НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАТЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ТА АНАЛІЗУ ДАНИХ

«До захисту допущено»
Виконувач обов'язків
завідувача катедри

_____ Іван ТЕРЕЩЕНКО

«_____»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності 113 Прикладна математика

на тему: «Методи класифікації шкідливих електронних листів на прикладі імпосторів»

Виконала: здобувачка вищої освіти 4 курсу, групи ФІ-92 Гаврилова Анастасія Володимирівна

Керівниця: старша викладачка, докторка філософії Яйлимова Ганна Олексіївна _____

(підпис)

Рецензентка: доцентка катедри інформаційної безпеки Носок Світлана Олександрівна

(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.

Здобувачка вищої освіти _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Катедра математичного моделювання та аналізу даних

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 113 «Прикадна математика»
Освітньо-професійна програма «Математичні методи моделювання, розпізнавання образів і комп'ютерного зору»

ЗАТВЕРДЖУЮ
Виконувач обов'язків завідувача катедри
_____ Іван ТЕРЕЩЕНКО

«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу здобувачки вищої освіти
Гаврилової Анастасії Володимирівни

1. Тема роботи *«Методи класифікації шкідливих електронних листів на прикладі імпосторів»*

керівниця роботи: старша викладачка, докторка філософії Яйлимова Ганна Олексіївна

затверджено наказом № по університету від « » 2023 р.

2. Термін подання здобувачкою вищої освіти роботи « » 2023 р.

3. Вихідні дані до роботи: *набори даних із електронними листами, опубліковані джерела за тематикою дослідження.*

4. Зміст роботи: *теоретичні відомості про різні види загроз, що можуть бути отримані в електронних листах, зокрема BEC, методи машинного навчання, методи попередньої обробки текстових даних, метрики для оцінки роботи моделей, виявлення найбільш ефективного методу класифікації імпосторів.*

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

6. Дата видачі завдання 29.09.2022

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Вибір теми	29.09.2022	виконано
2	Опрацювання інформаційних джерел	30.09.2022 - 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	12.04.2023 - 20.04.2023	виконано
7	Вивчення матеріалів для покращення роботи моделей	20.04.2023 - 05.05.2023	виконано
8	Реалізація обраних способів покращення	05.05.2023 - 18.05.2023	виконано
9	Аналіз різних варіантів покращення моделей	19.05.2023 - 31.05.2023	виконано
10	Оформлення дипломної роботи	01.06.2023 - 13.06.2023	виконано

Студентка _____ Анастасія ГАВРИЛОВА

Керівниця роботи _____ Ганна ЯЙЛИМОВА

РЕФЕРАТ

Обсяг роботи 72 сторінки, 19 ілюстрацій, 4 таблиці, 1 додаток, 16 джерел літератури. У роботі проведено аналіз різних методів машинного навчання та результатів роботи моделей у залежності від виду попередньої обробки тексту та вхідних датасетів (зі справжіми електронними листами та з додаванням синтетично згенерованих електронних листів; виявлено найбільш оптимальний метод класифікації Business Email Compromise (BEC) листів.

Метою даної роботи є знаходження та реалізація найкращого методу ідентифікації шкідивих електронних листів класу «імпостор».

Об'єктом дослідження даної роботи є шкідливі електронні листи.

Предметом дослідження даної роботи є методи класифікації шкідливих електронних листів класу «імпостор».

КЛАСИФІКАЦІЯ, BUSINESS EMAIL COMPROMISE, BEC, ІМПОСТОР, ЧЕРВОНІ ПРАПОРЦІ, МАШИННЕ НАВЧАННЯ, СТЕМІНГ, ЛЕМАТИЗАЦІЯ, КРОС-ВАЛІДАЦІЯ, НАЇВНИЙ БАЄС, ДЕРЕВО РІШЕНЬ, МЕТОД ОПОРНИХ ВЕКТОРІВ, АДАПТИВНИЙ БУСТИНГ, КЛАСИФІКАТОР ІЗ КВАДРАТИЧНОЮ МЕЖЕЮ РІШЕННЯ.

SUMMARY

Work summary 72 pages, 19 illustrations, 4 tables, 1 appendice, 16 sources of literature.

The work analyzes different methods of machine learning depending on the type of preprocessing of text and input datasets (with real emails and with the addition of synthetically generated e-mails; the most optimal method of classification of Business E-mail Compromise (BEC) e-mails is identified.

The purpose of this work is to find and implement the best method of identifying malicious impostor e-mails.

The object of research of this work is malicious e-mails.

The subject of research of this work is methods of classification of malicious impostor e-mails.

CLASSIFICATION, BUSINESS E-MAIL COMPROMISE, BEC, IMPOSTOR, RED FLAGS, MACHINE LEARNING, STEMMING, LEMMATIZATION, CROSS-VALIDATION, NAIVE BAYES, DECISION TREE, SVM, ADABOOST, QDA.

Зміст

ВСТУП

У наші часи активний розвиток цифрових технологій і мережі інтернет значно спростив життя людей: у всесвітньому павутинні можна легко знайти будь-яку інформацію за лічені хвилини, спілкуватися з друзями, що знаходяться в інших містах і навіть країнах, відкрити для себе перспективи віддаленої роботи та дистанційного навчання, що стало неабияк актуально під час пандемії COVID-19. Проте є й зворотня сторона медалі, адже розвиток цифрових комунікацій призвів до виникнення такого явища, як кібертероризм. Існує безліч його проявів: від булінгу в інтернеті до серйозних кібератак, котрі вражають системи по всьому світу (наприклад, вірус Petya, котрий був задіяний у масованій кібер-атаці в 2017 році). Що стосується пересічних громадян, вони мають ризик стати жертвами спроб зазіхання на приватні дані чи фінансові активи, а також потрапити під вплив жантажу з боку зловмисників у мережі. До кібер-загроз, що можуть бути отримані через листування електронною поштою, належать фішинг, спам, скам, малвар і BEC. Саме на боротьбу з останнім видом інтернет-шахрайства з цього списку розрахована дана бакалаврська робота. Для створення BEC-листа не потрібно багато знань і вмінь, але протидія імпосторам організована на значно нижчому рівні, ніж протидія іншим видам кібер-загроз. Такі листи здебільшого не містять ні фішингових посилань, ні вкладень із шкідливими документами, тому існує не так багато способів, котрі б могли ідентифікувати даний клас інтернет-шахрайства. Тим часом зловмисники активно користуються цим і за даними ФБР у 2018 році збитки, нанесені імпостор-атаками, склали більше ніж 1,2 млрд доларів США.

Новизна та актуальність

Станом на сьогодні існують різні методи протидії шахрайству в мережі, але не дивлячись на це, у відкритому доступі знайти роботи чи статі, темою яких є ідентифікація BEC за допомогою методів машинного навчання, майже неможливо. У більшості випадків для протидії імпосторам використовують метод виявлення

червоних прапорців, але він є менш раціональним і менш дієвим, ніж потенційне використання методів машинного навчання з метотою класифікації шкідливих електронних листів на прикладі BEC, адже останні вимагають менше людських і фінансових ресурсів, вони є більш адаптивно спроможними, простішими в реалізації та покращенні, відповідно використання моделей машинного навчання є більш вигідним для компаній.

Під час вивчення матеріалів із різних загальнодоступних джерел було переглянуто статі, автори яких займалися дослідженням використання методів машинного навчання для виявлення електронних листів лише класів «спам» і «фішинг». Із цього можна зробити висновок, що в галузі боротьби з імпосторами існує величезне поле для досліджень, засівання якого й почнеться з моєї бакалаврської роботи. У ній будуть представлені реалізації різних методів і їхніх модифікацій із метою підвищення ефективності роботи, також буде розглянуто найпопулярніший метод протидії імпосторам – червоні прапорці, буде проведено порівняльний аналіз, ціллю якого стане вибір найбільш оптимального методу класифікації шкідливих електронних листів класу «BEC».

Мета та завдання дослідження

Об'єкт дослідження – шкідливі електронні листи.

Предмет дослідження – методи класифікації шкідливих електронних листів класу «BEC».

Мета роботи – знаходження та реалізація найкращого методу ідентифікації шкідливих електронних листів класу «імпостор».

Завдання дослідження:

1. Дослідити різні види загроз, що можуть бути отримати в електронних листах, виокремити їхні характерні ознаки.

2. Проаналізувати існуючі методи протидії ВЕС.

3. Описати методи МН, котрі сможна використати для класифікації імпосторів.

4. Зібрати необхідні для тренування моделей дані, зокрема – згенерувати синтетичні ВЕС-листи.

5. Розробити ПЗ, що використовує кілька методів МН, застосувавши різні способи попередньої обробки тексту та вхідні датасети.

6. Проаналізувати за допомогою теплових карт роботу моделей і їхніх модифікацій із метою пошуку найефективнішого методу класифікації імпосторів.

7. Описати перспективи застосування та поліпшення результатів, отриманих під час виконання бакалаврської роботи.

Методи дослідження

Під час проведення наукового дослідження застосовувалися наступні методи:

- Огляд і аналіз літературних джерел (пошук існуючої інформації про різні види загроз, що можуть бути отримані в електронних листах, і методи протидії ним, про способи попередньої обробки тексту та методи машинного навчання, котрі є придатними для вирішення задачі класифікації).

- Збір інформації (створення двох різних датасетів).

- Висування гіпотези (про доцільність використання методів МН для класифікації шкідливих ектронних листів на прикладі імпосторів).

- Моделювання (програмна реалізація методів МН і їхніх модифікацій).

- Вимірювання (отримання метрик моделей).

- Порівняння (аналіз результатів роботи різних моделей).

Практичне значення одержаних результатів

Під час виконання даної роботи були отримані результати, котрі можуть стати основою програмного застосунку, дія котрого була б спрямована на боротьбу зі шкідливими електронними листами класу «BEC». Використання такого засобу протидії імпосторам допоможе значно зменшити витрати на захист від кібер-атак і потенційні збитки, котрі вони можуть нанести.

Апробація результатів роботи та публікацій

Робота була представлена на XXI Науково-практичній конференція студентів, аспірантів та молодих вчених «***Теоретичні і прикладні проблеми фізики, математики та інформатики***», що відбувалась 11-12 травня 2023 року в місті Київ.

1 ВИЗНАЧЕННЯ ТИПІВ ЗАГРОЗ, ЇХНІ ХАРАКТЕРНІ ОЗНАКИ, ЗАСОБИ ПРОТИДІЇ ІНТЕРНЕТ-ШАХРАЙСТВУ

1.1 Види загроз, що можуть бути отримані електронною поштою

Фішинг – шахрайські дії в мережі, що спрямовані на викрадення особистих даних отримувача (паролі, банківські рахунки, тощо). Дана загроза може бути реалізована за допомогою підроблених посилань або файлів, що копіюють зовнішній вигляд оригінальної сторінки з авторизацією для певного акаунту, або фейкових сторінок із оплатою товарів для інтернет-магазинів або інших рахунків, куди необачний отримувач може ввести свої банківські дані та стати жертвою зловмисників.

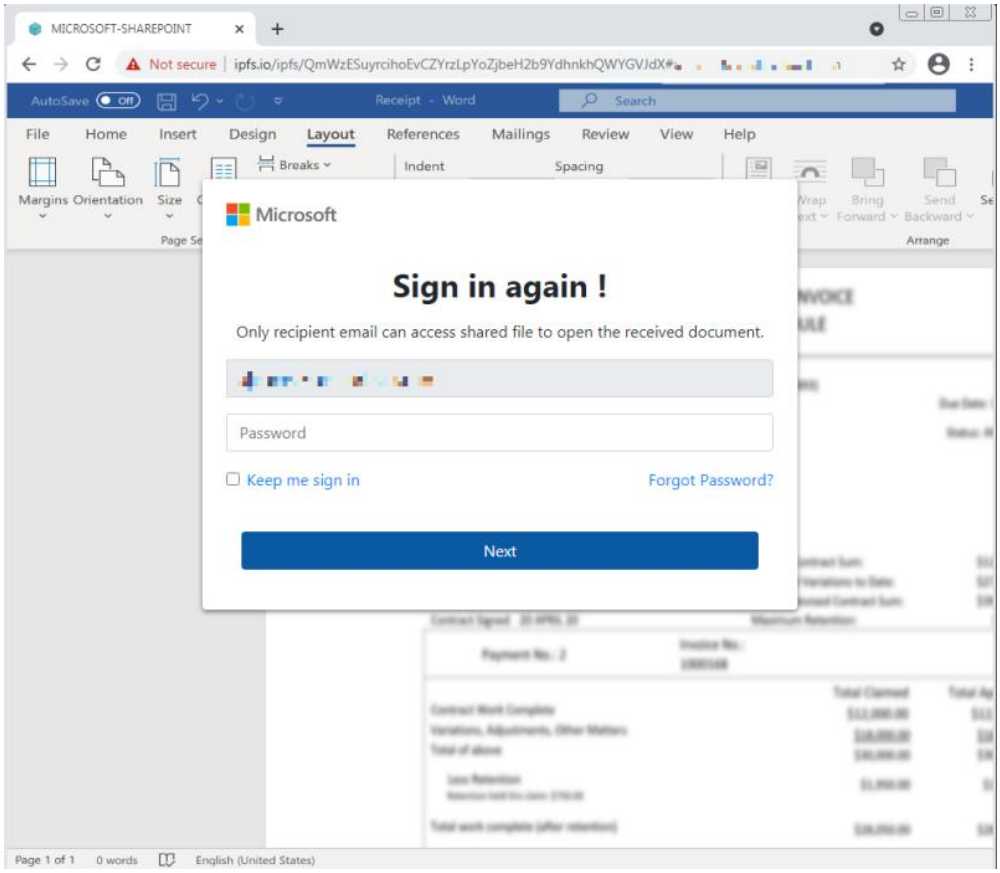
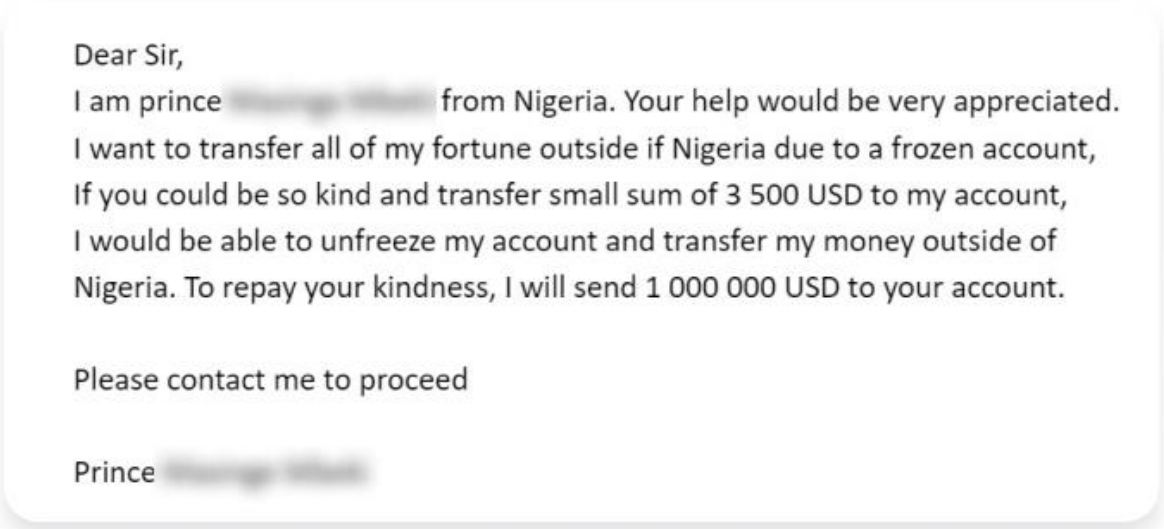


Рисунок 1.1.1 – Приклад фішингової сторінки

Скам – зловмисник намагається спровокувати отримувача на відповідь із метою подальшого шахрайства. Яскравим прикладом є «нігерійський скам» - випадок, коли в електронному листі до людини звертається ніби-то африканський принц і обіцяє великі гроші. Також до скаму належать повідомлення про виграш в лотереї, неочікувані величезні спадки, листи від багатих вдів, тощо.



