Хмельницький національний університет
Факультет програмування та комп’ютерних і телекомунікаційних систем
Кафедра комп'ютерних наук та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему *Застосування агрегативних підходів для класифікації на базі ансамблевих моделей*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп’ютерні науки*
Шифр і назва спеціальності

Освітня програма *Комп’ютерні науки*
Назва освітньої програми

Виконав:	*студент 4 курсу, група КН-17-1*		*М.М.Стебелецький*
	Курс, група виконавця	Підпис	Ініціали, прізвище
Керівник:	*к.т.н., доцент кафедри КНІТ*		*Е.А. Манзюк*
	Науковий ступінь, посада	Підпис	Ініціали, прізвище
Нормоконтроль:	*к.т.н., доцент кафедри КНІТ*		*Р.О. Багрій*
	Науковий ступінь, посада	Підпис	Ініціали, прізвище

До захисту допускаю:

Зав. кафедри КНІТ, д.т.н., професор		*О.В. Бармак*
	Підпис	Ініціали, прізвище

8 червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет *програмування та комп'ютерних і телекомунікаційних систем*
Кафедра *комп'ютерних наук та інформаційних технологій*
Освітній ступінь *бакалавр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук та інформаційних технологій*
_____
(підпис)
д.т.н., професор О.В. Бармак
«8» лютого *2021* року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

1. Тема кваліфікаційної роботи бакалавра: *«Застосування агрегативних підходів для класифікації на базі ансамблевих моделей»*

2. Завдання видано студентці *Стебелецькому Мирославу Мироновичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *доцент кафедри КНІТ Манзюк Едуард Андрійович*
(посада, прізвище, ім'я, по батькові)

4. Затверджено наказом університету від «05» 02 *2021* р. № 11

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Мета роботи – дослідження застосування агрегативних підходів для класифікації на базі ансамблевих моделей. Протягом дослідження слід розробити інформаційну технологію класифікації на базі кластерного аналізу, задля наочності результатів дослідження потрібно реалізувати систему візуалізації даних у двох та трьох-вимірному просторах.

Виконавець:	*студент 4 курсу, група КН-17-1*		*М.М. Стебелецький*
	Курс, група виконавця	Підпис	Ініціали, прізвище
Керівник:	*к.т.н., доцент кафедри КНІТ*		*Е.А. Манзюк*
	Науковий ступінь, посада	Підпис	Ініціали, прізвище

Анотація

Тема кваліфікаційної роботи бакалавра: *«Застосування агрегативних підходів для класифікації на базі ансамблевих моделей»*

Виконавець кваліфікаційної роботи бакалавра: *студент групи КН-17-1 Стебелецький Мирослав Миронович*

Керівник кваліфікаційної роботи бакалавра: *кандидат технічних наук, доцент кафедри КНІТ Манзюк Едуард Андрійович*

Кваліфікаційна робота бакалавра містить:

Пояснювальна записка				Кількість додатків
Сторінок	Рисунків	Таблиць	Джерел інформації	
73	28	2	33	5

Метою кваліфікаційної роботи бакалавра є дослідження застосування агрегативних підходів для класифікації контенту на базі ансамблевих моделей. Для виконання дослідження було обрано мову програмування Python, а також бібліотеку Scikit-Learn для зручного використання методів класифікації в машинному навчанні.

Результатом виконання кваліфікаційної роботи бакалавра є застосунок, призначений для класифікації контенту, який може аналізувати вхідні дані та розбивати їх на окремі кластери. Застосунок використовує таку ансамблеву модель, як беггінг, в якості агрегативного підходу.

Дослідження застосування агрегативних підходів допоможе покращити результативність класифікаційної системи, зокрема, покращенню підлягають такі метрики системи, як точність та повнота.

Виконавець: *студент 4 курсу, група КН-17-1* _____ *М.М.Стебелецьки*
Курс, група виконавця — Підпис — Ініціали, прізвище

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
МН	Машинне навчання
ШІ	Штучний інтелект
SDK	Software development kit
DBSCAN	Density-based spatial clustering of applications with noise
NLP	Natural Language Processing
TF-IDF	Term Frequency – Inverse Document Frequency
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
SVM	Support Vector Machine
KNN	K-Nearest Neighbors

Вступ

Штучний інтелект (ШІ), зокрема, його підрозділ машинне навчання (МН), сьогодні розповсюджене в багатьох сферах життя і виконує практичні задачі різного напрямку. Оскільки основними задачами в теперішньому часі є збір даних з необхідністю аналізування, найбільш ефективним методом роботи з даними є МН [1].

Алгоритми МН відшукують (ще поки невідомі) природні зв'язки між даними, аналізують їх й передбачають які рішення найкращі. При цьому до цих рішень машина приходить сама й про їх можливе існування до початку роботи вона не знала.

