP: усі метрики погіршились на 0.01-0.3 відсотки (ймовірно, погрішність).
- Логістична регресія: accuracy погіршилась на 0.5 відсотків, решта змінилась незначно. Отже, модель стала гірше класифікувати TN.
- Випадковий ліс: precision зменшилась на 1 відсоток, отже модель стала гірше класифікувати TP.
- k найближчих сусідів: усі метрики погіршились на 0.5-2 відсотки.
- Опорні вектори: всі метрики погіршились на 2.5-3 відсотки

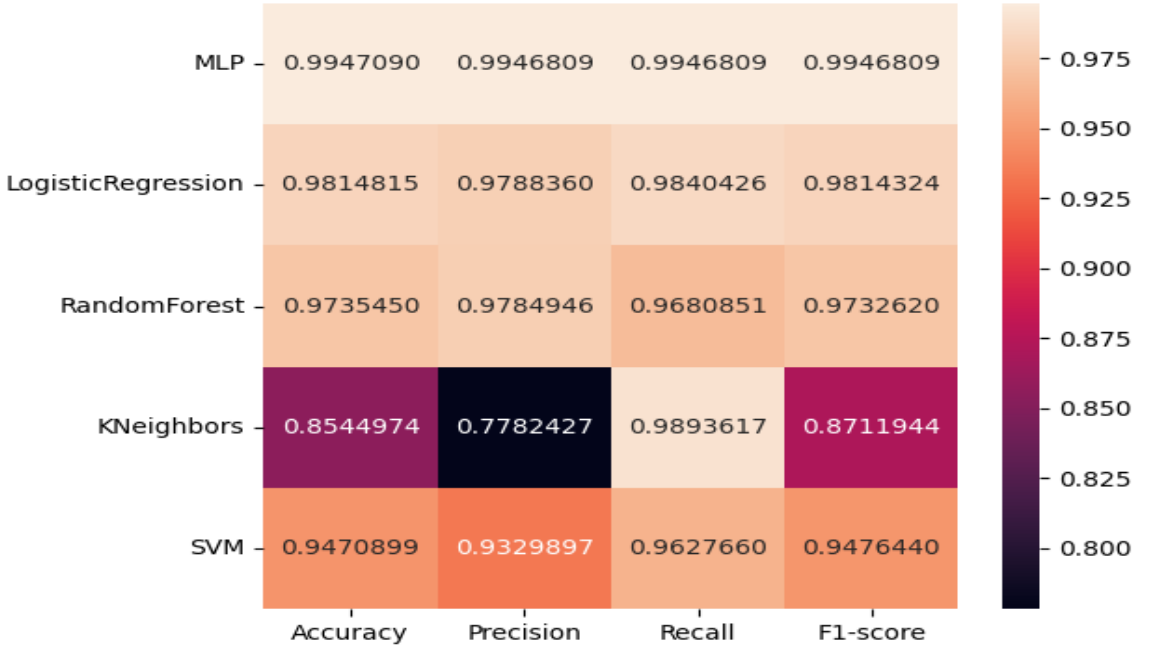

Рисунок 2.7 – класичні метрики з використанням лематизації

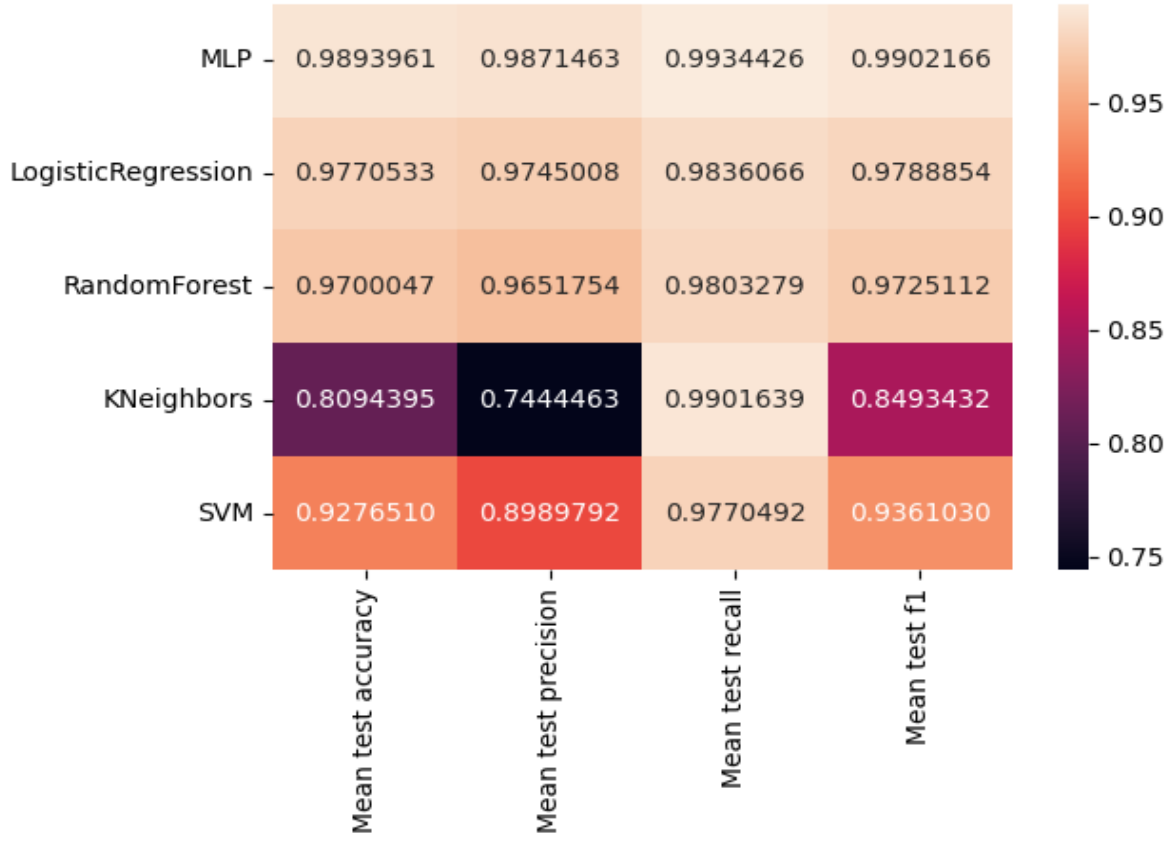

Рисунок 2.8 – крос-валідаційні метрики з використанням лематизації

2.2.5 Аналіз видалення стоп-слів

Відповідно до порівняння Рисунка 2.10 та Рисунка 2.2 можемо зробити висновки:

- MLP: усі метрики крім Precision змінились незначно, але враховуючи зміну Precision можна сказати, що модель стала краще робити TP класифікації, але так, як повнота зменшилась – це покращення відбулося за рахунок зменшення TN.
- Логістична регресія: всі метрики крім Precision змінились незначно (в рамках похибки), але це зменшення на 0.65% може значити погіршення TP класифікацій.
- Випадковий ліс: усі метрики трохи покращились.
- k найближчих сусідів: усі метрики значно погіршились, але повнота дещо збільшилась. Це значить, що модель стала надто агресивно розставляти позитивні реакції
- Опорні вектори: усі метрики крім повноти покращились на 1-2 відсотки, але повнота зменшилась, отже збільшилась кількість FP