Dear Sir,

I am prince [redacted] from Nigeria. Your help would be very appreciated.
I want to transfer all of my fortune outside if Nigeria due to a frozen account,
If you could be so kind and transfer small sum of 3 500 USD to my account,
I would be able to unfreeze my account and transfer my money outside of
Nigeria. To repay your kindness, I will send 1 000 000 USD to your account.

Please contact me to proceed

Prince [redacted]

Рисунок 1.1.2 – Приклад нігерійського скаму

Спам – вид шкідливих електронних листів, до котрих належать небажані розсилки, реклама різних чудодійних медичних засобів, пристроїв, казино, а також контент 18+.

Рисунок 1.1.3 – Приклад спамового листа

Малвар – шкідливе ПЗ, котре призначене для нанесення шкоди системі або «шпіонажу», що може бути отримане через вкладений файл у електронному листі або автозавантаження файлу за посиланням.

BEC (business e-mail compromise) або імпостор – вид кібер-шахрайства, котрий полягає в тому, що зловмисник видає себе за іншу людину (як правило – співробітника компанії) з метою отримання її привілеїв.

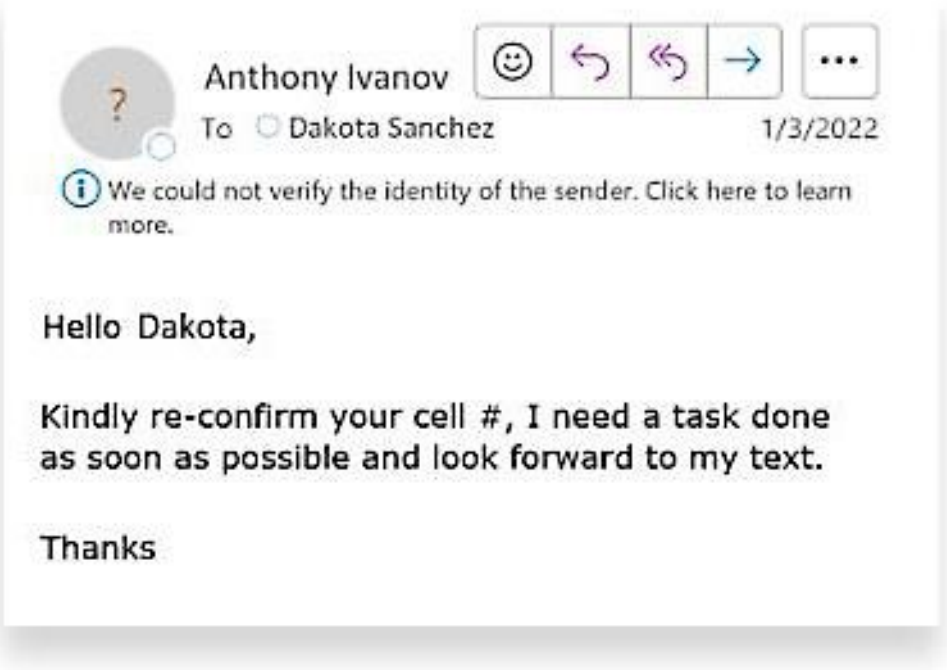


Рисунок 1.1.4 – Приклад BEC

Компанія, на базі якої відбувалася практика, спеціалізується здебільшого на протидії фішингу, тому й вагома частина детекшн логік була орієнтована на ідентифікацію саме цього типу загрози. У відсотковому розподілі спостерігалася наступна картина серед знайдених шкідливих листів: 50-55% - фішинг, 40% - спам/скам, 6% - BEC і 4% - малвар. Проте, варто зауважити, що саме від імпосторів компанії отримують найбільші збитки.

1.2 Види BEC

Імітація користувача – зловмисник на сторонньому домені створює пошту, що схожа на пошту користувача. Наприклад, замість пошти Івана Мартинюка на домені lll.kpi.ua буде пошта зловмисника на домені gmail.com:

ivamar-ipt24@lll.com.ua -> ivamar-ipt24@gmail.com

Імітація адреси домену - зловмисник підроблює домен електронної пошти так, щоб він був схожий на оригінальний домен організації:

lll.kpi.ua -> l1l.kpi.ua (друга літера «l» змінена на одиницю)

Спуфінг домену - зловмисник підроблює домен так, щоб він мав точно такий вигля як і домен організації. Оскільки вони абсолютно однакові, атака є більш переконливою. Протоколи електронної пошти спираються на стандарти автентифікації електронної пошти (SPF, DKIM і DMARC), щоб дозволити власникам доменів «автентифікувати» свою пошту. Якщо в домені не налаштовано ці параметри, шахрай має змогу підробити їх, щоб зробити електронний лист легітимним, але натомість він надходитиме з сервера електронної пошти зловмисника. Отримувач бачить домен організації, але фактичний відправник інший.

Захоплення пошти – зловмисник дійсно має доступ до скриньки жертви, у такому разі отримувачу надсилаються електронні листи з реальної пошти відправника.

1.3 Існуючі методи протидії BEC

Існують різні методи виявлення та запобігання BEC-атакам, які базуються на аналізі електронної пошти та її автентифікації:

1.3.1 Абсорбційна спектроскопія - перевірка стандартів автентифікації електронної пошти, таких як SPF, DKIM та DMARC, виявлення підроблених доменних імен.

В основі цього методу лежить принцип поглинання електромагнітного випромінення речовиною при певних довжинах хвиль. У контексті загроз, отриманих через e-mail, речовиною є електронна пошта, а довжинами хвиль – спеціальні записи DNS, що містять інформацію про дозволені джерела для кожного домену. Ці записи називають SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail) та DMARC (Domain-based Message Authentication, Reporting and Conformance). Із їхньою допомогою отримувачі листів можуть перевіряти справжність електронної пошти, використовуючи криптографічні підписи та політики безпеки.

- SPF - це метод, котрий дозволяє визначити, які IP-адреси дозволені для відправлення електронної пошти від певного домену. SPF записується у вигляді DNS TXT запису на домені відправника та містить список IP-адрес або інших доменів, які можуть надсилати електронну пошту від цього домену. Отримувач електронного листа може перевірити SPF запис, щоб побачити, чи збігається IP-адреса відправника з дозволеними IP-адресами.

- DKIM - це метод, що дозволяє підписувати електронні листи криптографічним підписом, який додається до заголовка e-mail-у. DKIM використовує пару ключів: приватний ключ, який зберігається на сервері відправника та використовується для підписання, і публічний ключ, який записується у вигляді DNS TXT запису на домені відправника та використовується для перевірки підпису. Отримувач електронної пошти може перевірити DKIM підпис, щоб побачити, чи не робилися зміни в листі після підписання.
- DMARC (Domain-based Message Authentication, Reporting and Conformance) - це метод, що дозволяє задавати політику для того, як отримувач електронних листів повинен обробляти електронну пошту, яка не проходить SPF або DKIM перевірку. DMARC записується у вигляді DNS TXT запису на домені відправника та містить параметри, такі як:

  - p - політика для неавтентифікованих повідомлень, яка може бути none (не робити нічого), quarantine (перемістити до спаму) або reject (відхилити).
  - rua - адреса електронної пошти для отримання звітів про результати DMARC перевірки в агрегованому форматі.
  - ruf - адреса електронної пошти для отримання звітів про результати DMARC перевірки в індивідуальному форматі.
  - pct - відсоток повідомлень, до яких застосовується політика DMARC.
  - sp - політика для неавтентифікованих повідомлень із піддоменами, якщо вона відрізняється від основної політики.

Приклад: “v=DMARC1; p=quarantine; rua=mailto:dmarc-reports@example.com; ruf=mailto:dmarc-failures@example.com; pct=100; sp=reject”

Це означає, що всі повідомлення з домену “example.com”, які не проходять SPF або DKIM перевірку, повинні бути переміщені до спаму, а всі повідомлення з

піддоменами “example.com” – відхилені. Звіти про результати перевірки надсилаються на дві різні адреси електронної пошти.

Якщо електронна пошта не має цих записів або не проходить перевірку, це може свідчити про те, що вона була вкрадена, або перехоплена зловмисниками, які намагаються обвести навколо пальця отримувача.

1.3.2 Виявлення червоних прапорців, які можуть свідчити про підозрілу електронну пошту:

- несподіване або незвичне прохання про грошовий переказ, зміну банківських реквізитів або купівлю подарункових карток;
- поганий синтаксис, граматика або орфографія в електронному листі;
- несподівана зміна країни отримання або імені бенефіціара;
- підроблена адреса електронної пошти (схожа на оригінальну, але з невеликими відмінностями, наприклад «facebook.com» -> «face.book.com»);
- обмеження варіантів відповіді на лист, наприклад: «Я у відпустці за кордоном, тому не можу приймати телефонні дзвінки, тільки відповідь на цей лист»;
- невідповідність манери або тону листа до типового способу спілкування відправника.

1.3.3 Застосування рішень для захисту від фішингу – використання спеціального ПЗ або сервісів, які можуть допомогти користувачам ідентифікувати потенційно шкідливий вміст. Такі додатки можуть блокувати, перенаправляти або позначати підозрілі електронні листи та попереджати користувачів і адміністраторів про можливий ризик. Приклади:

- Фільтрація вхідної та вихідної електронної пошти за допомогою безпечних електронних шлюзів (SEG), які перевіряють листи на наявність шкідливих посилань, вкладень, підроблених адрес і інших ознак фішингу.

- Використання хмарних технологій захисту, які сканують вхідну та вихідну електронну пошту на наявність складних загроз, таких як BEC, імітація VIP-осіб або захоплення облікових записів (ATO).

1.3.4 Навчання та освіта співробітників – проведення тренінгів, семінарів і курсів цифрової грамотності, аби навчити співробітників розрізняти фішингові листи, перевіряти електронні листи на достовірність, не надавати конфіденційну інформацію чи грошові перекази без попереднього підтвердження, використовувати багатофакторну автентифікацію та повідомляти мережевого адміністратора про підозрілу активність.

Найбільш поширений спосіб – SAT-інтеграції. Він полягає в тому, що здійснюється фішинг-тренінг для членів кампанії (співробітникам розсилаються фішингові листи, але шкідливі посилання ведуть не на реальну загрозу від шахраїв, а на сторінку вендора, де написано, що користувач наткнувся на шкідливе посилання та даються рекомендації, як можна було б уникнути цієї ситуації).

Висновки до розділу 1

Загрози, що можуть бути отримані через електронну пошту, щоденно наносять великі збитки як компаніям-гігантам, так і пересічним користувачам мережі. Існують різні методи запобігання інтернет-шахрайству, проте саме протидії BEC, на мою думку, приділяють недостатньо уваги. Під час огляду літературних джерел мені не вдалося у відкритому доступі знайти дослідження, котрі б використовували методи МН для боротьби з імпосторами, але потенційно це більш ефективний із багатьох точок зору спосіб протидії даному виду кібер-шахрайства, ніж ті, що використовуються зараз. Отже, в цій роботі підіймається важлива тема запобігання

шкоди, котра може бути отримана від BEC-атак, та пошуку нового та більш ефективного способу для вирішення даної проблеми.

2 МЕТОДИ КЛАСИФІКАЦІЇ ІМПОСТОРІВ

2.1 Класифікатори

Для реалізації задачі класифікації існує кілька відомих методів: метод опорних векторів, класифікатори наївного Баєса, дерево рішень, адаптивний бустінг, класифікатор із квадратичною межею рішення, штучні нейронні мережі, метод найближчих сусідів, логістична регресія, багатошарові перцептрони, тощо. Для перевірки на практиці було обрано перші 5 методів. Розглянемо теоретичні відомості, що вдалося знайти у відкритих джерелах для кожного з них.

Наївні Баєсовські класифікатори - це сімейство алгоритмів машинного навчання, які базуються на теоремі Баєса. Вони називаються “наївними”, тому що роблять припущення про незалежність ознак.

Теорема Баєса дозволяє обчислити ймовірність гіпотези H, даних доказів E, використовуючи наступну формулу:

P(H|E) = P(E|H) * P(H) / P(E), де

P(H|E) - це апостеріорна ймовірність гіпотези H, даних доказів E;

P(E|H) - це ймовірність доказу E, якщо гіпотеза H є істинною;

P(H) - це апріорна ймовірність гіпотези H;

P(E) - це ймовірність доказу E.

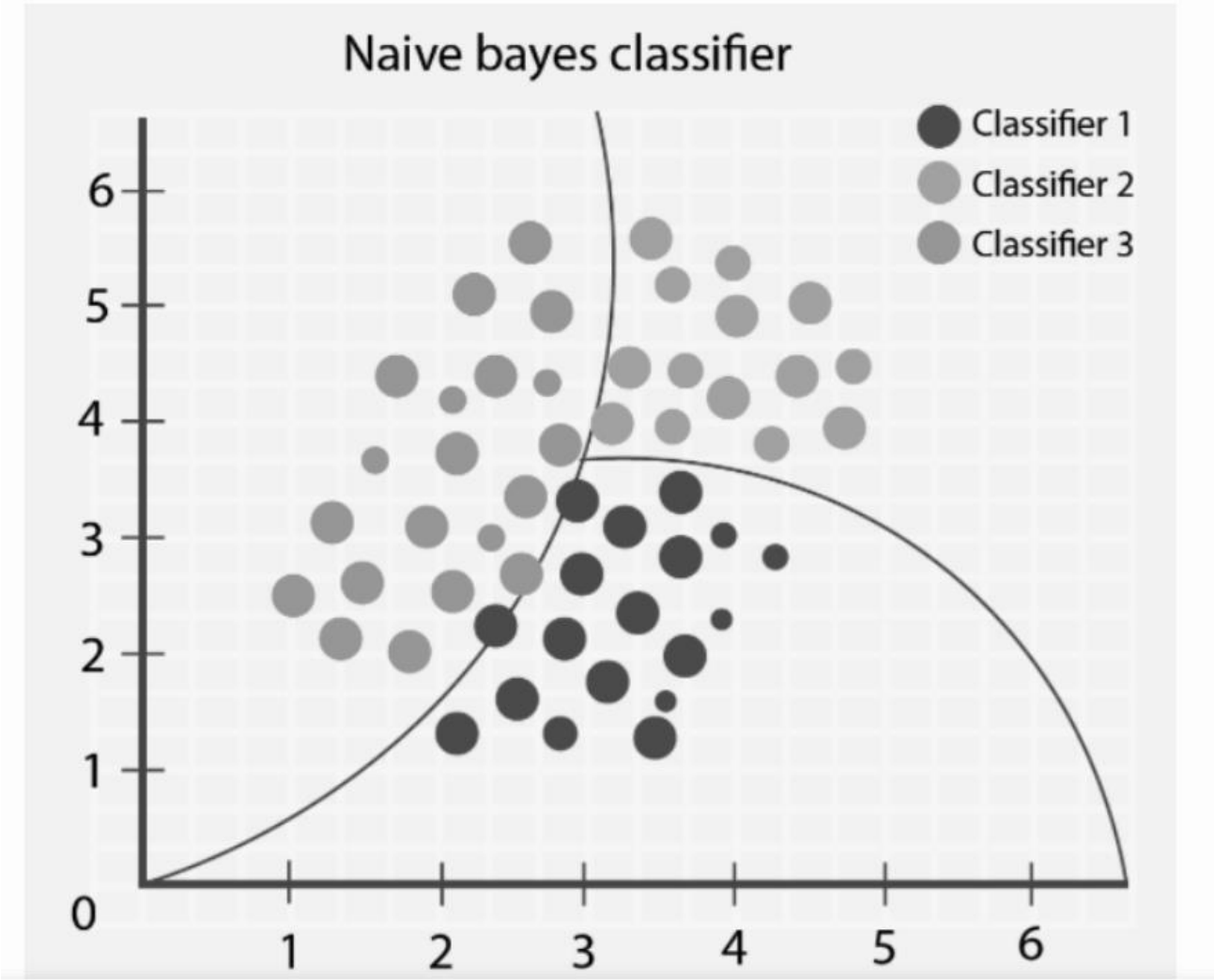


Рисунок 2.1.1 – Візуалізіція роботи класифікатора наївного Баєса

У контексті класифікації H виступає в ролі класу, а E може бути представлений набором ознак. У цьому випадку наївний Баєсовський класифікатор обчислює апостеріорну ймовірність кожного класу, даних ознаках, і передбачає клас із найбільшою ймовірністю, до якого потенційно й належить об'єкт.