Методи МН аналізують дані різними способами, зокрема, розповсюдженим методом аналізу даних є класифікація даних. Класифікація даних - завдання, яке необхідне в більшості дослідницьких робіт [2]. Задача класифікації даних вирішує необхідність класифікації довільного об'єкту з вихідної множини, для якого відомо, до якого класу він належить. Під класифікацією розуміють призначення об'єктові класу, до якого він належить.

Класифікація даних загалом визначається як процес організації даних за відповідними категоріями, щоб вони могли бути використані та захищені більш ефективно. На базовому рівні процес класифікації полегшує пошук та отримання даних. У бізнес-аналітиці класифікація даних тісно пов'язана з кластеризацією даних [3], але там, де кластеризація даних є описовою, класифікація даних є передбачуваною. По суті, класифікація даних полягає у використанні змінних з відомими значеннями для прогнозування невідомих або майбутніх значень інших змінних.

МН покликане для розробки складних моделей та алгоритмів в області аналізу даних, які піддаються прогнозуванню. Аналітичні моделі, які застосовує МН, дозволяють дослідникам, вченим, інженерам та аналітикам виробляти надійні, повторювані рішення та результати, розкривати «приховані уявлення» шляхом вивчення історичних взаємозв'язків та тенденцій у наборі даних.

Розділ 1
Характеристика предметної області та постановка задачі

1.1 Аналіз предметної області

Машинне навчання - це метод аналізу даних, який автоматизує побудову аналітичної моделі. Це галузь штучного інтелекту, заснована на ідеї, що системи можуть вчитися на основі даних, визначати закономірності та приймати рішення з мінімальним втручанням людини.

Через нові обчислювальні технології сьогодні МН не схоже на МН минулого. Воно народилось завдяки розпізнаванню образів та теорії, згідно з якою комп'ютери можуть вчитися, не будучи запрограмованими на виконання конкретних завдань; дослідники, які цікавляться ШІ, хотіли дізнатись, чи можуть комп'ютери вчитися на даних. Ітераційний аспект машинного навчання важливий, оскільки, коли моделі потрапляють до нових даних, вони можуть самостійно адаптуватися. Вони навчаються на попередніх обчисленнях для отримання надійних, повторюваних рішень та результатів. Це наука не нова, але така, що набрала нових обертів.

Хоча багато алгоритмів МН існують вже давно, можливість автоматичного застосування складних математичних обчислень до великих даних - знову і знову, все швидше і швидше - є недавньою розробкою. Прикладами рішень, які використовують машинне навчання, можуть слугувати наступні:

- самокерований автомобіль Google (показує суть машинного навчання, оскільки навчається подібно класифікаційної моделі) [4];
- інтернет-рекомендації, створені на основі алгоритмів МН, такі як пропозиції від YouTube, Netflix, Amazon [5][6][7];
- застосунок для виявлення критики у Twitter (МН в поєднанні з створенням лінгвістичних правил) [8];

Більшість галузей, що працюють з великими обсягами даних, визнали цінність технології машинного навчання. Вибираючи цю технологію - часто в реальному часі - організації можуть працювати ефективніше або отримати перевагу над конкурентами.

Існують такі види технік МН, а саме: контрольоване навчання або навчання під наглядом, неконтрольоване навчання або навчання без нагляду.

Контрольоване навчання (Supervised learning) - один із способів МН, в ході якого випробувана система примусово навчається за допомогою наявної множини прикладів «стимул-реакція» з метою визначення «реакції» для «стимулів», які не належать наявній множини прикладів [9]. Всі техніки контрольованого навчання сформовані на основі класифікації або регресії. Завданням класифікації є передбачення категорії об'єкта і поділ об'єктів згідно з визначеними і заданими наперед ознаками. Тобто, машина сортує дані за потрібними категоріями [10]. Завданням регресії є прогнозування цільової змінної за заданим набором ознак, іншими словами, передбачення місця на числовій прямій [11]. Прикладом для класифікації може бути класифікація зображень: користувач тренирує систему із зображеннями або ярликами, потім у майбутньому користувач подасть нове зображення, очікуючи, що ситема розпізнає новий об’єкт. Прикладом для регресії може бути прогнозування або регресія ринку: користувач навчає систему історичними даними ринку та просить систему прогнозувати нову ціну в майбутньому.

Неконтрольоване навчання (Unsupervised learning) - один зі способів МН, при вирішенні яких випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора [12]. При неконтрольованому навчанні (навчання без учителя) все зводиться до знаходження шаблону на основі тільки введених даних. Ці техніки є корисними, коли цілком не відомо, що шукати. Часто використовується для дослідницького аналізу «сирих» даних. Більшість з технік неконтрольованого навчання є формою кластерного аналізу. Кластерний аналіз - розбиття заданої вибірки даних (об'єктів) таким чином, щоб кожен кластер складався зі схожих об'єктів, а об'єкти різних кластерів значно відрізнялися одне від одного [13]. Завдання кластеризації - використовуючи всі наявні дані, передбачити відповідність об'єктів вибірки їхнім класам, сформувавши таким чином кластери. Відмінність класифікації від кластеризації полягає в тому, що при класифікації у наявності є набір визначених класів, користувач вчить

систему на конкретних прикладах і потім хоче знати, до якого класу належить новий об'єкт. При кластеризації система використовуєте алгоритм, який намагається згрупувати набір об'єктів і визначити, чи існує будь-який взаємозв'язок між об'єктами, система вчиться сама.