Рисунок 2.9 – класичні метрики з використанням видалення стоп-слів

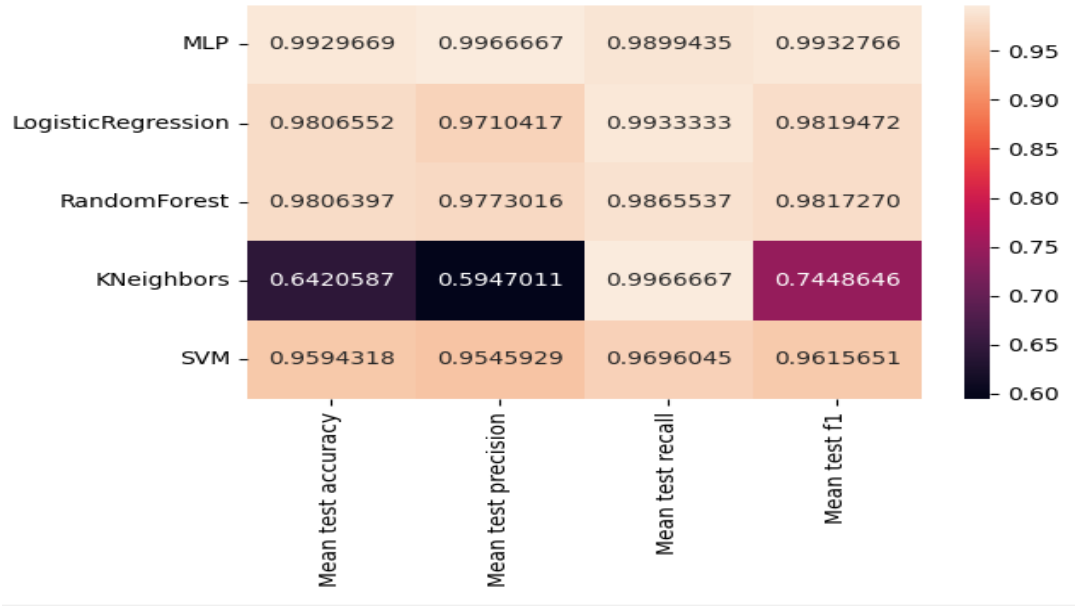

Рисунок 2.10 – крос-валідаційні метрики з використанням видалення стоп-слів

3.2.6 Аналіз усіх методів попередньої обробки

Відповідно до порівняння Рисунка 2.12 та Рисунка 2.2 можемо зробити висновки:

- MLP: незначно вплинуло на результат
- Логістична регресія: всі метрики погіршились на 0.1-0.7 відсотків
- Випадковий ліс: усі метрики трохи покращились (особливо precision)
- k найближчих сусідів: усі метрики крім повноти погіршились – дуже агресивна позитивна класифікація
- Опорні вектори: обидві метрики точності погіршились на 1 відсоток.

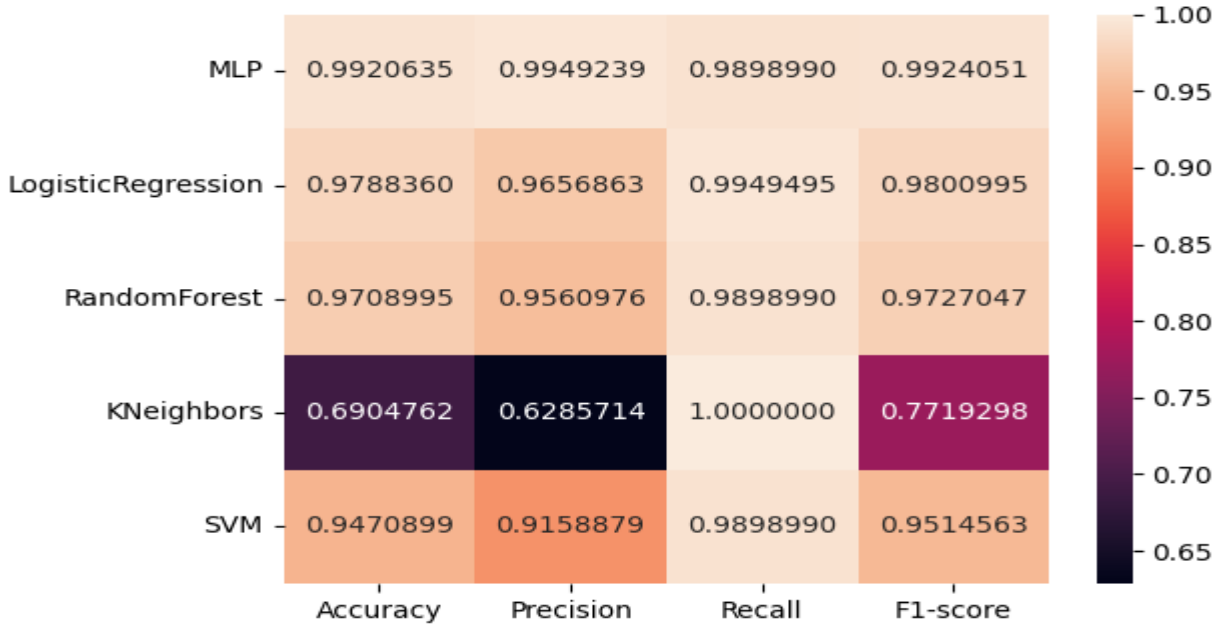

Рисунок 2.11 – класичні метрики з використанням усіх методів попередньої обробки

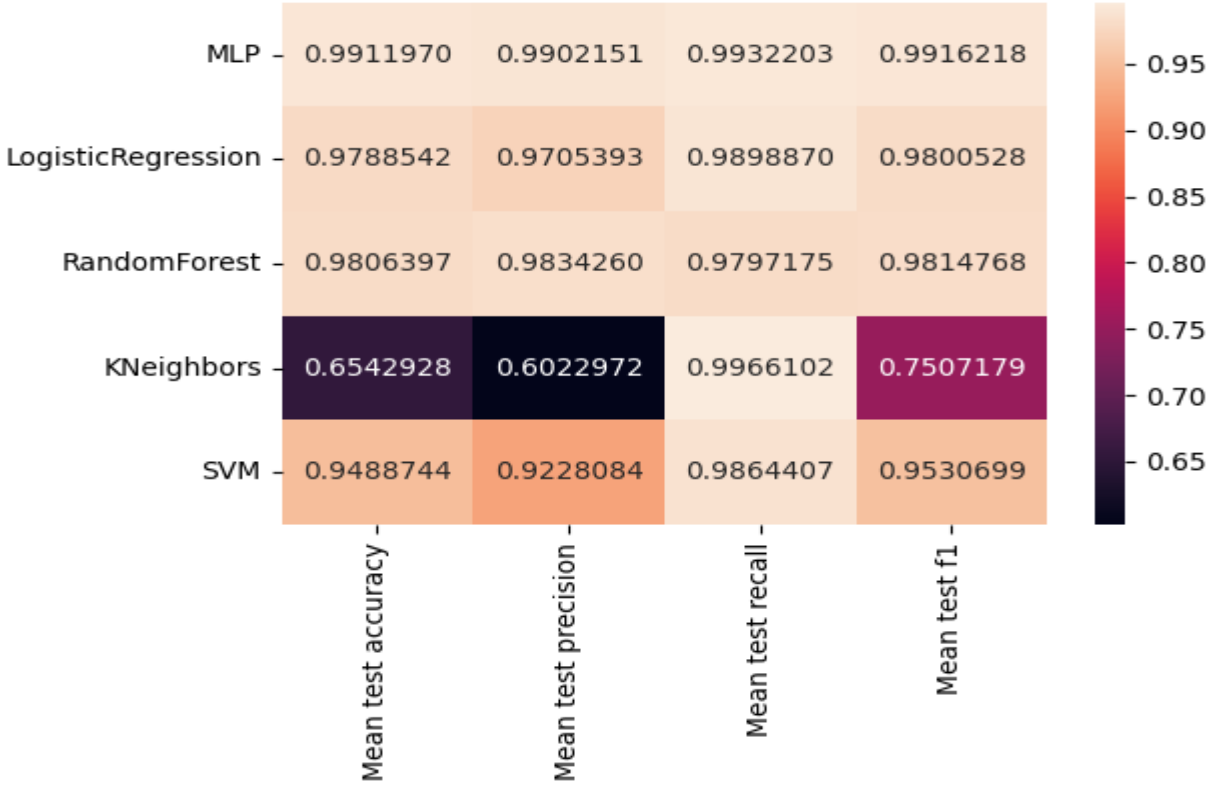

Рисунок 2.12 – крос-валідаційні метрики з використанням усіх методів попередньої обробки тексту

Висновки до розділу 2

На даному етапі було виконано пошук та підготовку даних для їхнього майбутнього використання. Шкідливі листи типу VEC було взято з бази компанії, що займається безпекою електронних скриньок, а чисті було створено самостійно з використанням текстів рекламних розсилок з відкритого доступу. Усі дані було приведено до єдиного вигляду: mail-body, user-name, sender-address, label.