Однак наївний Байесовський класифікатор робить припущення про незалежність ознак - відповідно це може стати причиною того, що припущення стосовно присутності або відсутності певної ознаки не впливає на присутність або

відсутність іншої ознаки. Дане припущення робить обчислення більш простим, але може призвести до неточностей у деяких випадках.

Наївні баєсові класифікатори можна застосувати для розв’язання задач класифікації в різних областях. У тому числі вони можуть бути використані для фільтрації ВЕС. У цьому випадку кожен електронний лист класифікується як імпостор або не імпостор.

Для побудови наївного баєсового класифікатора необхідно мати набір даних із позначеними класами. Цей набір даних розділяється на тренувальні та тестові вибірки. Тренувальна виборка даних використовується для обчислення апріорних ймовірностей класів і умовних ймовірностей кожної змінної для кожного класу.

Потім, коли ми отримуємо нове спостереження, з'являється можливість використання обчислених ймовірностей і теореми Баєса для обрахунку апостеріорних ймовірностей кожного класу для цього спостереження. Клас із найбільшою апостеріорною ймовірністю буде призначений цьому спостереженню.

На практиці було розглянуто роботу мультиноміального та гауссівського наївного Баєса.

Мультиноміальний наївний Баєс припускає, що кожен елемент у векторі ознак представляє кількість разів, коли він з’являється (або дуже часто його частоту). Це працює добре для даних, які можна легко перетворити на лічильники, наприклад, кількості слів у тексті.

Гауссівський наївний Баєс базується на неперервному розподілі та підходить для більш загальних завдань класифікації. Цей алгоритм припускає, що дані мають нормальний розподіл та можуть бути неперервними.

Основна різниця між мультиноміальним та гауссівським наївним Баєсом полягає в припущеннях щодо розподілу даних. Мультиноміальний наївний Баєс припускає, що дані є дискретними лічильниками, тоді як гауссівський наївний Баєс припускає, що дані мають нормальний розподіл і можуть бути неперервними.

Дерево рішень може бути використане в задачі класифікації для визначення класу нового спостереження на основі його характеристик.

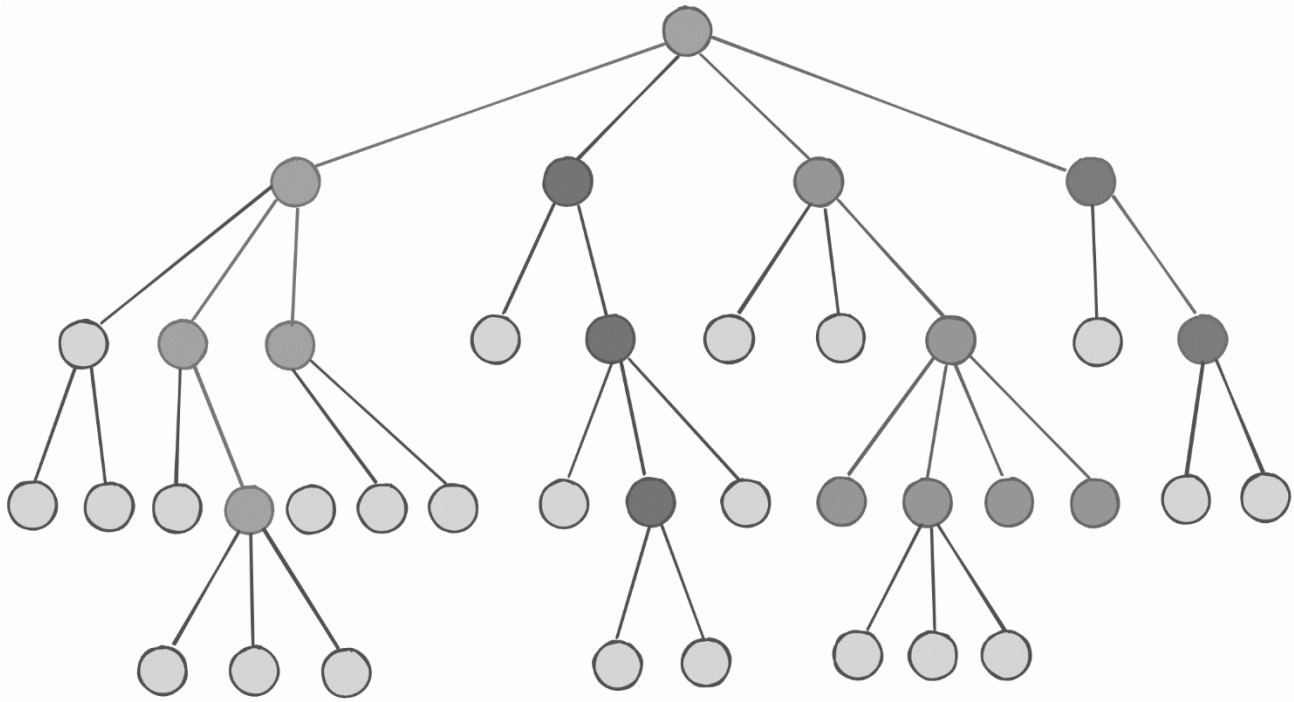


Рисунок 2.1.2 – Візуалізіція принципу роботи дерева рішень

Дерево рішень можна застосувати для класифікації електронних листів-імпосторів. Для побудови дерева рішень необхідно мати набір даних із позначеними класами або значеннями цільової змінної, тобто де кожен лист позначений як імпостор або не імпостор. Цей набір даних розділяється на два набори, як і у

попередньому методі. Тренувальний набір даних використовується для побудови дерева рішень.

При побудові дерева рішень використовуються різні характеристики електронних листів, такі як наявність певних слів у тексті листа, наявність посилань або зображень тощо.

Коли дерево рішень побудоване, його можна застосовувати для класифікації нових спостережень. Для цього необхідно пройти від кореня до листа, вибираючи гілки на основі значень характеристик спостереження. Клас, записаний у листі, призначається цьому спостереженню.

Метод опорних векторів (SVM) - це лінійний алгоритм, що використовується в задачах класифікації та регресії· Цей метод належить до класу ядрових методів і працює шляхом ідентифікації оптимальної межі рішення, яка розділяє точки даних із різних груп (або класів), а потім передбачає клас на основі цього розділення.

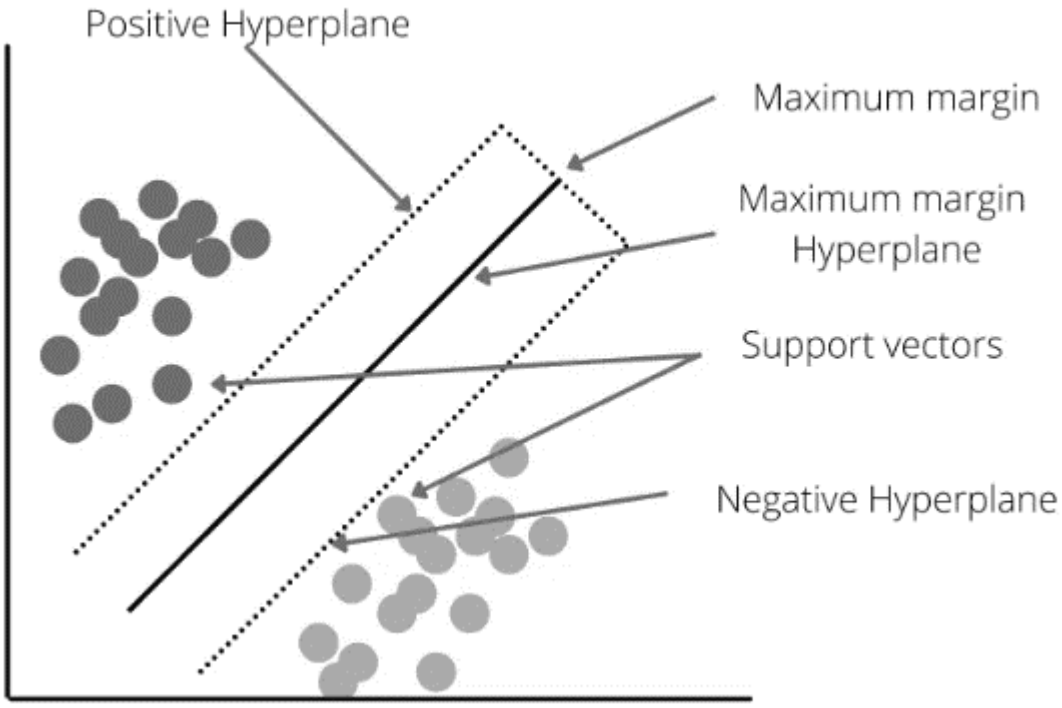


Рисунок 2.1.3 – Візуалізіція роботи SVM

SVM працює шляхом знаходження гіперплощини, яка найкращим чином ділить два класи даних. Гіперплощина обирається таким чином, щоб максимізувати відстань між найближчими точками двох класів (опорними векторами). Це дозволяє алгоритму краще узагальнювати та передбачати клас нових спостережень.

При побудові моделі SVM використовуються різні ознаки електронних листів (певні слова, адреси, імена, сабджекти). На основі цих характеристик будується модель SVM, яка дозволяє класифікувати нові електронні листи як імпостор або не імпостор.

Коли модель SVM побудована, її можна використовувати для класифікації нових електронних листів. Для цього необхідно обчислити значення функції

рішення для нового електронного листа на основі його характеристик. Значення функції рішення визначає клас електронного листа.

Один із популярних способів побудувати модель SVM - це використання бібліотеки машинного навчання Python scikit-learn, що й використовується в програмній реалізації даного методі в цій роботі. Вона має клас SVC, який може виконувати бінарну та багатокласову класифікації на наборі даних.

Однак перед тим, як використовувати SVM для класифікації електронних листів, необхідно перетворити текстовий вміст листів у числовий формат, який можна використовувати як вхідні дані для моделі SVM.

Адаптивний бустинг - це ансамблевий класифікаційний, котрий може бути використаним разом із іншими типами алгоритмів навчання для покращення продуктивності. Вихід інших алгоритмів навчання («слабких учнів») комбінується в зважену суму, яка представляє кінцевий вихід посиленого класифікатора. У більшості випадків адаптивний бустинг застосовується для бінарної класифікації, але він може мати узагальнення до кількох класів або обмежених інтервалів на дійсній прямій. Адаптивність бустингу полягає в тому, що наступні «слабкі учні» налаштовуються на користь тих зразків, котрі були неправильно класифіковані попередніми класифікаторами. У деяких проблемах метод може бути менш схильним до проблеми перенавчання, ніж альтернативні алгоритми. Окремо взяті «учні» можуть бути «слабкими», але за умови, що результат кожного з них трохи краще, ніж вгадування навмання, кінцева модель може бути доведена до зближення до «сильного учня». Хоча адаптивний бустинг здебільшого використовується для поєднання «слабких базових учнів», метод також ефективно поєднує «сильних базових учнів» (таких як дерево рішень), створюючи ще точнішу, ефективнішу модель.

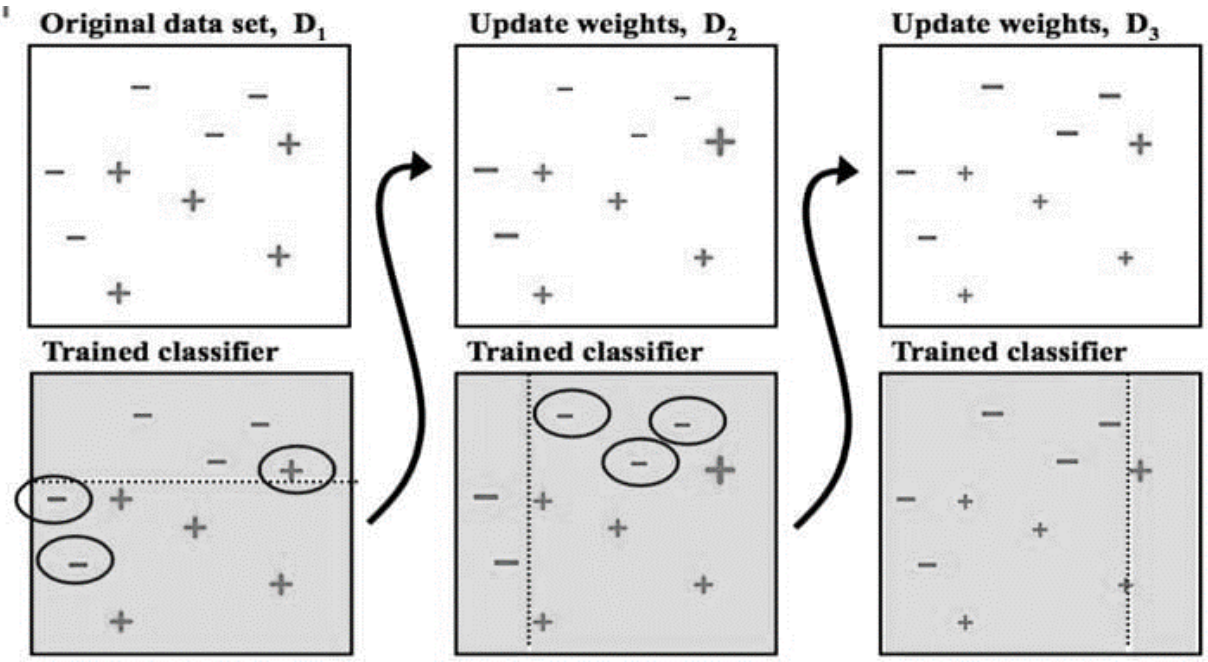


Рисунок 2.1.4 – Візуалізіція принципу роботи AdaBoost

Кожен алгоритм машинного навчання зазвисай є більш ефективним у певному виді завдань, ніж інші, та як правило має багато різних параметрів і конфігурацій для налаштування з метою досягненння оптимальної продуктивності на наборах даних. Адаптивний бустинг - це алгоритм, який часто вибирають за замовчуванням, коли потрібно класифікувати дані. Він використовує прості дерева як «базових учнів». У процесі навчання метод звертає увагу на те, як добре кожне дерево ідентифікувало дані. Ця інформація допомагає побудувати наступні дерева так, щоб вони краще розпізнавали більш складні випадки.

Адаптивний бустинг і дерево рішень можуть мати майже однакові результати, якщо дерево рішень - достатньо глибоке та складне, що дозволяє добре апроксимувати дані. У такому випадку адаптивний бустинг не дає значного покращення, оскільки він використовує прості дерева як «базових учнів». Адаптивний бустинг краще працює, коли «базові учні» є «слабкими» та мають

низьку дисперсію та високе зміщення. Також адаптивний бустинг та дерево рішень можуть мати схожі результати, якщо дані не містять складних залежностей або шуму, які потребують ансамблювання для покращення точності.

Класифікатор із квадратичною межею рішення - це метод машинного навчання, який використовується для класифікації даних. Він є розширенням лінійного дискримінантного аналізу (LDA) й також передбачає, що спостереження з кожного класу розподілені нормально. Метод будує квадратичну границю розділення між двома класами даних.

Цей алгоритм може бути застосований у багатьох областях, зокрема й у класифікації електронних листів.