Підсумовуючи, можна сказати, що штучний інтелект (ШІ) та машинне навчання (МН) сьогодні вважаються одними з найбільших нововведень з часів мікрочіпу. Раніше ШІ був фантастичним поняттям наукової фантастики, але зараз він стає щоденною реальністю.

МН на сьогоднішній час є одним з найкращих, найшвидших і найперспективніших методів для аналізу даних, дослідження в цій області зумовлює вирішення актуальних проблем оптимізації моделей агрегатування.

1.2 Аналіз існуючого програмного забезпечення предметної області

Задля оптимізації рішень алгоритмів, в машинному навчанні (МН) часто використовують ансамблеве моделювання [14]. Ансамблеве моделювання - це процес, коли для прогнозування результату створюється безліч різноманітних моделей, використовуючи численні різні алгоритми моделювання, або використовуючи різні набори навчальних даних. Потім ансамблева модель агрегує прогнозування кожної базової моделі і дає одноразово остаточне прогнозування для невидимих даних. Ансамблем (Ensemble, Multiple Classifier System) називається алгоритм, який складається з декількох алгоритмів машинного навчання, а процес побудови ансамблю називається ансамблюванням (ensemble learning). Найпростіший приклад ансамблю в регресії - усереднення декількох алгоритмів. Більшість прийомів в прикладному ансамблірованіі направлено на те, щоб ансамбль був досить різноманітним, тоді помилки окремих алгоритмів на окремих об'єктах будуть компенсуватися коректної роботою інших алгоритмів. По суті, при побудові ансамблю:

- підвищується якість базових алгоритмів;
- підвищується різноманітність (diversity) базових алгоритмів.

Існують різні види моделей ансамблювання.

Комітети (голосування, Voting Ensembles) – є одним з найпростіших способів поєднання прогнозів з декількох алгоритмів машинного навчання. При такому підході кожна базова модель робить прогноз і голосує для кожної вибірки. Тільки зразок класу з найбільшою кількістю голосів включається до остаточного класу передбачення [15].

Бегінг (Bagging) – цей тип навчання означає: багато разів навчати ансамбль на випадкових вибірках даних. І в кінцевому підсумку усереднити відповіді [16]. Це виглядає як голосування за найбільш популярну відповідь, де багато моделей працюють паралельно. Ідея бегінга (bootstrap aggregating) проста: кожен базовий алгоритм навчається на випадковій підмножині навчальної вибірки. В цьому випадку, навіть використовуючи одну модель алгоритмів, ми отримуємо різні базові алгоритми.

Випадковий ліс (Random Forest) – це керований алгоритм навчання. «Ліс», який він будує, - це ансамбль дерев рішень, які зазвичай навчають методом «Bagging» [17]. Загальна ідея методу бегінг полягає в тому, що поєднання моделей навчання покращує загальний результат. Випадковий ліс складається з множини дерев рішень. При задачі регресії їх відповіді усереднюються, при класифікації приймається рішення голосуванням за більшістю.

Бустінг (Boosting) – цей спосіб включає послідовне навчання алгоритмів. Тобто спершу навчається перший алгоритм і відзначаються місця, де він помилився [18].. Потім навчається другий, особливу увагу приділяючи місцям на яких помилявся перший. І так далі. До необхідного результату. Тут також робляться вибірки даних, проте вже не за випадковою ознакою. Тепер кожна наступна вибірка складається з тих даних, на яких помилився попередній алгоритм. Таким чином досягається найбільш якісний результат

Стекінг (Stacking) - спершу навчають кілька алгоритмів, потім результати їх роботи показують останньому алгоритму. Саме він і приймає остаточне рішення. Стекінг – хороший, але найменш точний ансамбль серед інших методів. Найпростіша схема стекінгу - блендінг (Blending): навчальну вибірку ділять на дві

частини. На першій навчають базові алгоритми. Потім отримують їхні відповіді на другій частині і на тестовій вибірці [19].

Наглядно показує можливості МН в області класифікації даних інтернет-ресурс uclassify.com (рисунок 1.1) [20]. uClassify - це безкоштовний веб-сервіс машинного навчання, де можливо легко створювати та використовувати класифікатори тексту.