- mail-body – текст електронного листа
- user-name – ім’я отримувача
- sender-address – адреса відправника
- label – класифікація листа (0 – чисте, 1 – шкідливе)

Таким чином, було створено новий набір даних, що складається з шкідливих листів та рекламних розсилок. Це дозволило провести аналіз та порівняння цих двох типів листів. Завдяки цьому було можливо виявити спільні та відмінні характеристики, що допомогли покращити роботу системи класифікації шкідливих листів. Зокрема, було виявлено, що шкідливі листи часто мають певні ключові слова та фрази, що відрізняють їх від рекламних розсилок. Це дало змогу налаштувати систему таким чином, щоб вона краще розпізнавала шкідливі листи.

Провівши дослідження метрик при різних умовах було виявлено, що до попередньої обробки тексту потрібно ставитись з обережністю. Підхід напряму залежить від моделі, до якої він застосовується та масиву даних, що використовуються. Багато комбінацій показали негативний результат, а деякі майже не змінили загальної картини.

3 ПОКРАЩЕННЯ РОБОТИ МОДЕЛІ

Тож, якщо ми виявили, що найкращим варіантом для класифікації VEC листів серед обраних є багатошаровий перцептрон, то як його можна покращити? Позитивний результат може дати розширення набору даних, як вздовж (збільшення кількості записів) так і вшир (збільшення кількості полів для аналізу). Але якщо в нас немає можливості зібрати більше даних, то можемо спробувати довчити модель на вже наявних даних (звісно, в даному контексті вважається, що ми знаходимось в умовах, коли модель вже працює і ми можемо використовувати лише ті листи, що приходять на даний момент).

3.1 Визначення найбільш ефективного підходу

Провівши це дослідження ми побачили, що дуже не часто попередня обробка покращує оцінку моделі, а застосування усіх одразу покращило тільки випадковий ліс. Також видно, що на багатошаровий перцептрон попередня обробка майже не впливає.

Чи має сенс застосовувати попередню обробку? В моїх результатах видно, що це залежить від декількох факторів: модель, котра використовується; цілі, котрі переслідують (наприклад, якщо користувачу все одно на FP, тоді можна використати усі методи попередньої обробки та метод k-найближчих сусідів, так як він демонструє агресивну політику відносно позитивних класифікацій); варто зазначити, що я не акцентував увагу на загальній картині, котра виникає, але, наприклад, при використанні видалення стоп-слів більшість моделей показали доволі високий та стабільний результат (стабільний в плані співвідношення).

Також дуже важлива метрика F1-оцінки, котра демонструє збалансованість моделі. В такому контексті багатошаровий перцептрон при стематизації показав найкращий результат.

Отже, найкращі результати показала модель багатошарового перцептрона при використанні тільки стематизації (вплив цього методу незначний, проте саме він буде застосовуватись на наступних етапах). Також непоганий результат показав випадковий ліс з усіма попередніми обробками, але гірший за MLP.

3.2 Оптимізація багатошарового перцептрону

Можна також застосувати популярні методи оптимізації багатошарового перцептрону, наприклад:

- Adam - це оптимізатор, що базується на стохастичному градієнтному спуску, який працює добре для великих наборів даних та глибоких архітектур. Він поєднує найкращі властивості алгоритмів AdaGrad та RMSProp, щоб забезпечити оптимізатор, який може обробляти розріджені градієнти на шумних задачах [7].
- L-BFGS (Limited-memory BFGS) - це оптимізатор у стилі квазі-Ньютоновських методів, який апроксимує алгоритм Бройдена-Флетчера-Гольдфарба-Шанно (BFGS), використовуючи обмежену кількість комп’ютерної пам’яті. Це популярний алгоритм для оцінки параметрів у машинному навчанні [8]. Він працює добре для невеликих наборів даних та забезпечує хорошу точність оцінки градієнтів. Оскільки L-BFGS зберігає лише кілька векторів, які неявно представляють апроксимацію, вимоги до пам’яті лінійні, що робить його особливо підходящим для оптимізаційних задач з багатьма змінними.

В якості налаштувань для цих оптимізаторів (гіперпараметри) були обрані:

- adam:
  - lr (швидкість навчання): це крок оптимізації, який визначає, наскільки швидко модель змінює свої параметри під час тренування.

- betas (моменти): це два гіперпараметри, які контролюють експоненційне згладжування градієнтів та квадратних градієнтів.
- eps (стабілізуючий параметр): це дуже мале число, яке додається до знаменника при оновленні параметрів для уникнення ділення на нуль. [7]

- L-BFGS:
  - max_iter (максимальна кількість ітерацій): це максимальна кількість ітерацій оптимізатора.
  - max_eval (максимальна кількість оцінок): це максимальна кількість оцінок функції втрат. [7]

Тепер розглянемо декілька варіантів налаштувань цих класифікаторів та оцінемо їхній вплив на результативність моделі:

1) Adam (learning_rate_init=0.01, beta_1=0.9, beta_2=0.999, epsilon=1e-8)

   L-BFGS (max_iter=200, max_fun=15000)

Як бачимо за рисунком 3.1 – з такими параматрами Adam не вплинув на роботу моделі, а L-BFGS погіршив результати.

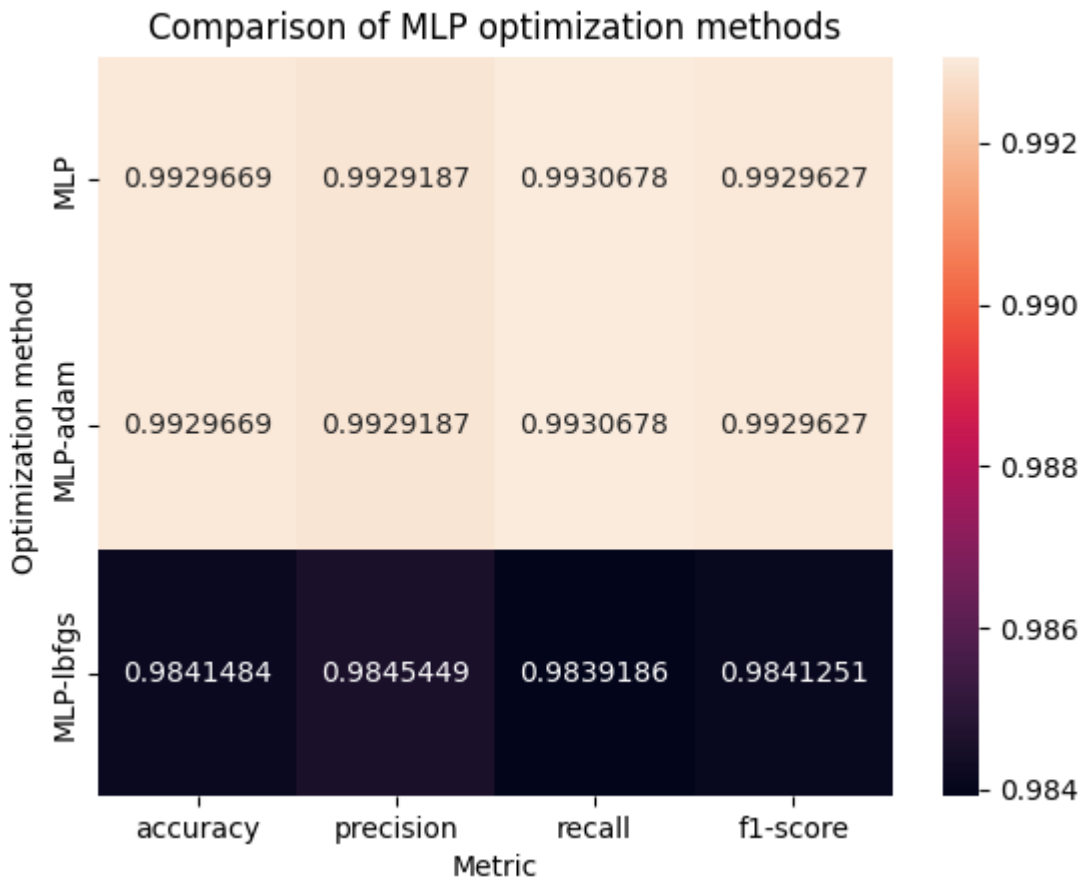

Рисунок 3.1 – Порівняння результатів оптимізаторів з першим варіантом налаштувань та багатошарового перцептрону без оптимізаторів