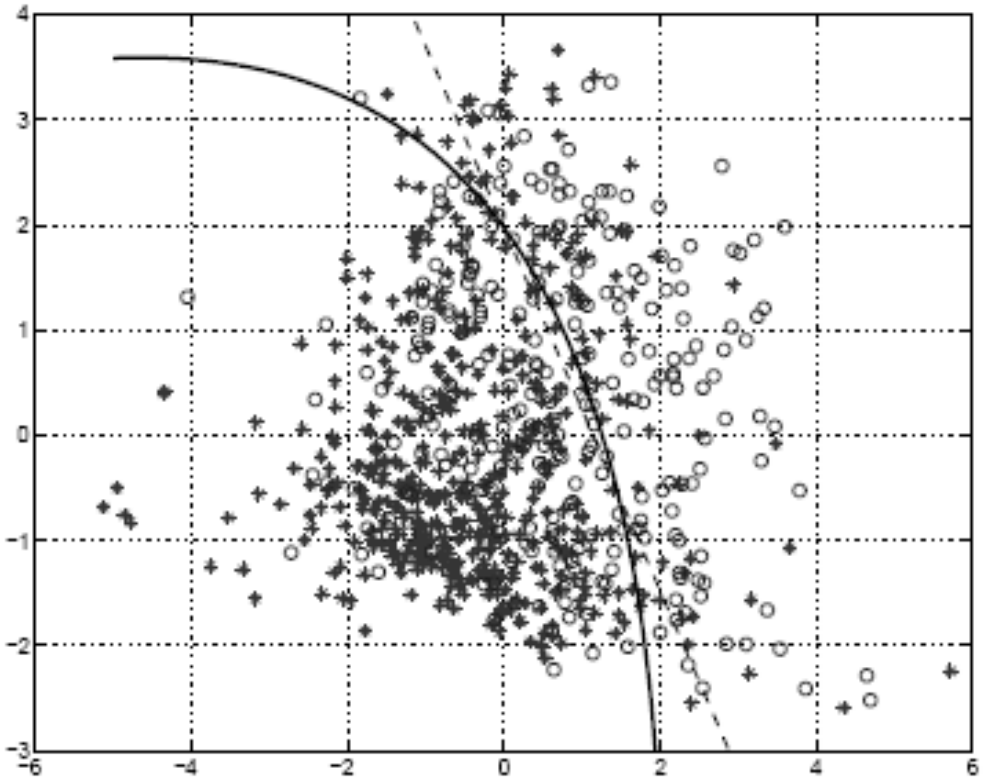


Рисунок 2.1.5 – Візуалізіція роботи класифікатора QDA

2.2. Попередня обробка тексту

Попередня обробка тексту в машинному навчанні – це процес очищення текстових даних та підготовки їх до подальшої роботи, який передбачає перетворення тексту, написаного людиною, до формату, котрий є «зрозумілим для комп'ютера», щоб надалі була змога застосувати до нього методи МН. Електронні листи можуть містити шум різних видів (емодзі, розділові знаки, різні регістри тексту, зайві пробіли). Прибравши ці особливості людською манери спілкування на етапі попередньої обробки, масив даних значно зменшується, як і його складність, ефективність алгоритмів МН збільшується за рахунок зростання точноті та швидкості їхньої роботи. Також попередня обробка текстових даних переводить значення листів у числових вектор, зберігаючи їхній контекст.

Приведення до нижнього регістру – це процес, при якому всі латери в словах стають малими (Paris -> paris).

Стемінг - це процес видалення закінчень і суфіксів слів для отримання їхньої основи (наприклад, apples -> apple).

Лематизація - це процес приведення слів до їхньої нормальної форми (наприклад, wrote -> write), використовує більш комплексний підхід, ніж стемінг, оскільки перед усіма діями, що виконує стемінг, визначає частину мови, до котрої належить слово.

Висновки до розділу 2: На практиці завжди корисно експериментувати з різними моделями та порівнювати їхні результати на перехресній перевірці або тестовому наборі даних. Це допомагає обрати модель, яка найкраще працює для конкретної задачі, тому було прийняте рішення про використання різних методів (метод опорних векторів, класифікатори наївного Баєса, дерево рішень, адаптивний бустинг, QDA), а також скористатися різними видами попередньої обробки тексту (приведення до нижнього регістру, стемінг, лематизація), щоб дослідити їхній

вплив на результат роботи моделей. Також спробуємо покращити роботу методів, використавши додавання синтетичних електронних листів класу «імпостор», отже в результаті буде проведений порівняльний аналіз із урахуванням двох датасетів для реалізації задачі класифікації шкідливих електронних листів на прикладі імпосторів із метою подальшого вибору найефективнішої моделі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Збір і підготовка даних

Для реалізації існуючих методів вирішення задачі класифікації необхідно мати набір даних із уже ідентифікованими електронними листами, де кожен зразок позначений як імпостор або не імпостор. За основу був узятий датасет шкідливих електронних листів із типом загрози BEC із компанії, на базі якої відбувалась практика, а також дані з відкритих джерел (Kaggle), що містили листи з типом загрози «спам». Із метою покращення роботи обраних методів були додані згенеровані за допомогою штучного інтелекту Bing AI синтетичні електронні листи з типом загрози BEC. Дані були скомпоновані в дві таблиці. Одна містила лише справжні листи, а інша – поєднання реальних і синтетичних електронних листів. Кожна з таблиць складалась із п'яти стовбців: 1 – сабджект і текст листа, 2 – адреса відправника, 3 – ім'я відправника, 4 – ім'я отримувача, 5 – клас листа (1, якщо електронний лист є імпостором, та 0, якщо лист належить до іншого типу загроз або зовсім не містить нічого шкідливого).

Під час написання коду для програмної реалізації існуючих методів вирішення задачі класифікації та застосування її для пошуку імпосторів виникла потреба перевести дані текстового формату листів у числовий формат, за допомогою кодування категоричних змінних, щоб «комп'ютер міг їх розуміти». Це є необхідниим для використання scikit-learn. Саме цей модуль дозволяє реалізовувати різні методи для класифікафії, зокрема наївний Баєс, метод опорних векторів, адаптивний бустинг, класифікатор із квадратичною межею рішення та дерево рішень, що були обрані для виконання індивідуального завдання з пошуку імпосторів серед збірки електронних листів. Кожен із методів був реалізований спочатку на датасеті з реальними листами, а потім – на датасеті зі справжніми та синтезованими електронними листами з метою порівняння залежності ефективності роботи як від самих методів, так і від датасетів, що лежали в основі.

Також була поставлена задача дослідити кореляцію між використанням різних методів попередньої обробки текстових даних і результатами, котрі покажуть моделі.

3.2 Опис програмної реалізації

- `import pandas as pd` - імпорт бібліотеки pandas, яка необхідна для роботи з даними в табличному форматі.

- `from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis` - імпорт класу QuadraticDiscriminantAnalysis із бібліотеки sklearn, який є одним із алгоритмів класифікації, що використовується в коді.

- `from sklearn.linear_model import SGDClassifier` - імпорт класу SGDClassifier із бібліотеки sklearn, який є іншим алгоритмом класифікації, що використовується в коді.

- `from sklearn.model_selection import train_test_split, cross_validate` - імпорт функцій train_test_split та cross_validate із бібліотеки sklearn, які дозволяють розділяти дані на навчальні та тестові вибірки та оцінювати ефективність моделей за допомогою крос-валідації.

- `from sklearn.naive_bayes import GaussianNB, MultinomialNB` - імпорт класів GaussianNB і MultinomialNB із бібліотеки sklearn, які є ще двома алгоритмами класифікації, що використовуються в коді. Вони належать до сімейства наївних Байесових класифікаторів, котрі базуються на статистичних припущеннях про розподіл даних.

- `from sklearn.tree import DecisionTreeClassifier` - імпорт класу DecisionTreeClassifier із бібліотеки sklearn, який є ще одним алгоритмом

класифікації, котрий використовується в коді. Він будує дерево рішень, яке розгалужується за допомогою правил на основі ознак даних.

- `from sklearn.svm import SVC` - імпорт класу SVC із бібліотеки sklearn, який є ще одним алгоритмом класифікації, що використовується в коді. Він належить до сімейства методів підтримуючих векторів (SVM), які шукають оптимальну границю між класами даних.

- `from sklearn.preprocessing import LabelEncoder` - імпорт класу LabelEncoder із бібліотеки sklearn, який дозволяє перетворювати рядкові значення на чисельний код для подальшого аналізу.

- `from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_score, f1_score, make_scorer, \ classification_report, confusion_matrix` - це імпорт ряду функцій із бібліотеки sklearn, які дозволяють розраховувати різні метрики продуктивності моделей, такі як точність, повторюваність, площа під кривою ROC, F1-міра, звіт про класифікацію та матриця помилок.

- `import matplotlib.pyplot as plt` - це імпорт бібліотеки matplotlib, за допомогою якої можна будувати графіки та робити візуалізуалізації даних.

- `import seaborn as sns` - це імпорт бібліотеки seaborn, яка є розширенням бібліотеки matplotlib і надає додаткові можливості для візуалізації даних.

- `import nltk` - імпорт бібліотеки nltk, яка є однією з найпопулярніших бібліотек для обробки природної мови (NLP) у Python. Вона надає ряд інструментів для аналізу та маніпуляцій із текстом, таких як токенізація, стемінг, лематизація, тегування частин мови, розпізнавання іменованих сутностей тощо.

- `from nltk.stem import SnowballStemmer, WordNetLemmatizer` - це імпорт класів SnowballStemmer та WordNetLemmatizer із бібліотеки nltk, які дозволяють застосовувати стемінг і лематизацію до текстових даних.

- `from sklearn.ensemble import AdaBoostClassifier, VotingClassifier` - імпорт класів AdaBoostClassifier і VotingClassifier із бібліотеки sklearn, які є прикладами ансамблевих методів машинного навчання. Ансамблевим методом називають той, котрий комбінує прогнози декількох моделей для отримання кращого результату. AdaBoostClassifier - це метод, який покращує слабкого класифікатора за допомогою послідовного навчання на зважених прикладах. VotingClassifier - це метод, який об'єднує прогнози декількох моделей за допомогою голосування (жорсткого або м'якого).

- `data = pd.read_csv('tas_dataset_n.csv')` - завантаження набору даних із csv-файлу за допомогою функції read_csv.

- `data = data.dropna()` - видалення рядків із пропущеними значеннями з набору даних за допомогою функції dropna з бібліотеки pandas. Це допомагає уникнути помилок і спотворень при аналізі даних.

- `def process_text(text):` - визначення функції для обробки текстових даних, яка приймає параметр text, який є рядком.

- `text = text.lower()` - приведення тексту до нижнього регістру за допомогою методу lower із бібліотеки Python. Це допомагає уніфікувати текст і зменшити розмір словника.

- `stemmer = SnowballStemmer('english')` - створення об'єкту stemmer, який є екземпляром класу SnowballStemmer із бібліотеки nltk. Він використовує алгоритм Snowball для стемінгу англомовних слів.

- `token_text = nltk.word_tokenize(text)` - це розбиває текст на окремі слова (токени) за допомогою функції word_tokenize із бібліотеки nltk. Це допомагає аналізувати текст на рівні слів і видаляє зайві символи.

- `if stemmer is not None:` - перевіряє, чи існує об'єкт stemmer, тобто чи був імпортований клас SnowballStemmer. Якщо так, то виконується наступний блок коду.

- `token_text = [stemmer.stem(word) for word in token_text]` - застосування метод ystem до кожного слова в списку token_text за допомогою спискового включення (list comprehension) із бібліотеки Python. Це дозволяє отримати основу кожного слова та зменшити розмір словника.

- `lemmatizer = WordNetLemmatizer()` - це створює об'єкт lemmatizer, який є екземпляром класу WordNetLemmatizer із бібліотеки nltk. Він використовує базу даних WordNet для лематизації англомовних слів.

- `token_text = [lemmatizer.lemmatize(word) for word in token_text]` - застосування методу lemmatize до кожного слова в списку token_text за допомогою спискового включення (list comprehension) із бібліотеки Python. Це дозволяє отримати нормальну форму кожного слова та зменшити розмір словника.

- `text = ' '.join(token_text)` - це об'єднання всіх слів у списку token_text у один рядок, розділяючи їх пробілами за допомогою методу join із бібліотеки Python. Це дозволяє повернути текст у первинний формат, але з обробленими словами.

- `return text` - повернення результату функції process_text, тобто оброблений текст.

- `data['mail'] = data['mail'].apply(process_text)` - застосування функції process_text до кожного значення в стовпці mail набору даних data за допомогою

методу apply з бібліотеки pandas. Це дозволяє обробити текст електронних листів та зберегти результат у тому ж стовпці.

- `data['user'] = data['user'].apply(process_text)` - застосування функції process_text до кожного значення в стовпці user набору даних data за допомогою методу apply з бібліотеки pandas. Це дозволяє обробити текст імен користувачів та зберегти результат у тому ж стовпці.

- `data['s_address'] = data['s_address'].apply(process_text)` - застосування функції process_text до кожного значення в стовпці s_address набору даних data за допомогою методу apply з бібліотеки pandas. Це дозволяє обробити текст адрес електронної пошти відправників та зберегти результат у тому ж стовпці.

- `data['s_name'] = data['s_name'].apply(process_text)` - застосування функції process_text до кожного значення в стовпці s_name набору даних data за допомогою методу apply з бібліотеки pandas. Це дозволяє обробити текст імен відправників та зберегти результат у тому ж стовпці.

- `le = LabelEncoder()` - створення об'єкту le, який є екземпляром класу LabelEncoder із бібліотеки sklearn. Він використовується для кодування категоріальних змінних у чисельний код.

- `data['mail'] = le.fit_transform(data['mail'])` - застосування методу fit_transform до стовпця mail набору даних data за допомогою об'єкта le. Це дозволяє перетворити рядкове представлення електронних листів на чисельний код і зберегти результат у тому ж стовпці.

- `data['user'] = le.fit_transform(data['user'])` - застосування методу fit_transform до стовпця user набору даних data за допомогою об'єкта le. Це дозволяє перетворити рядкове представлення імен користувачів на чисельний код і зберегти результат у тому ж стовпці.

- `data['s_address'] = le.fit_transform(data['s_address'])` - застосування методу fit_transform до стовпця s_address набору даних data за допомогою об'єкта le. Це дозволяє перетворити рядкове представлення адрес електронної пошти відправників на чисельний код і зберегти результат у тому ж стовпці.

- `data['s_name'] = le.fit_transform(data['s_name'])` - застосування методу fit_transform до стовпця s_name набору даних data за допомогою об'єкта le. Це дозволяє перетворити рядкове представлення імен відправників на чисельний код і зберегти результат у тому ж стовпці.

- `X = data[['mail', 'user', 's_address', 's_name']]` - вибирає чотири стовпці з набору даних data, які є ознаками для класифікації, та зберігає їх у змінну X. Це дозволяє виділити вхідні дані для моделей.

- `y = data['class']` - вибирає стовпець class з набору даних data, який є цільовою змінною для класифікації, та зберігає його в змінну y. Це дозволяє виділити вихідні дані для моделей.

- `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)` - розподілення даних на навчальні та тестові вибірки за допомогою функції train_test_split із бібліотеки sklearn. Це дозволяє випадковим чином розподілити дані в співвідношенні 3:2 та зберегти їх у відповідних змінних.

- `models = [` - початок списку моделей для класифікації разом із їхніми гіперпараметрами. Кожна модель представлена у вигляді кортежу з трьох елементів: назви, класу та словника параметрів.

- `('GaussianNB', GaussianNB(), {})` - це перший елемент списку models, який представляє модель GaussianNB. Це наївний Баєсовий класифікатор, який припускає, що ознаки мають гауссовий (нормальний) розподіл. Він не має гіперпараметрів, тому словник параметрів порожній.