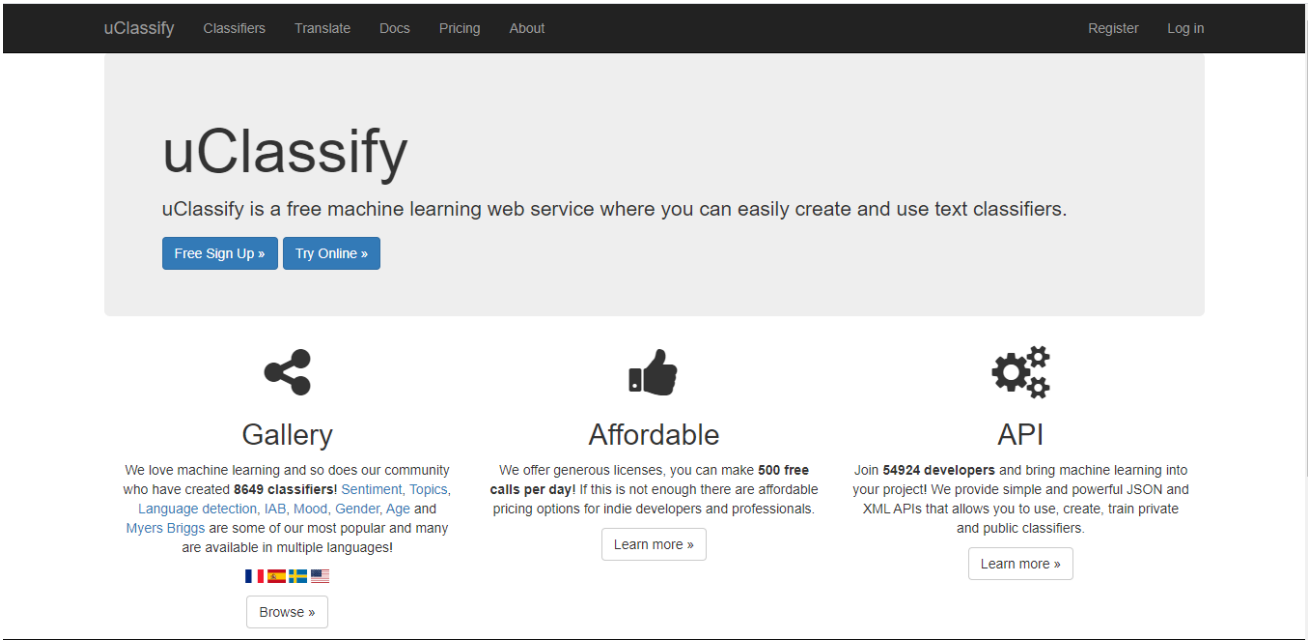


Рисунок 1.1 – Головна сторінка інтернет-ресурсу uclassify.com

На рисунках 1.2 – 1.3 зображений результат класифікатора аналізу сентиментальності тексту. Цей класифікатор визначає позитивний чи негативний текст. Він добре підходить як для коротких, так і для довгих текстів (твітів, статусів у Facebook, публікацій у блогах, оглядів продуктів тощо). Він навчений 2,8 мільйонам документів із даними Twitter, оглядів продуктів Amazon та оглядів фільмів. З його допомогою можна проводити дослідження, опитування брендів і бачити тенденції навколо ринкових кампаній.