2) Adam (learning_rate_init=0.005, beta_1=0.7, beta_2=0.888, epsilon=1e-7)

L-BFGS (max_iter=300, max_fun=10000)

З рисунка 3.2 – з такими параметрами Adam аналогічний варіанту без оптимізаторів, L-BFGS приблизно дорівнює попередньому варіанту

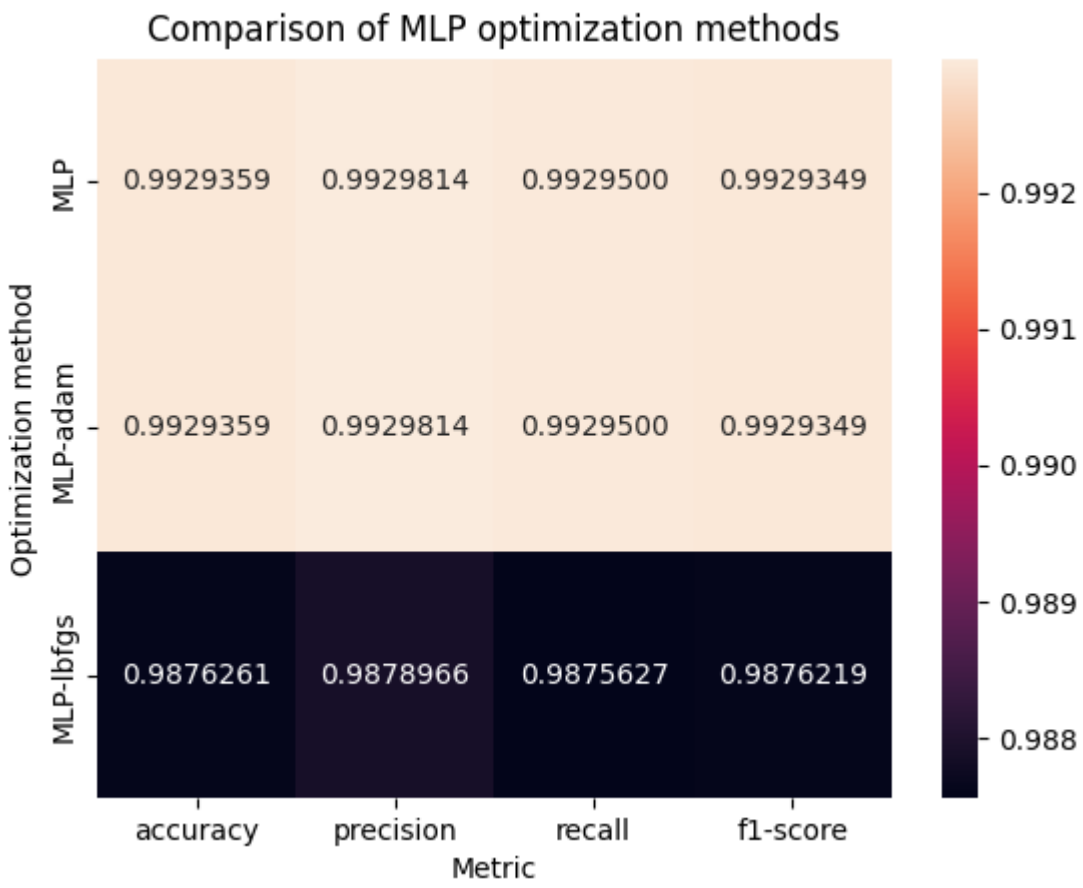

Рисунок 3.2 – Порівняння результатів оптимізаторів з другим варіантом налаштувань та багатошарового перцептрону без оптимізаторів

3) Adam (learning_rate_init=0.05, beta_1=0.1, beta_2=0.3, epsilon=1e-10)

L-BFGS (max_iter=500, max_fun=20000)

З рисунка 3.2 – з такими параметрами Adam показує гірший результат в порівнянні з попередніми спробами, L-BFGS відрізняється від попередніх спроб на десяті долі відсотку, але також гірший.

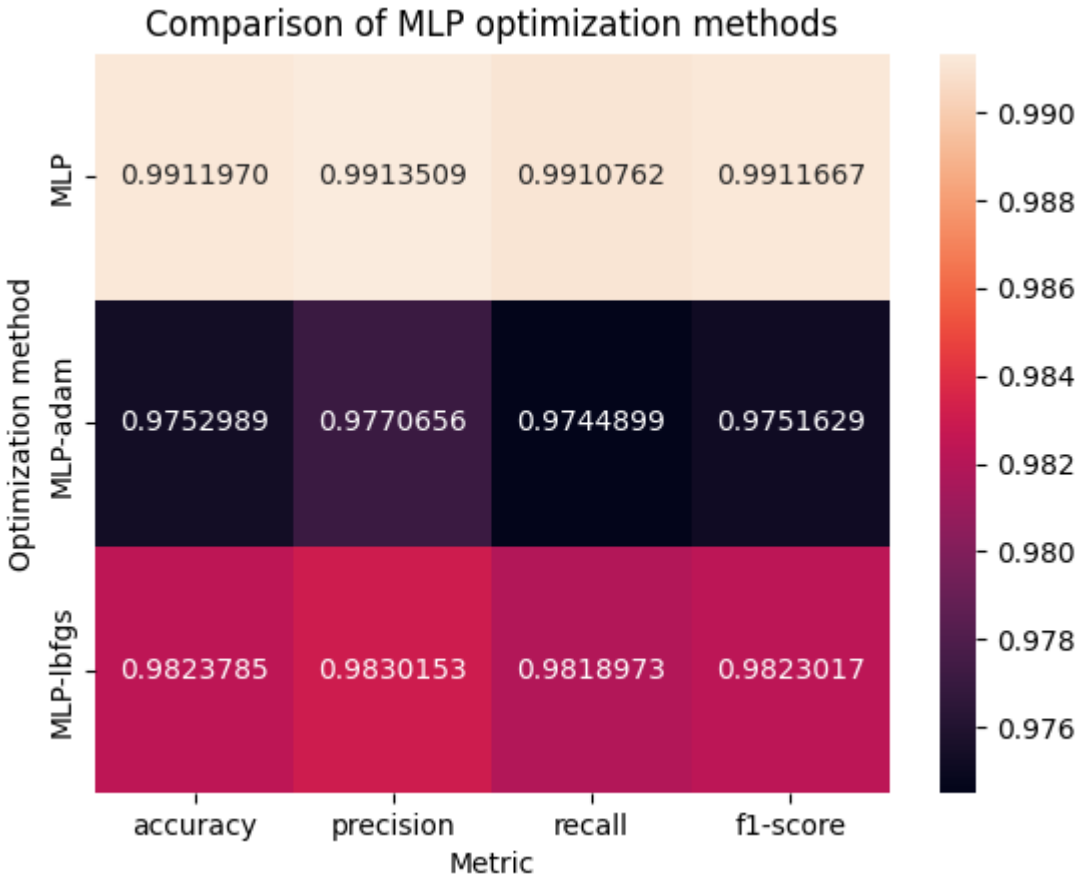

Рисунок 3.3 – Порівняння результатів оптимізаторів з третім варіантом налаштувань та багатошарового перцептрону без оптимізаторів

Тож, як бачимо – метод підбору погано працює в контексті мого завдання, тому можна спробувати метод автоматичного підбори гіперпараметрів моделі. Тому скористаємось автоматичною підбіркою гіперпараметрів: ми визначаємо простір пошуку для гіперпараметрів класифікатора MLP та використовуємо алгоритм Tree-structured Parzen Estimator (TPE) з бібліотеки Hyperopt для знаходження оптимальних значень. Ми також визначаємо цільову функцію, яка оцінює якость моделей за допомогою перехресної перевірки (cross-validation), та запускаємо оптимизацию.