- `('MultinomialNB', MultinomialNB(), {'alpha': [0.1, 0.5, 1.0]})` - це другий елемент списку models, який представляє модель MultinomialNB. Це наївний Баєсовий класифікатор, який припускає, що ознаки мають багатономіальний розподіл. Він має один гіперпараметр alpha, який визначає ступінь згладжування (smoothing) для уникнення нульових ймовірностей. Словник параметрів містить список можливих значень alpha.

- `('DecisionTree', DecisionTreeClassifier(), {'max_depth': [3, 5, 10], 'min_samples_split': [2, 5]})` - це третій елемент списку models, який представляє модель DecisionTreeClassifier. Це дерево рішень, яке розгалужується за допомогою правил на основі ознак даних. Воно має два гіперпараметри: max_depth, який визначає максимальну глибину дерева, та min_samples_split, що визначає мінімальну кількість прикладів, необхідних для розгалуження вузла. Словник параметрів містить списки можливих значень для кожного гіперпараметру.

- `('SVC', SVC(probability=True), {'C': [0.1, 1.0], 'kernel': ['linear', 'rbf']})` - це четвертий елемент списку models, який представляє модель SVC. Це метод підтримуючих векторів (SVM), який шукає оптимальну границю між класами даних. Він має два гіперпараметри: C, який визначає ступінь регуляризації (penalty) для запобігання перенавчанню (overfitting), та kernel, який визначає функцію ядра (kernel), що перетворює дані у вищий простір для легшого розділення. Словник параметрів містить списки можливих значень для кожного гіперпараметру. Також задано параметр probability=True для того, щоб модель могла видаляти ймовірності класифікації.

- `('AdaBoost', AdaBoostClassifier(), {'n_estimators': [50, 100], 'learning_rate': [0.1, 1.0]})` - це п'ятий елемент списку models, який представляє модель AdaBoostClassifier. Це ансамблевий метод, який покращує слабкого класифікатора за допомогою послідовного навчання на зважених прикладах. Він має два гіперпараметри: n_estimators, який визначає кількість слабких класифікаторів, що

використовуються для побудови ансамблю, та learning_rate, який визначає швидкість навчання, тобто вплив кожного класифікатора на загальний прогноз. Словник параметрів містить списки можливих значень для кожного гіперпараметру.

- `('QuadraticDiscriminantAnalysis', QuadraticDiscriminantAnalysis(), {})` - це шостий і останній елемент списку models, який представляє модель QuadraticDiscriminantAnalysis. Представляє собою дискримінантний аналіз, який шукає квадратичну функцію для розділення класів даних. Він не має гіперпараметрів, тому словник параметрів порожній.

- `]` - закриття списку моделей для класифікації разом з їх гіперпараметрами.

- `estimators = [(name, model) for name, model, _ in models]` - це створює список estimators за допомогою спискового включення (list comprehension) із бібліотеки Python. Цей список містить кортежі з двох елементів: назви та класу моделі для кожної моделі зі списку models. Третій елемент кортежу (словник параметрів) ігнорується за допомогою символу «_».

- `voting_clf = VotingClassifier(estimators=estimators, voting='soft')` - створення об'єкту voting_clf, який є екземпляром класу VotingClassifier із бібліотеки sklearn. Це ансамблевий метод, який об'єднує прогнози декількох моделей за допомогою голосування. Параметр estimators задає список моделей, які використовуються для голосування. Параметр voting задає тип голосування: жорстке (hard) або м'яке (soft). Жорстке голосування використовує більшість голосів для визначення класу. М'яке голосування використовує середню ймовірність для визначення класу.

- `voting_clf.fit(X_train, y_train)` - навчає об'єкт voting_clf на тренувальних даних за допомогою методу fit із бібліотеки sklearn, що дозволяє побудувати ансамбль моделей та визначити їхні ваги для голосування.

- `voting_clf.fit(X_train, y_train)` - повторно навчає об'єкт voting_clf на тренувальних даних за допомогою методу fit із бібліотеки sklearn.

- `y_pred = voting_clf.predict(X_test)` - прогнозування класу для тестових даних за допомогою об'єкта voting_clf та методу predict із бібліотеки sklearn. Це дозволяє отримати результат ансамблевого голосування для кожного прикладу з тестових даних та зберегти його в змінну y_pred.

- `threshold = 0.8` - встановлення порогу для невизначеності в класифікації. Це значення від 0 до 1, яке визначає, наскільки модель має бути впевнена у своїй класифікації. Якщо максимальна ймовірність класифікації менша за поріг, то модель вважається невизначеною. В цьому коді порог дорівнює 0.8, тобто модель має бути впевнена у своїй класифікації на 80% або більше.

- `cv_results_dict = {}` - створення порожнього словника cv_results_dict із бібліотеки Python. Цей словник буде використовуватися для збереження результатів крос-валідації для кожної моделі.

- `metrics = {}` - створення порожнього словника metrics із бібліотеки Python. Цей словник буде використовуватися для збереження метрик продуктивності для кожної моделі.

- `for name, model, params in models:` - початок циклу for із бібліотеки Python, який проходить по кожному елементу списку models. Кожен елемент списку models є кортежем із трьох елементів: назви, класу та словника параметрів моделі. Цикл for розпаковує кожен елемент списку models у три змінні: name, model та params. Цикл for виконує наступний блок коду для кожної моделі.

- `filename = f'{name}.pkl'` - створення рядка filename за допомогою форматованого рядка (f-string) із бібліотеки Python. Рядок filename складається з

назви моделі та розширення .pkl. Цей рядок буде використовуватися для збереження моделі до файлу після навчання.

- `model.fit(X_train, y_train)` - навчання моделі на тренувальних даних за допомогою методу fit із бібліотеки sklearn. Це дозволяє побудувати модель і визначити її параметри на основі даних.

- `y_pred = model.predict(X_test)` - це прогнозування класу для тестових даних за допомогою моделі та методу predict із бібліотеки sklearn. Це дозволяє отримати результат класифікації для кожного прикладу з тестових даних і зберегти його в змінну y_pred.

- `accuracy = accuracy_score(y_test, y_pred)` - це розрахунок точності (accuracy) моделі за допомогою функції accuracy_score.

- `uncertain_percentage = uncertain_count / len(y_test) * 100` - це розрахунок відсотка випадків, коли модель не впевнена у своїй класифікації за допомогою арифметичних операцій / та * з бібліотеки Python. Відсоток обчислюється як кількість невизначених випадків, поділена на загальну кількість тестових даних, помножена на 100. Це дозволяє отримати число у вигляді відсотка та зберегти його в змінну uncertain_percentage.

- `print(f'Кількість даних, в класифікації яких модель {name} не впевнена: {uncertain_count}'` - це виведення на екран кількості даних, у класифікації яких модель не впевнена за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі та кількість невизначених випадків.

- `f' ({uncertain_percentage:.2f}%)')` - це продовження форматованого рядка (f-string), який містить відсоток даних, у класифікації яких модель не впевнена. Вираз

{uncertain_percentage:.2f}% означає, що змінна uncertain_percentage буде виведена у форматі з плаваючою комою (float) із двома знаками після коми (.2f) та символом %.

- `else:` - це початок альтернативного блоку коду для випадку, коли модель не має атрибуту predict_proba. Цей блок коду буде виконаний, якщо умова if не справджується.

- `print(f'Модель {name} не підтримує метод predict_proba')` - це вивід на екран повідомлення про те, що модель не підтримує метод predict_proba за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі.

- `recall = recall_score(y_test, y_pred, average='macro')` - це розрахунок повноти (recall) моделі за допомогою функції recall_score із бібліотеки sklearn. Параметр average='macro' означає, що повнота буде обчислена для кожного класу окремо та узяте середнє арифметичне. Це дозволяє отримати загальну оцінку продуктивності моделі для всіх класів і зберегти її в змінну recall.

- `precision = precision_score(y_test, y_pred, average='macro')` - розраховує точність (precision) моделі за допомогою функції precision_score з бібліотеки sklearn. Точність - це метрика, яка вимірює, скільки з прикладів, які модель класифікувала як позитивні, справді є позитивними. Параметр average='macro' означає те саме, що й для повноти. Це дозволяє отримати загальну оцінку продуктивності моделі для всіх класів і зберегти її в змінну precision.

- `f1 = f1_score(y_test, y_pred, average='macro')` - озраховує F1-середнє (F1-score) моделі за допомогою функції f1_score з бібліотеки sklearn. F1-середнє - це метрика, яка вимірює гармонійне середнє між повнотою та точністю. Це допомагає оцінити баланс між цими двома аспектами продуктивності моделі. Параметр

average='macro' означає те саме, що й для повноти та точності. Це дозволяє отримати загальну оцінку продуктивності моделі для всіх класів і зберегти її в змінну f1.

- `print(f'Повнота (recall) для моделі {name}: {recall}')` - виводить на екран повноту моделі за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі та значення повноти.

- `print(f'Точність (precision) для моделі {name}: {precision}')` - виводить на екран точність моделі за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі та значення точності.

- `print(f'F1-середнє для моделі {name}: {f1}')` - виводить на екран F1-середнє моделі за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі та значення F1-середнього.

- `metrics[name] = {` - створює новий елемент у словнику metrics за допомогою оператора присвоєння (=) з бібліотеки Python. Ключем елемента є назва моделі (name), а значенням є інший словник, який містить метрики продуктивності моделі. Цей рядок починає новий словник за допомогою символу {.

- `'Accuracy': accuracy,` - це перший елемент нового словника, який містить пару ключ-значення. Ключем є рядок 'Accuracy', а значенням є змінна accuracy, яка містить точність моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'Recall': recall,` - це другий елемент нового словника, який містить пару ключ-значення. Ключем є рядок 'Recall', а значенням є змінна recall, яка містить повноту

моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'Precision': precision,` - це третій елемент нового словника, який містить пару ключ-значення. Ключем є рядок 'Precision', а значенням є змінна precision, яка містить точність моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'F1-score': f1,` - це четвертий і останній елемент нового словника, який містить пару ключ-значення. Ключем є рядок 'F1-score', а значенням є змінна f1, яка містить F1-середнє моделі. Цей рядок закінчується комою (,), але це не обов'язково.

- `}` - закриває новий словник за допомогою символу }. Цим самим закривається новий елемент у словнику metrics.

- `cm = confusion_matrix(y_test, y_pred)` - розрахунок матриці помилок за допомогою функції confusion_matrix із бібліотеки sklearn. Функція приймає два аргументи: y_test та y_pred, які є реальними та прогнозованими класами для тестових даних. Функція повертає матрицю помилок у вигляді масиву з бібліотеки numpy. Цей масив зберігається у змінну cm.

- `print(f'Матриця помилок для моделі {name}:')` - це вивід на екран назви моделі та слово 'Матриця помилок' за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі.

- `print(cm)` - це вивід на екран матриці помилок за допомогою функції print із бібліотеки Python. Функція print використовує змінну cm, яка містить матрицю помилок у вигляді масиву.

- `cr = classification_report(y_test, y_pred, digits=5)` - звіт класифікації за допомогою функції classification_report із бібліотеки sklearn. Функція приймає три аргументи: y_test та y_pred, які є реальними та прогнозованими класами для тестових даних, та digits, який визначає кількість знаків після коми для метрик. Функція повертає звіт класифікації у вигляді рядка з бібліотеки Python. Цей рядок зберігається у змінну cr.

- `print(f'Звіт класифікації для моделі {name}:')` - це вивід на екран назви моделі та слова 'Звіт класифікації' за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі.

- `print(cr)` - це виводить на екран звіт класифікації за допомогою функції print з бібліотеки Python. Функція print використовує змінну cr, яка містить звіт класифікації у вигляді рядка.

- `scoring_metrics = {'accuracy': make_scorer(accuracy_score),` - це створення словника scoring_metrics за допомогою оператора присвоєння (=) з бібліотеки Python. Цей словник містить пари ключ-значення, де ключем є назва метрики, а значенням є функція, яка розраховує цю метрику. Цей рядок починає новий словник за допомогою символу { та створює перший елемент словника. Перший елемент словника має ключ 'accuracy' та значення make_scorer(accuracy_score). Функція make_scorer із бібліотеки sklearn дозволяє створити об'єкт scorer, який можна використовувати для перехресної перевірки. Функція приймає іншу функцію як аргумент - у цьому випадку accuracy_score, яка розраховує точність моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'precision': make_scorer(precision_score),` - це другий елемент нового словника, який має ключ 'precision' та значення make_scorer(precision_score). Функція precision_score розраховує точність моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'recall': make_scorer(recall_score),` - це третій елемент нового словника, який має ключ 'recall' та значення make_scorer(recall_score). Функція recall_score розраховує повноту моделі. Цей рядок закінчується комою (,), що означає, що наступає інший елемент словника.

- `'f1': make_scorer(f1_score)}` - це четвертий і останній елемент нового словника, який має ключ 'f1' та значення make_scorer(f1_score). Функція f1_score розраховує F1-середнє моделі. Цей рядок закінчується комою (,), але це не обов'язково. Цей рядок також закриває новий словник за допомогою символу }.

- `cv_results = cross_validate(model, X_train, y_train, scoring=scoring_metrics)` - це виконання перехресної перевірки за допомогою функції cross_validate з бібліотеки sklearn. Функція приймає чотири аргументи: model, X_train, y_train та scoring. Перший аргумент model - це модель класифікації, яку потрібно оцінити. Другий та третій аргументи X_train та y_train - це тренувальні дані та їх класи. Четвертий аргумент scoring - це словник scoring_metrics, який містить метрики для оцінки продуктивності моделі. Функція повертає словник cv_results, який містить результати перехресної перевірки для кожної метрики.

- `cv_results_dict[name] = cv_results` - це створення нового елементу словнику cv_results_dict за допомогою оператора присвоєння (=) з бібліотеки Python. Ключем елемента є назва моделі (name), а значенням є словник cv_results, який містить результати перехресної перевірки для моделі. Цим самим зберігаються результати перехресної перевірки для поточної моделі в словнику cv_results_dict.

- `print(f'{name} cross-validation metrics:')` - це вивід на екран назви моделі та слів 'cross-validation metrics' за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву моделі.

- `for metric in scoring_metrics.keys():` - це початок циклу for із бібліотеки Python, який проходить по кожному ключу у словнику scoring_metrics. Кожен ключ у словнику scoring_metrics є назвою метрики. Цикл for розпаковує кожен ключ у змінну metric. Цикл for виконує наступний блок коду для кожної метрики.

- `mean_score = cv_results[f'test_{metric}'].mean()` - розраховує середнє значення метрики за допомогою методу mean із бібліотеки numpy. Метод mean приймає масив чисел і повертає їхнє середнє арифметичне. Масив чисел отримується зі словника cv_results за допомогою форматованого рядка (f-string), який містить назву метрики. Ключ f'test_{metric}' означає, що зі словника cv_results вибирається масив, який містить результати перехресної перевірки для поточної метрики. Це дозволяє отримати середнє значення метрики та зберегти його у змінну mean_score.

- `print(f' Mean {metric}: {mean_score}')` - це вивід на екран назви та середнбого значення метрики за допомогою функції print із бібліотеки Python. Функція print використовує форматований рядок (f-string), який містить назву та середнє значення метрики.

- `cv_results_df = pd.DataFrame(` - це створення нового датафрейму за допомогою функції DataFrame з бібліотеки pandas. Функція приймає один аргумент - дані для датафрейму. Цей рядок починає новий датафрейм за допомогою символу (.