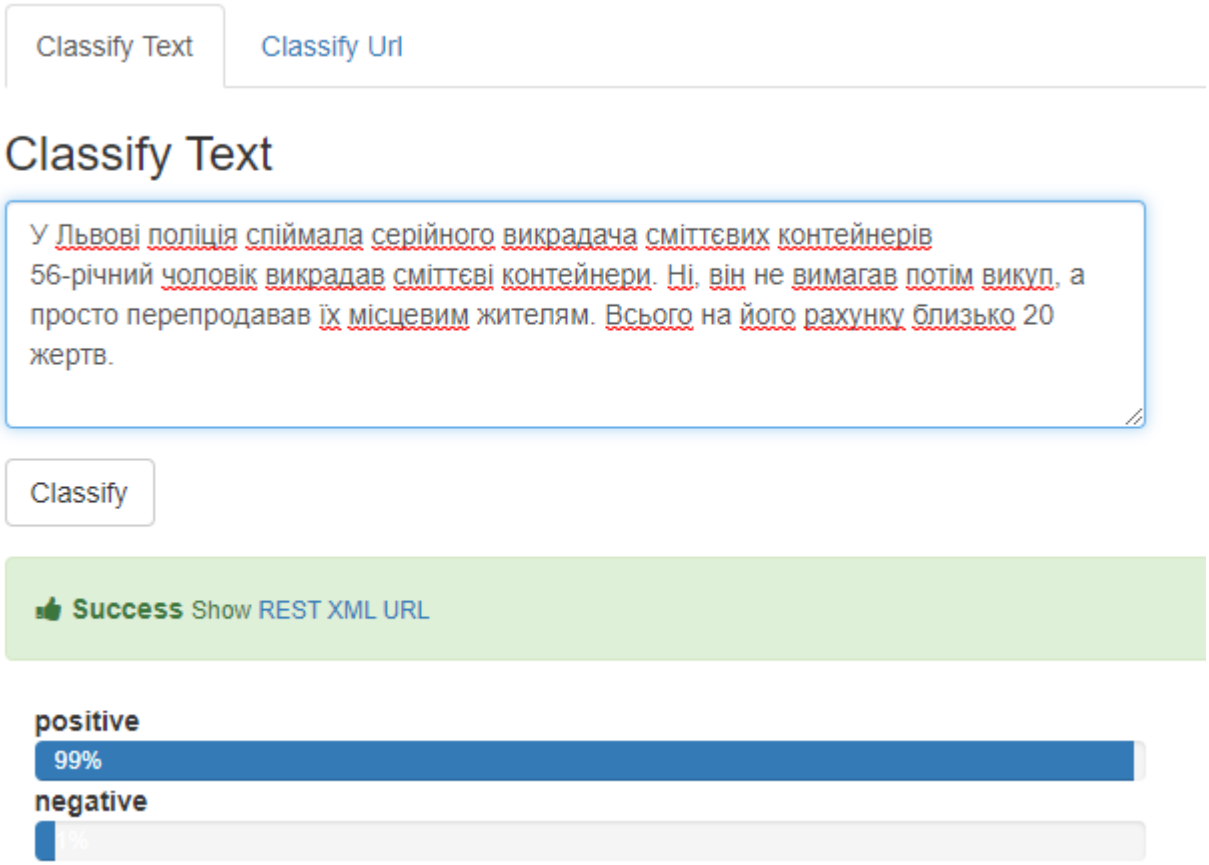


Рисунок 1.2 – Результат класифікатора аналізу сентиментальності тексту

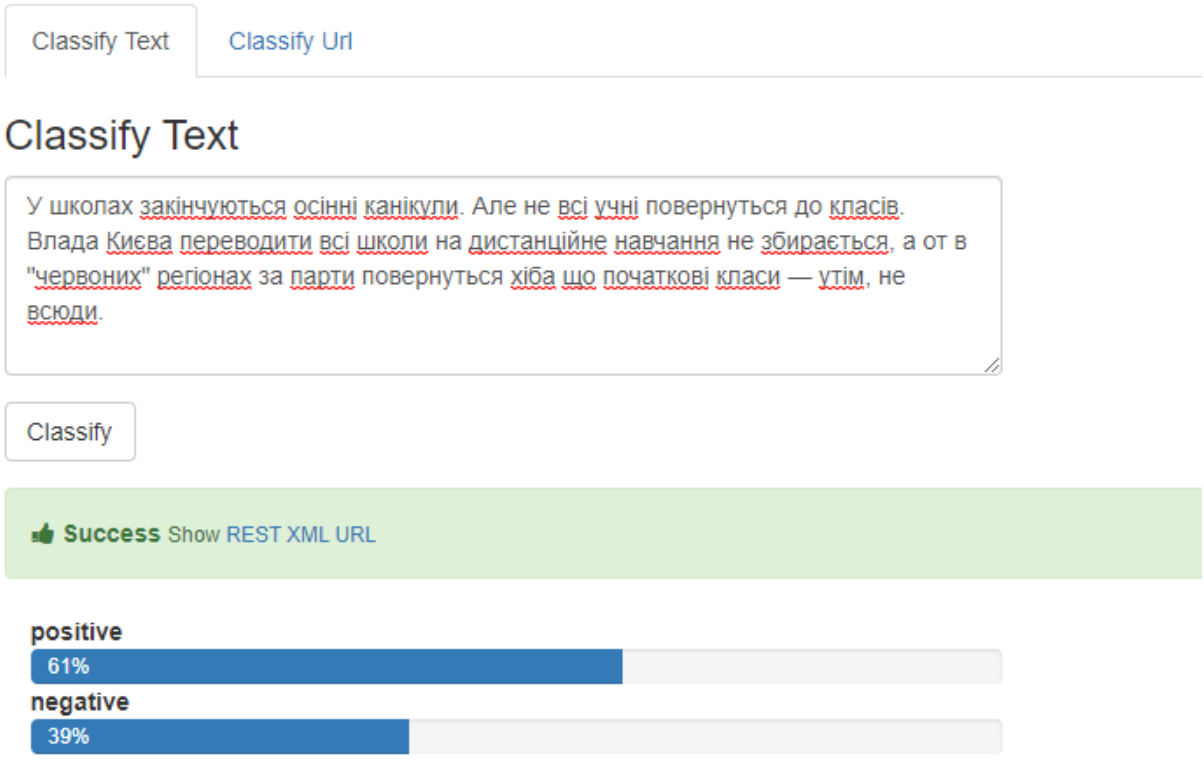


Рисунок 1.3 – Результат класифікатора аналізу синтементальності тексту

Класифікатор аналізу сентиментальності тексту, який зображений на рисунках 1.2 – 1.3, використовує модель наївного баєсівського класифікатора [21]. Класифікатори, які використовують цю модель, містять сукупність алгоритмів класифікації, заснованих на теоремі Байєса [22] . Це не єдиний алгоритм, а сімейство алгоритмів, де всі вони мають спільний принцип, тобто кожна пара ознак, що класифікуються, не залежить одна від одної.