Через 100 ітерацій ми отримуємо найбільш оптимальний набір:

{'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.020073958255417797, 'max_iter': 200.0, 'solver': 'adam'}

Відповідно до рисунку 3.4 – результати з оптимізатором adam покращились на десяту долю відсотка, що може бути похибкою, так як на рисунку 3.5 видно, що там різниця ще менша. Тож, оптимізатори не дають значного виграшу.

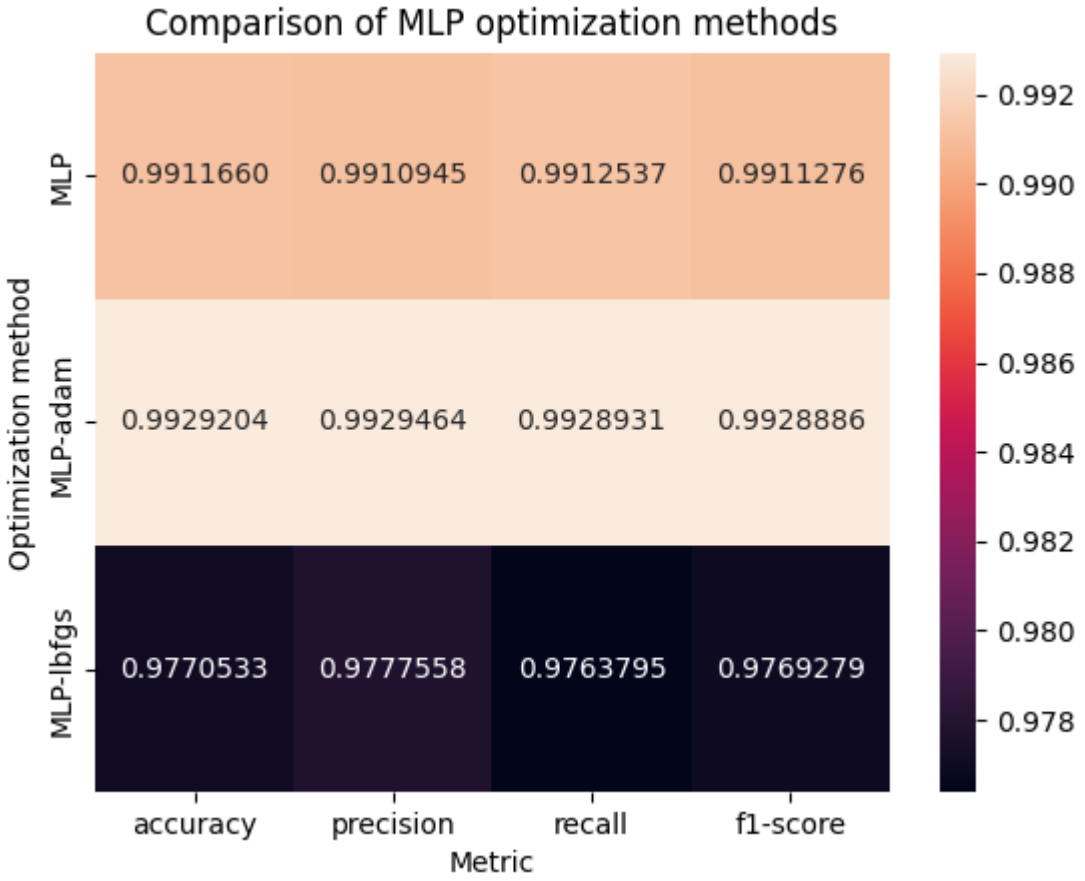

Рисунок 3.4 – Порівняння результатів оптимізаторів (гіперпараметри adam були автоматично підібрані) з моделлю без додаткових оптимізацій

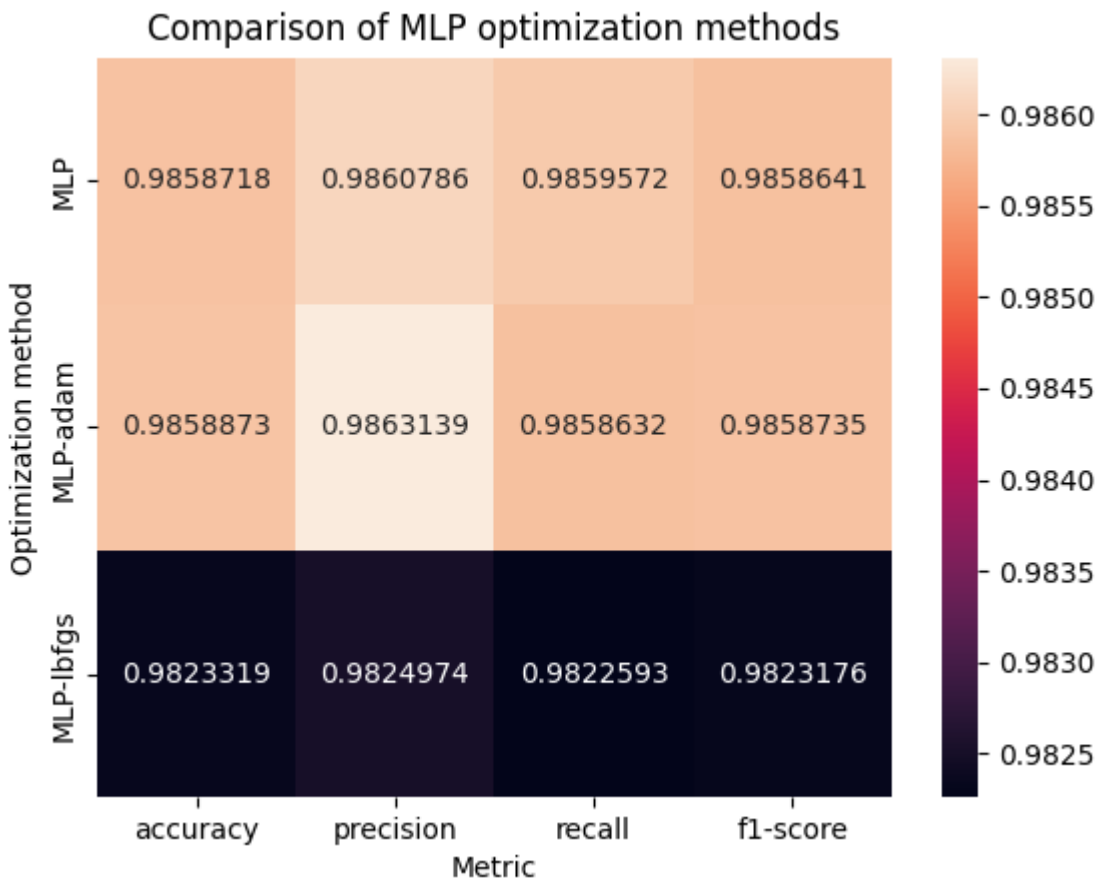

Рисунок 3.5 – порівняння результатів оптимізаторів (гіперпараметри adam були автоматично підібрані) з моделлю без додаткових оптимізацій

3.3 Послуги аналітика

Так як оптимізація гіперпараметрів моделі не спрацювала, то можна скористуватись послугами аналітика. Сценарій наступний: модель не впевнена в своєму виборі (в моєму коді це менше 80%) і пропонує аналітику перевірити її рішення. Якщо рішення аналітика не співпадає з рішенням системи, то модель довчається враховуючи оновлені дані.