- `{name: {f'Mean test {metric}': cv_results[f'test_{metric}'].mean() for metric in scoring_metrics.keys()} for` - це створенн словника за допомогою словникового включення (dictionary comprehension) з бібліотеки Python. Цей словник буде використовуватися як дані для датафрейму. Словникове включення - це спосіб створити новий словник за допомогою одного рядка коду. Словникове включення складається з двох частин: пари ключ-значення та ітератора. Пара ключ-значення

визначає, як будуть створюватися елементи нового словника. Ітератор визначає, по якому набору даних будуть проходитися пари ключ-значення. У цьому випадку парою ключ-значення є name: {f'Mean test {metric}': cv_results[f'test_{metric}'].mean() for metric in scoring_metrics.keys()}, а ітератором є for name, cv_results in cv_results_dict.items(). Це означає, що новий словник буде створюватися за допомогою наступної логіки: для кожної пари назви моделі та результатів перехресної перевірки зі словника cv_results_dict, створити новий елемент у новому словнику, де ключем буде назва моделі, а значенням буде інший словник, який містить середні значення метрик для моделі. Цей рядок закінчується символом :, що означає, що наступає інша частина словникового включення.

- `name, cv_results in cv_results_dict.items()}` - це ітератор для словникового включення, який проходить по кожному елементу словника cv_results_dict. Кожен елемент словника cv_results_dict є парою ключ-значення, де ключем є назва моделі, а значенням є результати перехресної перевірки для моделі. Ітератор розпаковує кожну пару ключ-значення у дві змінні: name та cv_results. Цей рядок закриває новий словник за допомогою символу }.

- `)` - закриває новий датафрейм за допомогою символу). Цим самим закривається виклик функції DataFrame. Новий датафрейм зберігається у змінну cv_results_df.

- `sns.heatmap(cv_results_df.T, cmap='Purples', annot=True, fmt='.9f')` - це створює та виводить на екран теплову карту за допомогою функції heatmap з бібліотеки seaborn. Функція heatmap приймає чотири аргументи: data, cmap, annot та fmt. Перший аргумент data - це дані для теплової карти. У цьому випадку data = cv_results_df.T, що означає, що дані беруться з датафрейму cv_results_df, який транспонується за допомогою методу T з бібліотеки pandas. Транспонування означає, що рядки та стовпці датафрейму обмінюються місцями. Це дозволяє показати назви моделей як рядки та назви метрик як стовпці на тепловій карті.

Другий аргумент cmap - це колірна схема для теплової карти. У цьому випадку cmap = 'Purples', що означає, що теплова карта буде використовувати різні відтінки фіолетового кольору для позначення числових значень. Третій аргумент annot - це логічний параметр, який визначає, чи будуть показані числові значення на тепловій карті. У цьому випадку annot = True, що означає, що числові значення будуть показані на тепловій карті. Четвертий аргумент - `fmt - це формат для числових значень на тепловій карті. У цьому випадку fmt = '.9f', що означає, що числові значення будуть виведені з дев'ятьма знаками після коми (.9f). Це дозволяє показати дуже малі різниці між результатами перехресної перевірки для різних моделей та метрик.

- `plt.title('Cross-validation metrics for different classifiers')` - це додавання заголовку до теплової карти за допомогою функції title з бібліотеки matplotlib. Функція title приймає один аргумент - рядок з бібліотеки Python, який містить текст заголовка. У цьому випадку текст заголовка - 'Cross-validation metrics for different classifiers', що означає 'Метрики перехресної перевірки для різних класифікаторів'.

- `plt.show()` - виводить теплову карту на екрані за допомогою функції show із бібліотеки matplotlib. Функція show не приймає жодних аргументів.

Також був реалізований метод виявлення червоних прапорців. Програма реагувала на підозрілі елементи листів, такі як:

1) Запит на терміновий переказ грошей (кодові слова – urgent, asap, important, immediate та інші);

2) Посада відправника (CEO, COO, president, founder), адже зловмисники часто представляються представниками верхівок окремих компаній, щоб заволодіти довірою, а в результаті й конфіденційними даними співробітників;

3) Перевірка домену відправника (gmail.com, hotmail.com, outlook.com), найчастіше шахраї використовують пошти, створені на доменах загального користування, до котриї і належать наведені вище адреси.

При знаходженні якогось із цих тригерів з’являється відповідне повідомлення, котре каже, що повідомлення є підозрілим, оскільки містить червоний прапорець певного типу.

Для мультиноміального наївного Баєса в якості експерименту була реалізована можливість перегляду підозрілих повідовлень користувачем у випадку, коли модель має середній рівень впевненості у своєму рішенні.

3.3 Аналіз результатів

Метрики - це числові показники, які вимірюють продуктивність моделі за різними аспектами.

Теплова карта - це графічне зображення даних у вигляді таблиці, де кольори відповідають числовим значенням. Теплова карта дозволяє візуалізувати та порівняти результати перехресної перевірки для різних моделей та метрик.

Матриця помилок - це таблиця, яка показує, скільки прикладів кожного класу було правильно та неправильно класифіковано моделлю.

Звіт класифікації - це текстовий документ, який показує різні метрики продуктивності моделі для кожного класу.

Recall - це метрика, яка вимірює, скільки реально позитивних прикладів було правильно класифіковано моделлю.

Accuracy – це метрика, яка вимірює, скільки правильно класифікованих спостережень прийшлося на суму всіх спостережень.

Precision – в це метрика, яка вимірює, скільки позитивних спостережень прийшлося на суму істинно позитивних і хибно позитивних спостережень.

F1-score – це гармонійне середнє між Precision і Recall. Використовується для оцінення балансу між Precision і Recall.

Перехресна перевірка - це метод оцінки продуктивності моделі, який ділить дані на декілька частин (фолдів) та використовує одну частину для тестування, а решту - для навчання. Цей процес повторюється для кожної частини даних і усереднюється.

Проаналізувавши вплив усіх методів попередньої обробки даних на результати, що показали різні моделі машинного навчання, а також датасети з синтетичними електронними листами та без них, можна встановити наступні зміни в порівнянні з роботою моделей машинного навчання бнз попередньої обробки тексту (Рисунок 3.3.1, Рисунок 3.3.2):

- Приведення тексту до нижнього регістру (Рисунок 3.3.3, Рисунок 3.3.4, Таблиця 3.3.1):
  - Гаусівський наївний Байєс – незначний вплив;
  - Мультиноміальний наївний Байєс – значне покращення позитивних класифікацій;
  - Дерево рішень – зменшення кількості хибно позитивних класифікацій (незначне);
  - Метод опорних векторів – робота моделі значно покращилась;
  - Адаптивний бустинг – незначний вплив;
  - Квадратичний дискримінантний аналіз – незначні зміни.
- Стематизація (Рисунок 3.3.5, Рисунок 3.3.6, Таблиця 3.3.2):
  - Гаусівський наївний Байєс – значний позитивний вплив на кількість хибно позитивних класифікацій;

- Мультиноміальний наївний Байєс – значне загальне погіршення роботи моделі;
- Дерево рішень – погіршення позитивної класифікації;
- Метод опорних векторів – зменшення кількості хибно позитивних класифікацій за рахунок загальної кількості позитивних класифікацій;
- Адаптивний бустинг – незначні зміни;
- Квадратичний дискримінантний аналіз – збільшилась загальна кількість негативних класифікацій (в тому числі й хибно негативних).

• Лематайзинг (Рисунок 3.3.7, Рисунок 3.3.8, Таблиця 3.3.3):

- Гаусівський наївний Байєс – збільшилась загальна кількість негативних класифікацій (в тому числі й хибно негативних);
- Мультиноміальний наївний Байєс – загальне погіршення роботи моделі;
- Дерево рішень – незначні зміни;
- Метод опорних векторів – більша кількість позитивних класифікацій (в тому числі й хибних);
- Адаптивний бустинг – змін не відбулося;
- Квадратичний дискримінантний аналіз – зменшення загальної кількості позитивних реакцій, що призвело до збільшення кількості хибно негативних.

• Усі попередні обробки (Рисунок 3.3.9, Рисунок 3.3.10, Таблиця 3.3.4):

- Гаусівський наївний Байєс – зменшення загальної кількості позитивних реакцій, що призвело до збільшення кількості хибно негативних;
- Мультиноміальний наївний Байєс – погіршення роботи моделі;
- Дерево рішень – незначні зміни;
- Метод опорних векторів – збільшилась кількість позитивних реакцій, що призвело до зменшення кількості хибно негативних за рахунок збільшення хибно позитивних;
- Адаптивний бустинг – незначні зміни;

- Квадратичний дискримінантний аналіз – збільшилась кількість хибно позитивних реакцій за рахунок зменшення хибно негативних.

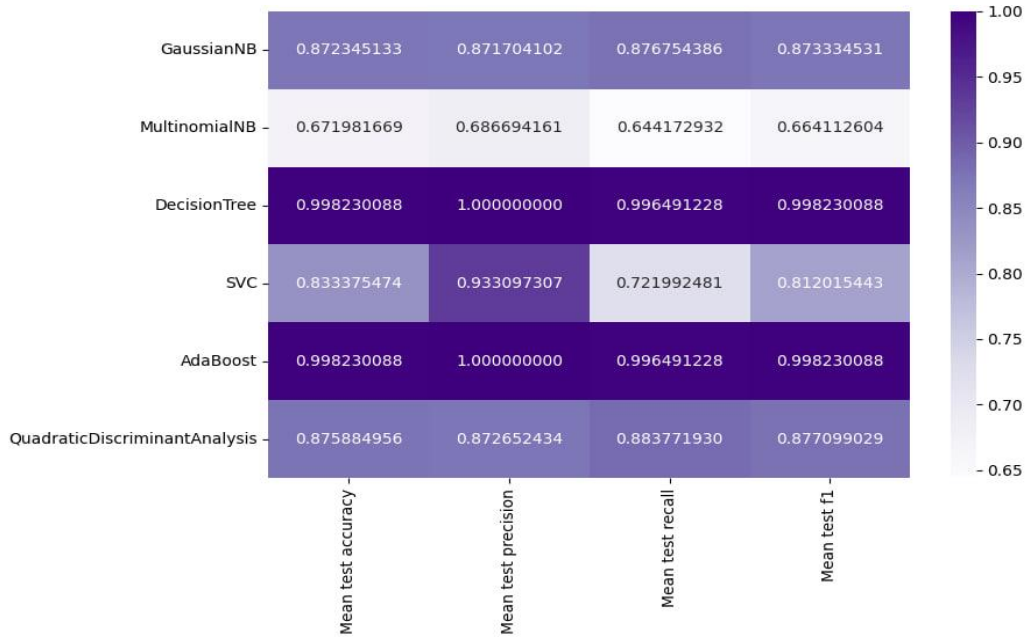


Рисунок 3.3.1 – Візуалізація роботи різних методів на датасеті без додавання синтетичних листів

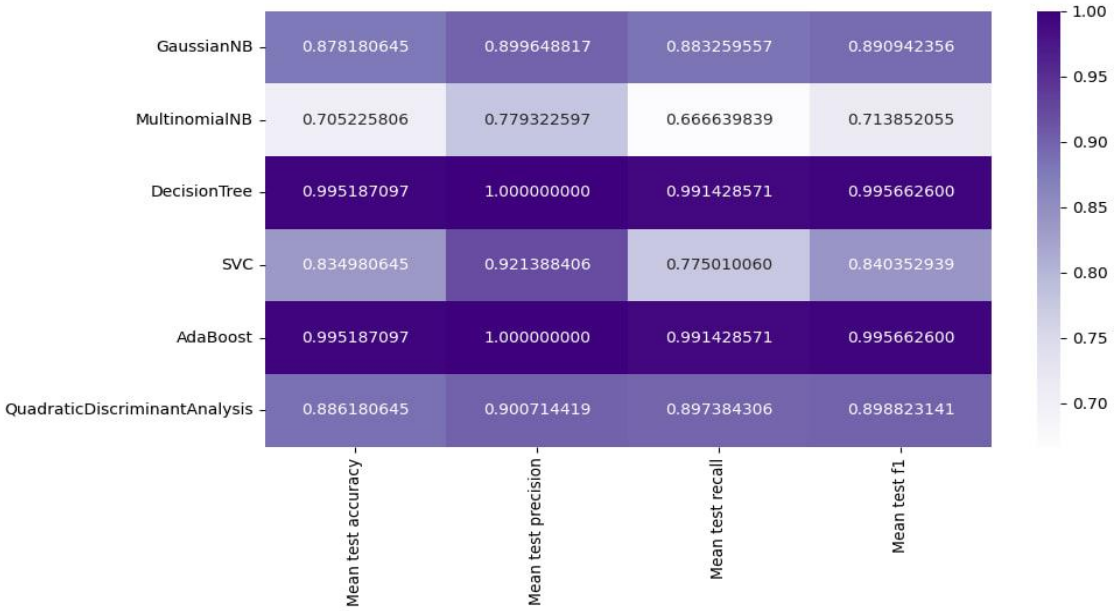


Рисунок 3.3.2 – Візуалізація роботи різних методів на датасеті з додаванням синтетичних листів

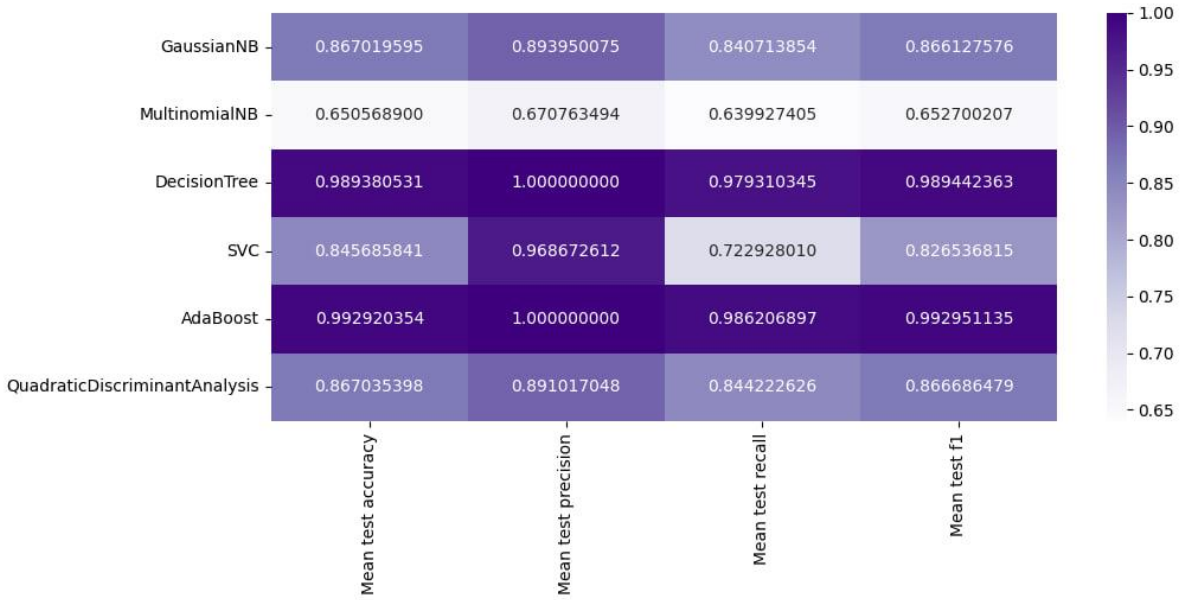


Рисунок 3.3.3 – Візуалізація роботи різних методів із приведенням до нижнього регістру на датасеті без додавання синтетичних листів