Інтернет-ресурс uclassify.com містить в собі великий обсяг класифікаторів різної мети, на рисунку 1.4 зображений результат класифікатору, метою якого є визначити гендер особи, яка написала текст. Цей класифікатор намагається з'ясувати, написаний текст чоловіком чи жінкою. Він пройшов навчання на 11000 блогах (5500 блогів, написаних жінками та 5500 чоловіками). Більший об'єм тексту дає кращі результати.

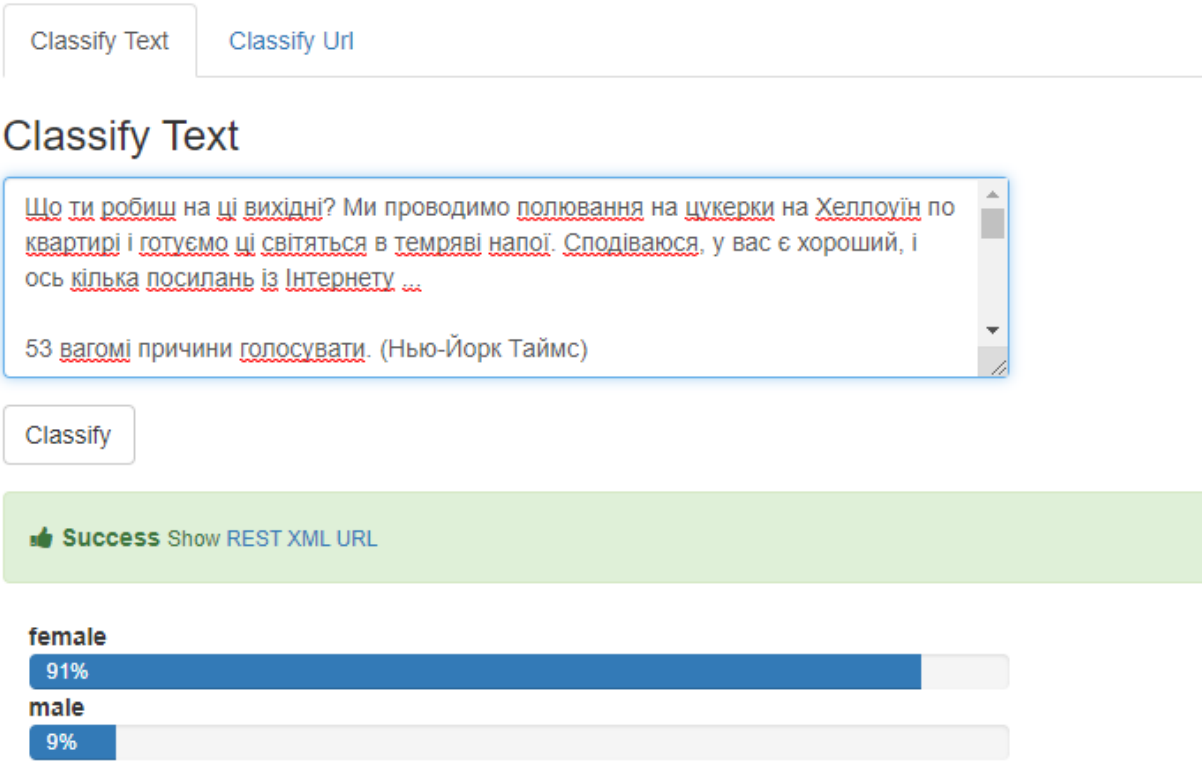


Рисунок 1.4 – Результат класифікатора аналізу гендера

Основним недоліком класифікаторів, які містить інтернет-ресурс uclassify.com є те, що класифікація не містить 100% (конкретної) відповіді, тобто відповідь «розмивається» у відсотковому співвідношенні між класами, які передбачає класифікатор.

Виходячи з вище зазначеного, можна сказати, що найбільш актуальним і популярним напрямком в МН є класифікація, оскільки процес класифікації даного набору даних за класами, може виконуватися як на структурованих, так і на неструктурованих даних. Головна мета - визначити, до якого класу категорії потраплять нові дані.

1.3 Аналіз сучасних засобів створення програмного забезпечення

Вибір засобів й технології створення програмного продукту визначає загальну організацію роботи застосунку, тому істотно впливає на постановку завдання. Необхідно базуватися на доцільності використання засобів розробки для реалізації застосунку, відповідному меті дослідження. Для аналізу була обрана платформа ARM з бібліотекою NN, мова програмування Python з бібліотекою Scikit-Learn, платформа c# .Net з бібліотекою Accord.Net.

ARM NN - це набір програмного забезпечення та інструментів з відкритим кодом для Linux, що забезпечує завантаження машинного навчання на енергоефективних пристроях. Він забезпечує взаємозв’язок між існуючими нейронними мережевими структурами та енергоефективними процесорами Cortex-A , графічними процесорами Arm Mali та NPM Arm Ethos [23].

ARM NN містить наступні основні переваги:

- arm NN SDK використовує обчислювальну бібліотеку для максимально ефективного націлювання на програмовані ядра;
- є досить універсальною для масштабування на будь-який пристрій;
- містить відкритий програмний код.