Тож як приклад було розроблено додаткову функцію check_mlp_confidence, котра отримує впевненості моделі для кожного випадку і якщо ця впевненість менше 80% (можна налаштувати), то додає їх у відповідний список. Тепер аналітик може продивитись усі листи з цього списку та змінити їхні класифікації. Якщо класифікація аналітика не співпадає з класифікацією моделі – модель навчається на змінених даних.

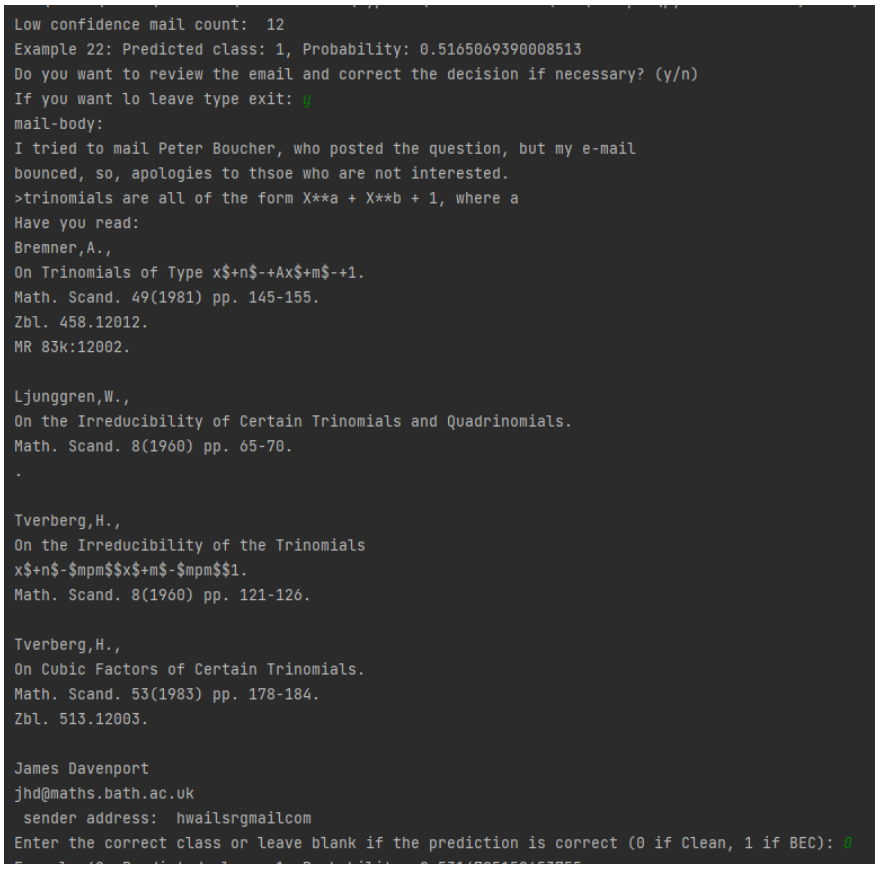

Рисунок 3.6 – Система каже користувачу, що є 12 листів, в котрих вона не впевнена, користувач вирішує, чи хоче він переглянути лист, після чого мануально його класифікує. Якщо листи закінчились, або замість вибору користувач вийшов з меню – система довчається на всіх відповідях користувача, котрі не співпали з рішенням системи

Ще один можливий варіант – користуватись редакціями користувачів електронної пошти. В такому випадку в кожного користувача буде свій “рейтинг аналітика”, що призначається відповідно до його правильних рішень і наступні рішення будуть мати більшу вагу. При такому сценарії замість команди аналітиків можна користуватись рішеннями користувачів, що зменшить фінансові витрати і буде відповідати очікуванням цільової аудиторії.

Висновки до розділу 3

Найкращу продуктивність продемонстрував багатошаровий перцептрон у комбінації зі стематизацією, але якщо говорити в контексті впливу методу

попередньої обробки на модель не враховуючи наявність решти моделей, то k найближчих сусідів у комбінації зі стематизацією а також метод опорних векторів у комбінації з видаленням стоп-слів показали значне покращення результатів.

Отже, використання стандартних оптимізаторів adam та lbfgs продемонструвало погіршення результатів багатошарового перцептрона це може бути пов’язано з тим, що архітектура мережі занадто складна або недостатньо складна для задачі, це може призвести до перенавчання або недонавчання. Аналогічно, якщо данні не масштабовано або не нормалізовано належним чином, це також може призвести до поганих результатів. Застосування автоматичного підбору гіперпараметрів також не дало результату і просто прирівняло оптимізовану модель з неоптимізованною.

Також був розглянутий варіант при якому в якості додаткового вчителя використовується людина. В даному випадку якщо модель не впевнена в додатково поступаючих листах, то вона запитує допомоги в аналітика, після чого використовує виправлені дані для навчання.

Таким чином було виявлено, що багатошаровий перцептрон без додаткових налаштувань з моїм набором даних працює краще, ніж оптимізовані за допомогою adam та lbfgs. Тож, в контексті моєї роботи найкращий ефект буде мати додаткове аналізування класифікацій нейронної мережі людиною так як це дозволить виправити всі помилки моделі і досягти результату, близького до 100%

ВИСНОВКИ

Результати даної роботи підкреслюють необхідність уважного тестування та перевірки всіх “покращень” для методів машинного навчання. Було виявлено, що більшість оптимізацій, які мали на меті підвищення якості роботи моделей насправді спричиняли погіршення метрик, а тому і результативності.

Було виявлено, що різні методи попередньої обробки мають різний вплив на окремі методи. Наприклад, видалення стоп-слів мало дуже негативний вплив на k-найближчих сусідів, але позитивно вплинуло на метод опорних векторів. В свою чергу жоден метод попередньої обробки не мав значного впливу на багатошаровий перцептрон, тобто цей метод відмінно працює з текстами будь-якого формату.

Обравши найбільш успішний метод машинного навчання (багатошаровий перцептрон) після аналізу послідувала спроба оптимізувати його для покращення його результативності, проте, не дивлячись на призначення оптимізаторів вони тільки погіршили його роботу (або просто не дали вагомих результатів, аби зробити висновок, що вони допомогли), тому найкращою реалізацією програми (на основі досліджених методів) стане використання багатошарового перцептрона з попередньою обробкою тексту у вигляді видалення стоп-слів із наступним контролем від аналітика, котрий буде спроможний виправити помилки моделі, після чого вона довчить оновлені дані.