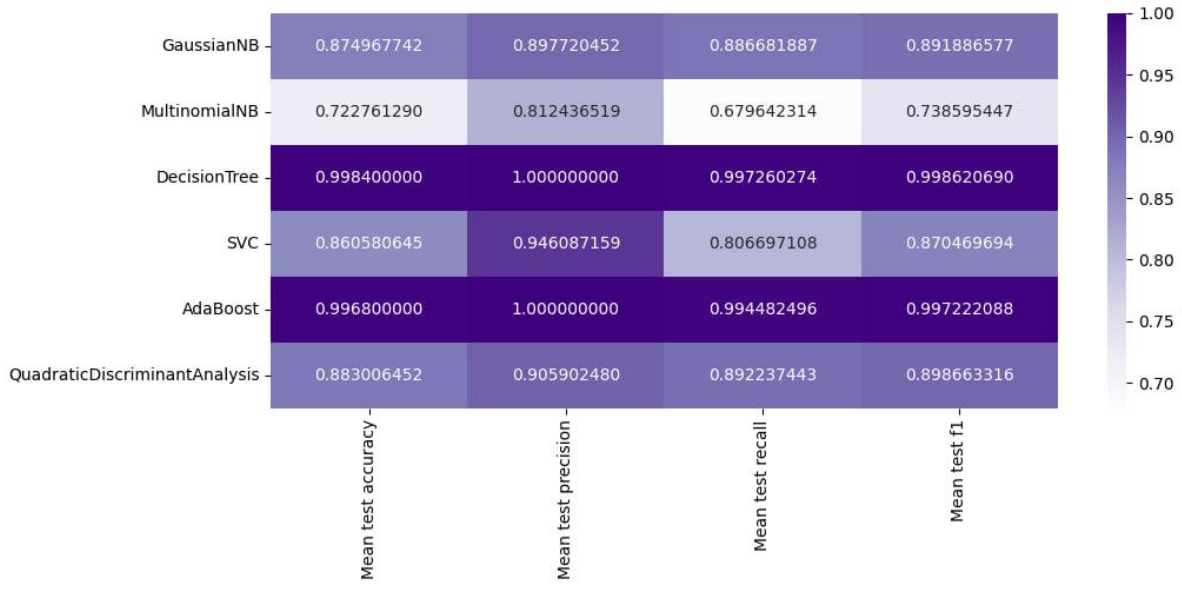


Рисунок 3.3.4 – Візуалізація роботи різних методів із приведенням до нижнього регістру на датасеті з додаванням синтетичних листів

Таблиця 3.3.1

	Точність (accuracy)	Точність (precision)	Повнота	F1 - оцінка	Висновок
Наївний Байєс (гаусівський)	Зменшилась на 0.3%	Збільшилась на 0.2%	Збільшилась на 0.2%	Збільшилась на 0.1%	Загалом збільшення повноти при зменшенні точності (accuracy) можна пояснити більш агресивною позитивною класифікацією, але метрики змінились незначно, що можна прирівняти до похибки
Наївний Байєс (мультиноміальний)	Збільшилась на 1.7%	Збільшилась на 3.4%	Збільшилась на 1.3%	Збільшилась на 2%	Найбільш інтенсивне збільшення precision означає, що модель стала краще правильно робити позитивні класифікації
Дерево рішень	Збільшилась на 0.3%	Не змінилась	Збільшилась на 0.8%	Збільшилась на 0.3%	Збільшення повноти призвело до зменшення хибно позитивних класифікацій (так як precision не змінилась)
Метод опорних векторів	Збільшилась на 2.5%	Збільшилась на 2.5%	Збільшилась на 3.1%	Збільшилась на 3%	Збільшення усіх метрик демонструє покращення роботи моделі в усіх сенсах
Адаптивний бустінг	Збільшилась на 0.1%	Не змінилась	Збільшилась на 0.3%	Збільшилась на 0.2%	Збільшення повноти зменшення хибно позитивних класифікацій (так як precision не змінилась), але зміна надто мала, аби враховувати це як позитивний результат
Квадратичний дискримінантний аналіз	Зменшилась на 0.3%	Збільшилась на 0.5%	Збільшилась на 0.5%	Не змінилась	Збільшення повноти при зменшенні accuracy означає, що модель стала гірше працювати з негативною класифікацією

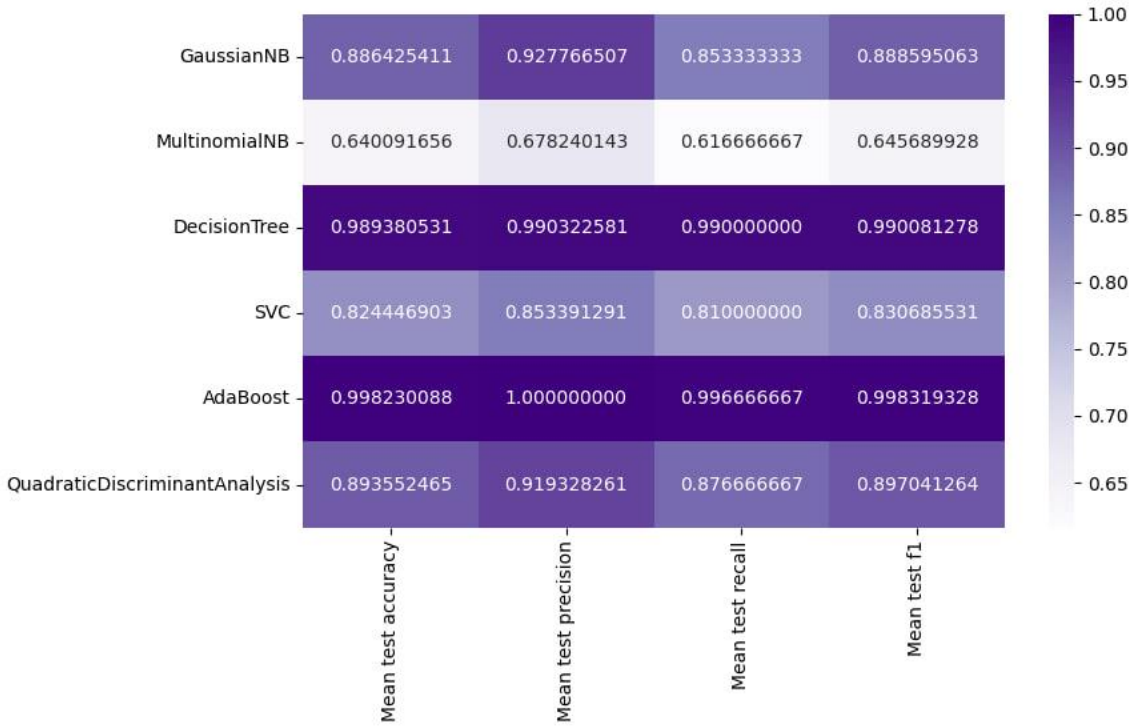


Рисунок 3.3.5 – Візуалізація роботи різних методів зі стемінгои на датасеті без додавання синтетичних листів

Рисунок 3.3.6 – Візуалізація роботи різних методів зі стемінгом на датасеті з додаванням синтетичних листів

Таблиця 3.3.2

	Точність (accuracy)	Точність (precision)	Повнота	F1 - оцінка	Висновок
Наївний Байєс (гаусівський)	Збільшилась на 0.8%	Збільшилась на 2.8%	Збільшилась на 3%	Збільшилась на 0.2%	Так як precision збільшилась на багато більше, ніж accuracy, то можно зробити висновок, що зменшилась кількість хибно позитивних класифікацій
Наївний Байєс (мультиноміальний)	Зменшилась на 5.5%	Зменшилась на 10%	Зменшилась на 5%	Зменшилась на 6.8%	Усі метрики погіршились, отже модель стала працювати значно гірше
Дерево рішень	Зменшилась на 0.6%	Зменшилась на 1%	Зменшилась на 0.1%	Зменшилась на 0.5%	Усі метрики погіршились, але значно лише precision, отже погіршилась позитивна класифікація
Метод опорних векторів	Зменшилась на 0.1%	Зменшилась на 6.8%	Збільшилась на 3.5%	Зменшилась на 1%	Так як precision збільшилась на багато більше, ніж accuracy, то можно зробити висновок, що зменшилась кількість хибно позитивних класифікацій
Адаптивний бустінг	Збільшилась на 0.3%	Не змінилась	Збільшилась на 0.5%	Збільшилась на 0.3%	Збільшилась повнота, отже хибно позитивних класифікацій стало менше (враховуючи стагнацію precision) отже зменшилась кількість хибно негативних класифікацій
Квадратичний дискримінантний аналіз	Збільшилась на 0.3%	Збільшилась на 1.9%	Зменшилась на 2.1%	Зменшилась на 0.1%	Так як обидві точності збільшились (precision значно підвищилась), але повнота зменшилась, то можна зробити висновок, що збільшилась кількість хибно негативних класифікацій

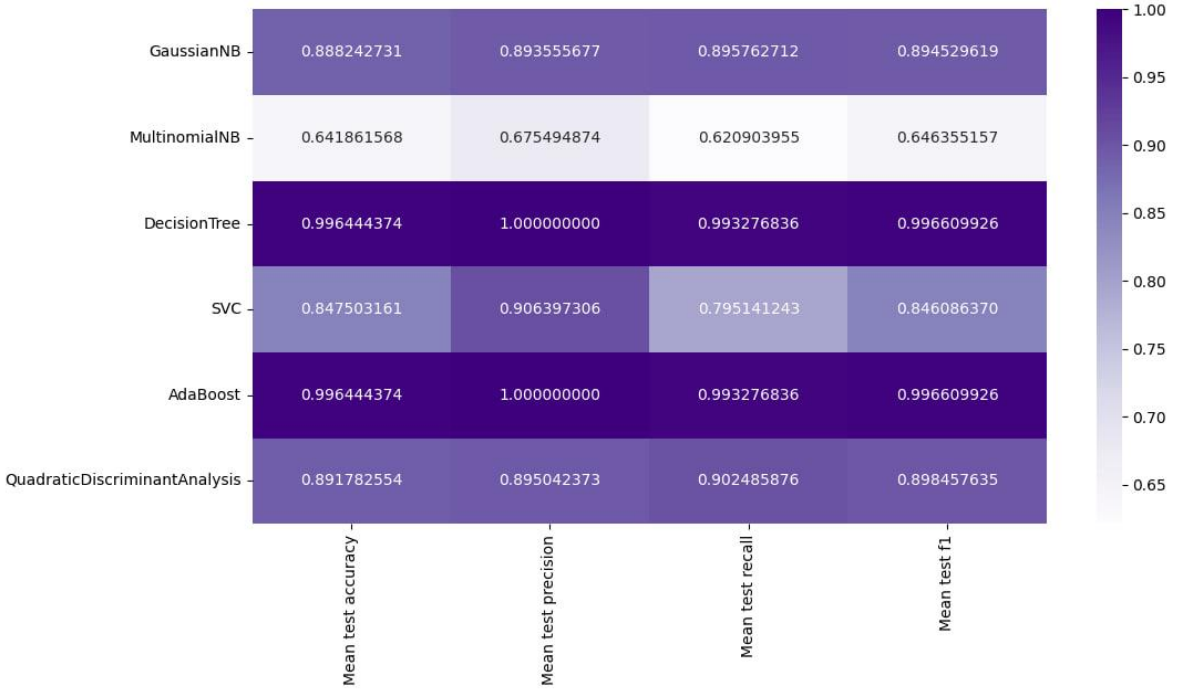


Рисунок 3.3.7 – Візуалізація роботи різних методів із лематайзингом на датасеті без додавання синтетичних листів

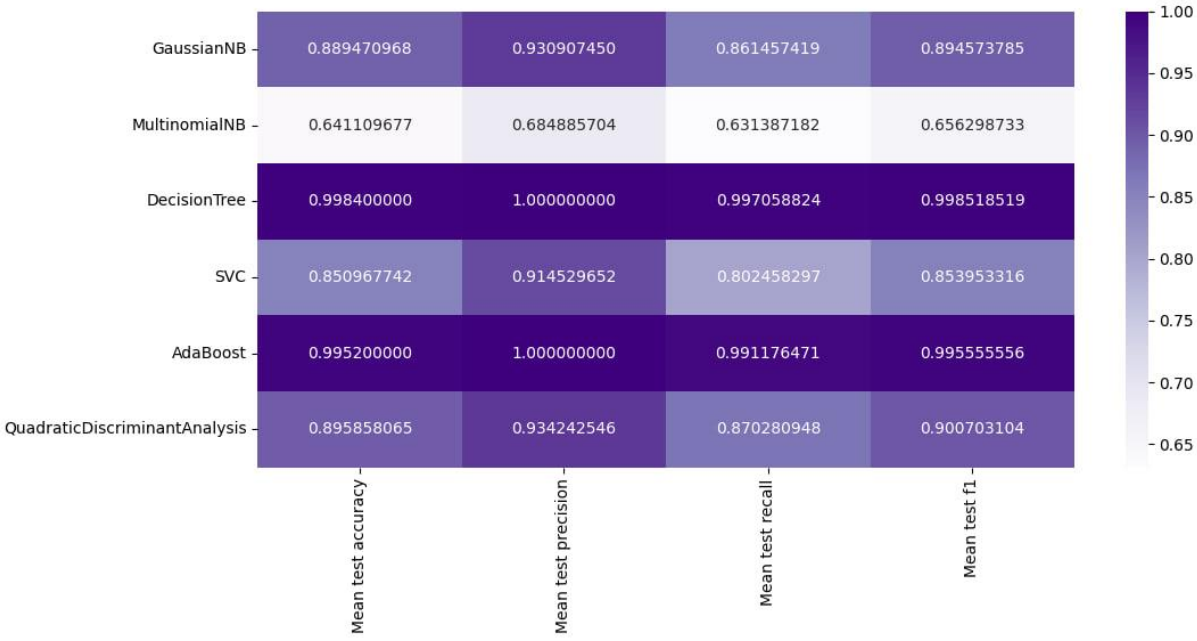


Рисунок 3.3.8 – Візуалізація роботи різних методів із лематайзингом на датасеті з додаванням синтетичних листів

Таблиця 3.3.3

	Точність (accuracy)	Точність (precision)	Повнота	F1 - оцінка	Висновок
Наївний Байєс (гаусівський)	Збільшилась на 1.1%	Збільшилась на 3.1%	Зменшилась на 2%	Збільшилась на 0.4%	Так як обидві точності збільшились (precision значно підвищилась), але повнота зменшилась, то можна зробити висновок, що збільшилась кількість хибно негативних класифікацій
Наївний Байєс (мультиноміальний)	Зменшилась на 6%	Зменшилась на 9%	Зменшилась на 3.5%	Зменшилась на 6%	Усі метрики значно погіршились – модель стала працювати гірше
Дерево рішень	Збільшилась на 0.3%	Не змінилась	Збільшилась на 0.6%	Збільшилась на 0.3%	Повнота збільшилась при незмінній precision, отже зменшилась кількість хибно негативних класифікацій, але ця зміна незначна, що може вказувати на похибку
Метод опорних векторів	Збільшилась на 1.5%	Зменшилась на 1%	Збільшилась на 2.5%	Збільшилась на 1.4%	Збільшилась повнота та accuracy при зменшенні precision, що вказує на збільшення хибно позитивних класифікацій, але менш інтенсивно, чим зменшення кількості хибно негативних
Адаптивний бустінг	Не змінилась	Не змінилась	Не змінилась	Не змінилась	Лематайзер не вплинув на цю модель
Квадратичний дискримінантний аналіз	Збільшилась на 1%	Збільшилась на 3.4%	Зменшилась на 2.9%	Збільшилась на 0.2%	Precision збільшилась при зменшенні повноти, отже в моделі стало менше хибно позитивних класифікацій, але стало більше хибно негативних