До основних недоліків ARM NN можна віднести наступні:

- оптимізація виконання алгоритмів МН можлива тільки на процесорах на базі Arm;
- підтримувана операційна система – Linux.

Python Scikit-Learn - це безкоштовна програмна бібліотека машинного навчання для мови програмування Python, яка надає функціональність для створення та тренування різноманітних алгоритмів класифікації, регресії та кластеризації, і працює у зв'язці з бібліотеками NumPy та SciPy. Scikit-learn є однією з найбільш популярних бібліотек машинного навчання [24].

Основними переваги Python Scikit-Learn є наступні:

- низький поріг входження (за рахунок мови програмування);
- працює на основі декількох поширених математичних бібліотек, і легко інтегрує їх один з одним;
- забезпечує узгоджений інтерфейс для алгоритмів машинного навчання;
- пропонує безліч параметрів настройки для кожного алгоритму і використовує розумні значення за замовчуванням;
- має багатий функціонал для задач, пов'язаних з машинним навчанням;
- широке співтовариство і докладна документація.

Scikit-Learn написана на Python, що, з одного боку, уможливлює низький поріг входження, проте, з іншого боку, Python – це мова із динамічною типізацією типів, що в деяких моментах може дуже сильно ускладнити роботу, тому що «помилки», якщо такі існують, появляються не на етапі компіляції коду, а на етапі виконання програми.

Accord.Net – це середовище машинного навчання .NET, об'єднане з бібліотеками обробки аудіо та зображень, повністю написаними на C #. Це повна структура для створення додатків для комп'ютерного зору промислового рівня, комп'ютерного прослуховування, обробки сигналів і статистики для комерційного використання [25].

Переваги Accord.Net:

- великий набір прикладів додатків (забезпечують швидкий початок роботи);

- розроблене на об'єктно-орієнтованій мові програмування C #;
- містить набір бібліотек, доступних у вихідному коді;
- велика документація та Wiki (допомагають заповнити деталі).

Основним недоліком Accord.Net є те, що .Net Framework є залежним від платформи Windows, а розробка і підтримка додатків вимагає використання ліцензійного програмного забезпечення, вартість ліцензування висока порівняно з конкурентами. Також, до недоліку можна віднести високий поріг входження в в середовище .Net Framework.

Підсумувавши переваги і недоліки перерахованих платформ і бібліотек, та з огляду на доцільність їх використання для обраного напрямку, оптимальним є вибір мови програмування Python з бібліотекою Scikit-Learn.

1.4 Постановка задачі та вимоги до розробки інформаційної системи

Потрібно провести дослідження застосування агрегативних підходів для класифікації контенту на базі ансамблевих моделей. Застосунок, розроблений в ході дослідження, повинен бути реалізованим на мові програмування Python з використанням бібліотеки Scikit-Learn. Під час дослідження потрібно виконати наступні завдання:

1) провести аналіз сучасних підходів на базі ансамблів;
2) порівняти ефективність застосування відомих методів щодо предметної області;
3) розробити інформаційну технологію класифікації на базі кластерного аналізу;
4) дослідження ефективності запропонованих рішень шляхом експериментальних досліджень на відомих корпусах даних;
5) провести порівняльний аналіз ефективності запропонованого методу та відомих підходів.

Дослідження в цій області покликане вирішити актуальні проблеми оптимізації моделей агрегатування.

Розділ 2
Проектування інформаційної системи

2.1 Загальний опис інформаційної технології класифікації на базі кластерного аналізу

Аналіз текстових даних в машинному навчанні використовує методи регресії, класифікації та кластеризації. Але варто відзначити що є головна відмінність в аналізі текстових даних, так як сама обробка тексту є дуже складним завданням в машинному навчанні. Головна відмінність - це інтелектуальний аналіз текстових даних. Так як текстовий документ для людини - це набір слів, який несе сенс, для машини - це просто бітові дані. І завдання інтелектуального аналізу текстових даних полягає в тому, щоб машина змогла розуміти сенс текстового документа. Перед тим як використовувати алгоритми машинного навчання, потрібно також застосувати методи обробки текстових даних.

Інтелектуальний аналіз даних і методи машинного навчання використовуються разом для автоматичної класифікації і виявлення шаблонів в електронних документах. Основна мета аналізу тексту - дати користувачам можливість отримувати інформацію з текстових ресурсів і займатися такими операціями, як класифікація.

Аналіз тексту складається з декількох завдань, зокрема:

- правильна анотація до документів;
- відповідне подання документа;
- зменшення розмірності для обробки алгоритмічних питань і відповідна функція класифікатора для отримання хорошого узагальнення.

Подання документів є одним з методів попередньої обробки, який використовується для зменшення складності документів і полегшення їх обробки, документ повинен бути перетворений з повнотекстової версії в вектор документа. Текстове представлення є важливим аспектом в класифікації або кластеризації документів.