На фоні класичних методів, наприклад, червоних прапорців, цей підхід краще адаптується до нових видів атак та дозволить зменшити витрати на контролюючу команду, так, як достатньо декілька людей для навчання моделі на невідомих їй шаблонах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

[1] Що таке порушення безпеки корпоративної електронної пошти? | Захисний комплекс Microsoft – https://www.microsoft.com/uk-ua/security/business/security-101/what-is-business-email-compromise-bec

[2] What Is Vendor Email Compromise? A Guide to VEC | Armorblox – https://www.armorblox.com/learn/vendor-email-compromise/

[3] How to Prevent Vendor Email Compromise (VEC) Attacks (xorlab.com) – https://www.xorlab.com/en/blog/how-to-prevent-vendor-email-compromise-vec-attacks

[4] Гущин І.В., Сич Д.О. АНАЛІЗ ВПЛИВУ ПОПЕРЕДНЬОЇ ОБРОБКИ ТЕКСТУ НА РЕЗУЛЬТАТИ ТЕКСТОВОЇ КЛАСИФІКАЦІЇ – Харківський національний університет імені В.Н. Каразіна, 2018р

[5] sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.2.2 documentation – https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[6] sklearn.preprocessing.OneHotEncoder — scikit-learn 1.2.2 documentation – https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[7] sklearn.neural_network.MLPClassifier – https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Додаток А

Програмний код основного застосунка:

```
import pandas as pd
import re
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk.corpus import stopwords
import nltk
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_validate
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, make_scorer
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import xlsxwriter


# Завантаження даних з файлу CSV
data = pd.read_csv('D_dataset.csv')

# Зміна назв стовпців у датафреймі
data.columns = ['mail_body', 'user_fullname', 'sender_address', 'label']

# Створення нового стовпця з оригінальним текстом листа
data['original_mail_body'] = data['mail_body']


# Видалення рядків з пропущеними значеннями
data = data.dropna()


# Функція для отримання важливих ознак для MLP класифікатора
def get_important_features_for_mlp(X_test, y_test, y_pred, preprocessor,
mlp_clf):
    # Отримання індексів, де передбачення вірні та клас дорівнює 1
    mlp_correct_indices = np.where((y_test == y_pred) & (y_test == 1))[0]
    # Трансформація даних X_test за допомогою preprocessor
                                            X_test_transformed =
preprocessor.transform(X_test.iloc[mlp_correct_indices])
    # Отримання коефіцієнтів MLP класифікатора
    coefs = mlp_clf.coefs_[0]
    # Отримання індексів 10 найбільших коефіцієнтів для кожного класу
          top_coefs = np.argpartition(np.abs(coefs), -10, axis=0)[-10:,
:min(X_test_transformed.shape[1], coefs.shape[0])]
    # Отримання імен ознак з preprocessor
                                                feature_names =
preprocessor.named_transformers_['body'].get_feature_names_out()

    # Створення нового файлу Excel та додавання робочого аркуша
    workbook = xlsxwriter.Workbook('important_features.xlsx')
    worksheet = workbook.add_worksheet()
```

```
    # Запис заголовка рядка
    worksheet.write(0, 0, 'Class')
    worksheet.write(0, 1, 'Feature')
    worksheet.write(0, 2, 'Coefficient')
    worksheet.write(0, 3, 'Count')

    row = 1
    feature_counts = {}
    written_feature_names = []

    # Прохід по індексах найбільших коефіцієнтів для кожного класу
    for i in range(top_coefs.shape[1]):
        for j in top_coefs[:, i]:
            if j < len(feature_names):
                # Отримання імен ознак та коефіцієнтів
                feature_name = feature_names[j]
                coef = coefs[j, i]

                # Запис даних у робочий аркуш
                worksheet.write(row, 0, i)
                worksheet.write(row, 1, feature_name)
                worksheet.write(row, 2, coef)

                # Оновлення лічильника ознак
                if feature_name in feature_counts:
                    feature_counts[feature_name] += 1
                else:
                    feature_counts[feature_name] = 1

                # Додавання імен ознак до списку записаних імен ознак
                written_feature_names.append(feature_name)

                row += 1

    # Запис лічильників ознак у робочий аркуш
    for i in range(1, row):
        feature_name = written_feature_names[i - 1]
        count = feature_counts.get(feature_name, 0)
        worksheet.write(i, 3, count)

    # Закриття робочої книги
    workbook.close()

# Функція для обробки тексту (перетворення у нижній регістр,
# видалення пунктуації та зайвих пробілів,
# стемминг та лемматизация слів,
# видалення стоп-слів)
def process_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text)

    stemmer = SnowballStemmer('english')
    stop_words = stopwords.words('english')

    words = nltk.word_tokenize(text)

    if stemmer is not None:
        words = [stemmer.stem(word) for word in words]

        lemmatizer = WordNetLemmatizer()
        words = [lemmatizer.lemmatize(word) for word in words]

    words = [word for word in words if word not in stop_words]
```

```
    text = ' '.join(words)

    return text


def check_mlp_confidence(pipeline, X_test, y_test, X_train, y_train):
    # Отримання ймовірностей для кожного класу
    y_proba = pipeline.predict_proba(X_test)

    # Отримання максимальних ймовірностей для кожного прикладу
    max_proba = np.max(y_proba, axis=1)

    # Отримання індексів прикладів, де впевненість менше 80%
    low_confidence_idx = np.where(max_proba < 0.8)[0]
    print("Low confidence mail count: ", len(low_confidence_idx))
    # Прохід по індексах з низькою впевненістю
    for idx in low_confidence_idx:
        # Отримання передбаченого класу та його ймовірності
        pred_class = pipeline.classes_[np.argmax(y_proba[idx])]
        pred_proba = max_proba[idx]

        # Виведення інформації про приклад
            print(f'Example {idx}: Predicted class: {pred_class}, Probability:
{pred_proba}')

        # Запит на перегляд листа користувачем
           user_input = input('Do you want to review the email and correct the
decision if necessary? (y/n)\n'
                           'If you want lo leave type exit: ')

        # Якщо користувач хоче переглянути лист
        if user_input.lower() == 'y':
            # Виведення оригінального тексту листа
             print("mail-body: ", X_test['original_mail_body'].iloc[idx], "sender
address: ",
                   X_test['sender_address'].iloc[idx])

            # Запит на корекцію рішення
             correct_class = int(input('Enter the correct class or leave blank if
the prediction is correct '
                                       '(0 if Clean, 1 if BEC): '))

            # Якщо користувач ввів коректний клас
            if correct_class != pred_class:
                # Заміна мітки у тестових даних на коректний клас
                y_test.iloc[idx] = correct_class

                # Перевчання моделі з урахуванням нових даних
                X_train = X_train + X_test.iloc[idx]
                y_train = y_train +y_test.iloc[idx]
                pipeline.fit(X_train, y_train)
        elif user_input.lower() == 'exit':
            pipeline.fit(X_train, y_train)
            return 0


# Застосування функції обробки тексту до стовпців датафрейму
data['mail_body'] = data['mail_body'].apply(process_text)
data['user_fullname'] = data['user_fullname'].apply(process_text)
data['sender_address'] = data['sender_address'].apply(process_text)

# Розділення даних на навчальні та тестові
X_train,   X_test,   y_train,   y_test   =   train_test_split(data[['mail_body',
'user_fullname', 'sender_address',
```

```
'original_mail_body']], data['label'], test_size=0.4)

# Створення об'єкта ColumnTransformer для попередньої обробки даних
preprocessor = ColumnTransformer(
   transformers=[
       ('body', CountVectorizer(), 'mail_body'),
       ('user', OneHotEncoder(handle_unknown='ignore'), ['user_fullname']),
       ('address', OneHotEncoder(handle_unknown='ignore'), ['sender_address'])
   ])

# Створення списку класифікаторів
classifiers = [
   ('MLP', MLPClassifier()),
   ('LogisticRegression', LogisticRegression(max_iter=1000)),
   ('RandomForest', RandomForestClassifier()),
   ('KNeighbors', KNeighborsClassifier()),
   ('SVM', SVC()),
]

# Ініціалізація словників для збереження результатів
results = {}
cv_results_dict = {}

# Прохід по списку класифікаторів
for name, clf in classifiers:
   # Створення конвеєра з попередньою обробкою даних та класифікатором
   pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('classifier', clf)])

   # Якщо класифікатор - логістична регресія
   if name == 'LogisticRegression':
       # Визначення сітки параметрів для пошуку по сітці
       param_grid = {
           'classifier__C': np.logspace(-4, 4, 9),
           'classifier__penalty': ['l2']
       }

       # Створення об'єкта GridSearchCV
       grid_search = GridSearchCV(pipeline, param_grid, cv=5)

       # Навчання GridSearchCV на даних
       grid_search.fit(X_train, y_train)

       # Виведення найкращих параметрів для логістичної регресії
       print(f'Best parameters for {name}: {grid_search.best_params_}')

       # Робота передбачень за допомогою GridSearchCV
       y_pred = grid_search.predict(X_test)

       # Встановлення найкращих параметрів для конвеєра
       pipeline.set_params(**grid_search.best_params_)

   # Якщо класифікатор - MLP
   elif name == 'MLP':
       # Навчання MLP класифікатора та робота передбачень
       pipeline.fit(X_train, y_train)

       check_mlp_confidence(pipeline, X_test, y_test, X_train, y_train)

   # Для інших класифікаторів
   else:
       # Навчання конвеєра на даних
       pipeline.fit(X_train, y_train)
```

```
        # Робота передбачень за допомогою конвеєра
        y_pred = pipeline.predict(X_test)

    # Обчислення метрик для передбачень
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Збереження метрик у словнику results
    results[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1,
    }

    # Виведення метрик для поточного класифікатора
    print(f'{name} metrics:')
    print(f' Accuracy: {accuracy}')
    print(f' Precision: {precision}')
    print(f' Recall: {recall}')
    print(f' F1-score: {f1}')

    # Визначення метрик для перехресної перевірки
    scoring_metrics = {'accuracy': make_scorer(accuracy_score),
                       'precision': make_scorer(precision_score),
                       'recall': make_scorer(recall_score),
                       'f1': make_scorer(f1_score)}

    # Виконання перехресної перевірки для конвеєра
            cv_results    =    cross_validate(pipeline,    X_train,    y_train,
scoring=scoring_metrics)

    # Збереження результатів перехресної перевірки у словнику cv_results_dict
    cv_results_dict[name] = cv_results

    # Виведення результатів перехресної перевірки для поточного класифікатора
    print(f'{name} cross-validation metrics:')

    for metric in scoring_metrics.keys():
        mean_score = cv_results[f'test_{metric}'].mean()

        print(f' Mean {metric}: {mean_score}')

# Виведення теплової карти з результатами для усіх класифікаторів
sns.heatmap(pd.DataFrame(results).T, annot=True, fmt='.4f')
plt.show()

# Створення датафрейму з результатами перехресної перевірки для усіх
класифікаторів
cv_results_df = pd.DataFrame(
      {name: {f'Mean test {metric}': cv_results[f'test_{metric}'].mean() for
metric in scoring_metrics.keys()} for
    name, cv_results in cv_results_dict.items()})
# Виведення теплової карти з результатами перехресної перевірки для усіх
класифікаторів
sns.heatmap(cv_results_df.T, annot=True, fmt='.4f')
plt.show()
```