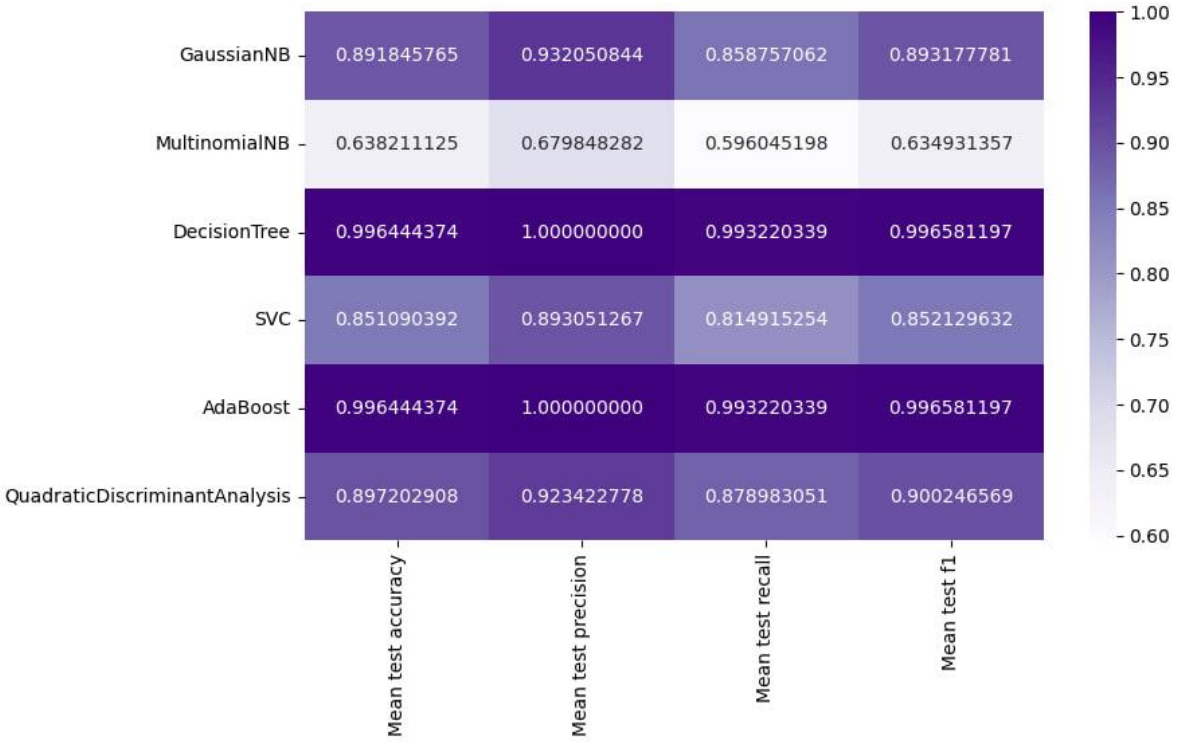


Рисунок 3.3.9 – Візуалізація роботи різних методів із використанням усіх засобів попередньої обробки тексту на датасеті без додавання синтетичних листів

Рисунок 3.3.10 – Візуалізація роботи різних методів із використанням усіх засобів попередньої обробки тексту на датасеті з додаванням синтетичних листів

Таблиця 3.3.4

	Точність (accuracy)	Точність (precision)	Повнота	F1 - оцінка	Висновок
Наївний Байєс (гаусівський)	Зменшилась на 1%	Збільшилась на 1.5%	Зменшилась на 5%	Зменшилась на 1.8%	Усі метрики погіршились крім precision, що вказує на зменшення кількості хибно позитивних класифікацій при збільшенні хибно негативних
Наївний Байєс (мультиноміальний)	Зменшилась на 7%	Зменшилась на 10%	Зменшилась на 4%	Зменшилась на 6.3%	Усі метрики значно погіршились, що вказує на погіршення роботи моделі
Дерево рішень	Збільшилась на 0.1%	Не змінилась	Збільшилась на 0.3%	Збільшилась на 0.2%	Усі метрики крім повноти покращились, отже в моделі стало менше хибно негативних класифікацій, але ця зміна незначна, що може бути похибкою
Метод опорних векторів	Збільшилась на 0.5%	Зменшилась на 1.6%	Збільшилась на 1.3%	Збільшилась на 0.2%	Повнота збільшилась при зменшенні precision, що зменшення кількості хибно негативних класифікацій за рахунок збільшення хибно позитивних
Адаптивний бустінг	Збільшилась на 0.1%	Не змінилась	Збільшилась на 0.3%	Збільшилась на 0.2%	Усі метрики крім повноти покращились, отже в моделі стало менше хибно негативних класифікацій, але ця зміна незначна, що може бути похибкою
Квадратичний дискримінантний аналіз	Зменшилась на 0.1%	Збільшилась на 2%	Зменшилась на 5%	Зменшилась на 1.7%	Precision та повнота значно зменшились, що вказує на збільшення кількості хибно позитивних та хибно негативних реакцій

Висновки до розділу 3

Отже, приведення тексту до нижнього регістру найкраще вплинуло на мультиноміальний наївний Байєс, що вказує на доцільність використання такої комбінації. Стемінг добре вплинув на гаусівський наївний Байєс. Лематайзер покращив роботу модель опорних векторів, що зробило її більш агресивною відносно позитивних класифікацій. Послідовна комбінація з усіх методів попередньої обробки даних зробила метод опорних векторів більш агресивним відносно позитивних класифікацій.

Важливо зазначити, що найкращі результати в усіх тестах показали дерево рішень та адаптивний бустинг, але всі методи попередньої обробки даних не спричинили значного впливу на покращення чи погіршення їхньої роботи. Додавання синтетичних листів у датасет також покращило роботу цих двох методів. Щодо решти методів машинного навчання, то якщо вибір пав на них, то необхідно розуміти цілі цього рішення, так як деякі комбінації спричиняють більшу кількість хибно позитивних класифікацій при зменшенні хибно негативних, що може бути важливо для деяких клієнтів, котрим байдуже на хибно позитивні рішення, але вони не хочуть бачити небажану пошту.

Методи МН показали себе як гарна альтернатива методу виявлення червоних прапорців.

ВИСНОВКИ

У ході виконання даної роботи було виконано всі поставленні завдання.

При детальному аналізі різних видів загроз і методів боротьби з ними виявилося, що існує широке поле для досліджень засобів протидії ВЕС із використанням методів машинного навчання. У ході експерименту та порівняльного аналізу було встановлено, що найкращі результати в класифікації імпосторів показали методи «дерево рішень» і «адаптивний бустинг», а використання засобів попередньої обробки тексту покращило їхню роботу незначною мірою, додавання в датасет синтетичних листів також спричинило підвищення ефективності пошуку ВЕС.

Результати, отримані в ході даного дослідження, можуть бути застосовані в різних сферах, адже кібер-злочинці не гребують ні маленькими компаніями, ні підприємствами державного значення. В умовах гібридної війни, котра зараз охопила нашу державу, треба з великою відповідальністю підходити до захисту інформації на всіх рівнях. Запропонований метод протидії імпосторам дозволить більш ефективно та менш витратно боротися з даним типом загроз, ніж його попередники.

Оскільки досліджень на тему протидії ВЕС за допомогою методів МН майже немає, дана робота може стати поштовхом для інших наукових досліджень у цьому напрямку.

У подальшому існують перспективи продовження даного дослідження та створення повноцінного продукту для ідентифікації та боротьби з різними видами кібер-загроз шляхом впровадження модифікацій в уже описані методи та розширення функціоналу. Маю наміри дослідити дані перпективи у своїй майбутній магістерській роботі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Microsoft-Security. What is Business Email Compromise (BEC)? — URL: https://www.micro soft.com/en-us/security/business/security101/what-isbusiness-email-compromise-bec.

2) Gmail-Help. Prevent mail to Gmail users from being blocked or sent to spam. — URL: https://suppor t.google.com/mail/answer/81126?hl=en.

3) Blackie A. Email Authenticity 101: DKIM, DMARC, and SPF. — URL: https://support.google.com /mail/answer/81126?hl=en.

4) microsoft.com. Business Email Compromise Part One. — URL: https : / / techcommunity . micros oft.com/t5/microsoft-defender-for-office/b usiness-emailuncompromised-part-one/ba-p /2159900.

5) Security-News. Red Flags: How to Spot a Business Email Compromise Scam. — URL: https://www.t rendmicro.com/vinfo/us/security/news/cyber crime-anddigital-threats/how-to-spot-busi ness-email-scam.

6) FBI. Business Email Compromise. — URL: https: //www.fbi.gov/how-we-canhelp-you/safety-r esources/scams-and-safety/common-scamsandcrimes/business-email-compromise

7) Domingos, Pedro & Michael Pazzani (1997) «On the optimality of the simple Bayesian classifier under zero-one loss». Machine Learning, 29:103-137

8) Mozina M, Demsar J, Kattan M, & Zupan B. (2004). «Nomograms for Visualization of Naive Bayesian Classifier». In Proc. of PKDD-2004, pages 337— 348.

9) Liu, W.; Principe, J.; Haykin, S. (2010). Kernel Adaptive Filtering: A Comprehensive Introduction. Wiley.

10) Shawe-Taylor, J.; Cristianini, N. (2004). Kernel Methods for Pattern Analysis. Cambridge University Press.

11) Rokach, Lior; Maimon, O. (2008). Data mining with decision trees: theory and applications. World Scientific Pub Co Inc. ISBN 978-9812771711. 19

12) Підтримка векторної машини (SVM) у машинному навчанні - techukraine.net

13) 1.4. Support Vector Machines — scikit-learn 1.2.2 documentation

14) Freund, Yoav; Schapire, Robert E. (1995), "A desicion-theoretic [sic] generalization of on-line learning and an application to boosting", *Lecture Notes in Computer Science*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 23–37

15) "Linear & Quadratic Discriminant Analysis · UC Business Analytics R Programming Guide". uc-r.github.io. Retrieved 2020-03-29.

16) Internet Crime Report (FBI) – 2018 – https://www.ic3.gov/Media/PDF/AnnualReport/2018_IC3Report.pdf

ДОДАТКИ

Додаток А

Код

```
    # Імпорт необхідних бібліотек
import numpy as np
import pandas as pd
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score, make_scorer, \
    classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from sklearn.ensemble import AdaBoostClassifier, VotingClassifier

# Завантаження набору даних з csv-файлу та видалення рядків з
пропущеними значеннями
data = pd.read_csv('tas_dataset_n.csv')
data = data.dropna()


def notify_user(pipeline, X_test):
    # 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%
    should_sheck_confidence_idx = np.where(max_proba < 0.8)[0]
    # Прохід по індексах з низькою впевненістю

    for idx in should_sheck_confidence_idx:
        if pipeline.classes_[np.argmax(y_proba[idx])] == 1:
            print(f"{should_sheck_confidence_idx[idx]}>>>Система
виявила цей лист як підозрилий, рекомендуємо поставитись до нього з
обережністю.")
```

```
def machine_learning(data):
    # Застосування функції process_text до відповідних стовпців у
наборі даних
    data['mail'] = data['mail'].apply(process_text)
    data['user'] = data['user'].apply(process_text)
    data['s_address'] = data['s_address'].apply(process_text)
    data['s_name'] = data['s_name'].apply(process_text)

    # Кодування категоріальних змінних за допомогою LabelEncoder
    le = LabelEncoder()
    data['mail'] = le.fit_transform(data['mail'])
    data['user'] = le.fit_transform(data['user'])
    data['s_address'] = le.fit_transform(data['s_address'])
    data['s_name'] = le.fit_transform(data['s_name'])

    # Розділення набору даних на ознаки та цільову змінну, а потім на
навчальні та тестові набори.
    X = data[['mail', 'user', 's_address', 's_name']]
    y = data['class']
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4)

    # Визначення списку моделей для класифікації разом з їх
гіперпараметрами.
    models = [
        ('GaussianNB', GaussianNB(), {}),
        ('MultinomialNB', MultinomialNB(), {'alpha': [0.1, 0.5,
1.0]}),
        ('DecisionTree', DecisionTreeClassifier(), {'max_depth': [3,
5, 10], 'min_samples_split': [2, 5]}),
        ('SVC', SVC(probability=True), {'C': [0.1, 1.0], 'kernel':
['linear', 'rbf']}),
        ('AdaBoost', AdaBoostClassifier(), {'n_estimators': [50,
100], 'learning_rate': [0.1, 1.0]}),
        ('QuadraticDiscriminantAnalysis',
QuadraticDiscriminantAnalysis(), {})
    ]

    # Створення класифікатора голосування за допомогою визначених
моделей.
    estimators = [(name, model) for name, model, _ in models]
    voting_clf = VotingClassifier(estimators=estimators,
voting='soft')
    voting_clf.fit(X_train, y_train)
    voting_clf.fit(X_train, y_train)
    y_pred = voting_clf.predict(X_test)
    # Встановлення порогу для невизначеності у класифікації.
    threshold = 0.5
    cv_results_dict = {}
```

```
    metrics = {}
    # Навчання кожної моделі на тренувальних даних та оцінка їх
продуктивності на тестових даних.
    for name, model, params in models:
        filename = f'{name}.pkl'
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print(f'Точність {name}: {accuracy}')

        # Якщо модель підтримує метод predict_proba,
        # розрахуйте кількість випадків, коли модель не впевнена у
своїй класифікації.
        if hasattr(model, 'predict_proba'):
            proba = model.predict_proba(X_test)
            max_proba = proba.max(axis=1)
            uncertain_count = (max_proba < threshold).sum()
            uncertain_percentage = uncertain_count / len(y_test) *
100
            print(f'Кількість даних, в класифікації яких модель
{name} не впевнена: {uncertain_count}'
                  f' ({uncertain_percentage:.2f}%)')
        else:
            print(f'Модель {name} не підтримує метод predict_proba')

        if name == "MultinomialNB":
            notify_user(model, X_test)

        # Обчислення різних метрик
        recall = recall_score(y_test, y_pred, average='macro')
        precision = precision_score(y_test, y_pred, average='macro')
        f1 = f1_score(y_test, y_pred, average='macro')

        print(f'Повнота (recall) для моделі {name}: {recall}')
        print(f'Точність (precision) для моделі {name}: {precision}')
        print(f'F1-середнє для моделі {name}: {f1}')

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

        # Виведення матриці помилок та звіту класифікації для кожної
моделі
        cm = confusion_matrix(y_test, y_pred)
        print(f'Матриця помилок для моделі {name}:')
```

```
        print(cm)

        cr = classification_report(y_test, y_pred, digits=5)
        print(f'Звіт класифікації для моделі {name}:')
        print(cr)

        # 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(model, 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}')

        # 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, cmap='Purples', annot=True,
fmt='.9f')
    plt.show()


def detect_red_flags_mail(mail):
    red_flags = []
    # Перевірка термінового запиту на переказ коштів
    urgent_request_keywords = ['urgent', 'immediate', 'asap',
'important']
    if any(keyword in mail.lower() for keyword in
```

```
urgent_request_keywords):
        red_flags.append('Urgent request detected')

    # Перевірка позиції відправника електронного листа
    high_position_keywords = ['ceo', 'coo', 'president', 'founder']
    if any(keyword in mail.lower() for keyword in
high_position_keywords):
        red_flags.append('Email sent from someone in a high
position')

    # Перевірка тексту електронного листа
    email_text_keywords = ['money transfer', 'bank transfer', 'wire
transfer', 'DD', 'direct deposit']
    if any(keyword in 'mail'.lower() for keyword in
email_text_keywords):
        red_flags.append('Email text contains request for money
transfer')

    return red_flags


def detect_red_flags_address(s_address):
    red_flags = []
    domain_keywords = {'gmail.com', 'hotmail.com', 'outlook.com'}
    # Перевірка спуфінгу домену відправника
    sender_domain = s_address.split('@')[-1]
    if any(keyword in sender_domain for keyword in domain_keywords):
        red_flags.append('Hotmail sender domain detected')
    if sender_domain != "company.com":
        red_flags.append('Email came outside of the organization')
    return red_flags


# Визначення функції для обробки текстових даних шляхом токенізації,
стемінгу та лематизації.
def process_text(text):
    text = text.lower()
    stemmer = SnowballStemmer('english')
    token_text = nltk.word_tokenize(text)
    if stemmer is not None:
        token_text = [stemmer.stem(word) for word in token_text]
        lemmatizer = WordNetLemmatizer()
        token_text = [lemmatizer.lemmatize(word) for word in
token_text]
    text = ' '.join(token_text)
    return text


machine_learning(data)
```

```
"""for index, row in data.iterrows():
    red_flags_mail = detect_red_flags_mail(row['mail'])
    red_flags_address = detect_red_flags_address(row['s_address'])
    if red_flags_mail:
        print(f"Red flags for mail in row {index}: {red_flags_mail}")
    if red_flags_address:
        print(f"Red flags for address in row {index}:
{red_flags_address}")"""
```