Додаток Б

Таблиця Б.1 – Найбільш вагомі слова, що вплинули на правильну класифікацію багатошаровим перцептроном

Class	Feature	Count
5	semiconductor	33
7	andi	33
12	hooper	33
14	touch	33
17	fairchild	33
18	bruce	33
19	issa	33
20	valid	33
27	arlington	33
30	va	33
31	nough	33
35	trademark	33
44	hilari	33
45	bodi	33
49	knife	33
50	orman	33
64	occam	33
67	butcher	33
69	edward	33
73	noscblkhole	33
74	confer	33
76	taker	33
81	fl	33
83	finit	33
86	creation	33
87	warn	33
90	greed	33

Додаток В

Код програми для порівняння оптимізованих багатошарових перцептронів зі звичайним:

```
import pandas as pd

import re

from nltk.stem import SnowballStemmer, WordNetLemmatizer

from nltk.corpus import stopwords

import nltk

from sklearn.model_selection import train_test_split, GridSearchCV, cross_validate

from sklearn.compose import ColumnTransformer

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural_network import MLPClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, make_scorer

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import xlsxwriter


# Завантаження даних з файлу CSV

data = pd.read_csv('D_dataset.csv')


# Зміна назв стовпців у датафреймі

data.columns = ['mail_body', 'user_fullname', 'sender_address', 'label']


# Створення нового стовпця з оригінальним текстом листа

data['original_mail_body'] = data['mail_body']
```

```
# Видалення рядків з пропущеними значеннями
data = data.dropna()

def process_text(text):
   text = text.lower()
   text = re.sub(r'[^\w\s]', '', text)
   text = re.sub(r'\s+', ' ', text)

   stemmer = SnowballStemmer('english')
   stop_words = stopwords.words('english')

   words = nltk.word_tokenize(text)

   if stemmer is not None:
       words = [stemmer.stem(word) for word in words]

       lemmatizer = WordNetLemmatizer()
       words = [lemmatizer.lemmatize(word) for word in words]

   words = [word for word in words if word not in stop_words]

   text = ' '.join(words)

   return text

data['mail_body'] = data['mail_body'].apply(process_text)
data['user_fullname'] = data['user_fullname'].apply(process_text)
data['sender_address'] = data['sender_address'].apply(process_text)

# Розділення даних на навчальні та тестові
X_train, X_test, y_train, y_test = train_test_split(
                 data[['mail_body',     'user_fullname',     'sender_address',
'original_mail_body']],
   data['label'],
```

```
    test_size=0.4
)

# Створення об'єкта ColumnTransformer для попередньої обробки даних
preprocessor = ColumnTransformer(
    transformers=[
        ('body', CountVectorizer(), 'mail_body'),
        ('user', OneHotEncoder(handle_unknown='ignore'), ['user_fullname']),
        ('address', OneHotEncoder(handle_unknown='ignore'), ['sender_address'])
    ]
)

# Створення списку класифікаторів
classifiers = [
    ('MLP', MLPClassifier()),
    ('MLP-adam', MLPClassifier(solver='adam')),
    ('MLP-lbfgs', MLPClassifier(solver='lbfgs')),
    #('MLP-sgd', MLPClassifier(solver='sgd', max_iter=1000)),
]

# Ініціалізація словників для збереження результатів
results = {}
cv_results_dict = {}

# Прохід по списку класифікаторів
for name, clf in classifiers:
    # Створення конвеєра з попередньою обробкою даних та класифікатором
      pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('classifier',
clf)])

    # Навчання класифікатора та робота передбачень
    pipeline.fit(X_train, y_train)

    if name.startswith('MLP'):
        cv_results = cross_validate(
```

```
            pipeline,
            X_train,
            y_train,
            cv=5,
            scoring=['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
        )

        cv_results_dict[name] = {
            'accuracy': cv_results['test_accuracy'].mean(),
            'precision': cv_results['test_precision_macro'].mean(),
            'recall': cv_results['test_recall_macro'].mean(),
            'f1-score': cv_results['test_f1_macro'].mean()
        }

# Графічне порівняння результатів застосування різних методів оптимізації для багатошарового перцептрону
sns.heatmap(pd.DataFrame(cv_results_dict).T, annot=True, fmt='.7f')
plt.title('Comparison of MLP optimization methods')
plt.xlabel('Metric')
plt.ylabel('Optimization method')
        plt.show()
```

Додаток Г

Програмний код для виділення необхідних полів з .json файлу

```
import csv
import glob
import json
import re

json_files = glob.glob('*.json')

def remove_html_tags(text):
   clean = re.compile('<.*?>')
   return re.sub(clean, '', text)

with open('output.csv', 'w', encoding='utf-8', newline='') as file:
   writer = csv.writer(file)
   writer.writerow(['body', 'fullname', 'mail_address'])
   for json_file in json_files:
       with open(json_file, 'r', encoding='utf-8') as file:
           data = json.load(file)
       if isinstance(data, dict):
           body = remove_html_tags(data.get('body', ''))
           fullname = data.get('sender_email_address', {}).get('fullname', '')
                           mail_address   =   data.get('sender_email_address',
{}).get('mail_address', '')
           writer.writerow([body, fullname, mail_address])
```