Попередня обробка включає в себе такі етапи як:

- вилучення ознак (характеристик);
- вибір ознак.

Вилучення ознак є першим етапом попередньої обробки, який використовується для представлення текстових документів у форматі слів. Таким чином, алгоритми видалення стоп-слів, стеммитизація, токенізація і інші - це завдання попередньої обробки. Дані алгоритми відноситься до методу обробки природної мови (NLP) [26].

Після вилучення ознак важливим кроком попередньої обробки тексту є вибір ознак для побудови векторного простору, що підвищує масштабованість, ефективність і точність текстового класифікації або кластеризації. В цілому, хороший метод вибору ознак повинен враховувати характеристики вхідних даних і алгоритму. Основна ідея вибору ознаки полягає у виборі підмножини об'єктів з вихідних документів.

Для класифікації або кластеризації тексту основною проблемою є висока розмірність простору об'єктів. Майже кожен текстовий документ має велику кількість ознак, більшість з яких не є актуальними і корисними для завдання машинного навчання, і навіть деякі шумові ознаки можуть різко знизити точність алгоритму. Тому вибір ознак зазвичай використовується для зменшення розмірності простору ознак і підвищення ефективності та точності.

Для візуального представлення структури набору даних з багатомірними характеристиками можуть бути використані 1 2- і 3-мірні простори відображень, для дослідження було обрано візуалізацію за допомогою 2-мірної та 3-мірної поверхні, оскільки саме в такому вигляді людина сприймає геометричні структури найприродніше і зв’язки між об'єктами виглядають найбільш наочно. Візуалізація даних - спосіб відображення багатовимірних характеристик даних на площині, при цьому, якісно відображені основні закономірності, притаманні вхідному наборові даних .

2.2 Математико-алгоритмічне забезпечення інформаційної технології класифікації на базі кластерного аналізу

Кластеризація (або кластерний аналіз) - це задача розбиття множини об'єктів на групи, які називаються кластерами. У середині кожної групи повинні міститись «схожі» об'єкти, а об'єкти різних груп повинні бути якомога більш відмінні. Головна відмінність кластеризації від класифікації полягає в тому, що перелік груп чітко не заданий і визначається в процесі роботи алгоритму.

Застосування кластерного аналізу в загальному вигляді зводиться до наступних етапів:

- відбір вибірки об'єктів для кластеризації;
- визначення множини змінних (стимулів), за якими будуть оцінюватися об'єкти у вибірці. При необхідності - нормалізація значень змінних;
- обчислення значень міри схожості між об'єктами;
- застосування методу кластерного аналізу для створення груп схожих об'єктів (кластерів);
- представлення результатів аналізу.

Методи кластерного аналізу можна розділити на дві групи: ієрархічні, неієрархічні.

При ієрархічній кластеризації виконується послідовне об'єднання менших кластерів у великі або поділ великих кластерів на менші. Ієрархічні методи кластерного аналізу поділяються на дві групи: ієрархічні агломеративні методи (Agglomerative Nesting, AGNES), ієрархічні дивізивні (подільні) методи (DIvisive ANAlysis, DIANA) [27].

Агломеративні методи характеризуються послідовним об'єднанням вихідних елементів і відповідним зменшенням числа кластерів. На початку роботи алгоритму всі об'єкти є окремими кластерами. На першому кроці найбільш схожі об'єкти об'єднуються в кластер. На наступних кроках об'єднання триває до тих пір, поки всі об'єкти не будуть складати один кластер.

Дивізивні методи є логічною протилежністю агломеративним методам. На початку роботи алгоритму всі об'єкти належать одному кластеру, який на наступних кроках ділиться на менші кластери, в результаті утворюється послідовність роздільних груп.

При неієрархічній кластеризації повинна бути визначена кількість кластерів, кількість ітерацій або правило зупинки, що являється суттєвим недоліком відносно ієрархічної кластеризації. Проте, при великій кількості спостережень ієрархічні методи кластерного аналізу не придатні. У таких випадках використовують неієрархічні методи, засновані на поділі, які представляють собою ітеративні методи дроблення вихідної сукупності. У процесі поділу нові кластери формуються до тих пір, поки не буде виконано правило зупинки. Така неієрархічна кластеризація полягає в поділі набору даних на певну кількість окремих кластерів. Існує два підходи. Перший полягає у визначенні меж кластерів як найбільш щільних ділянок в багатовимірному просторі вихідних даних, тобто визначення кластера там, де є велике "згущення точок". Другий підхід полягає в мінімізації ступеня відмінності об'єктів.

Найбільш поширений серед неієрархічних методів - алгоритм k-середніх (k-means), також званий швидким кластерним аналізом [28]. На відміну від ієрархічних методів, які не вимагають попередніх припущень щодо числа кластерів, для можливості використання цього методу необхідно мати гіпотезу про найбільш ймовірну кількість кластерів. Мета алгоритму наступна: мінімізувати середньоквадратичне відхилення на точках кожного кластера. Основна ідея полягає в тому, що на кожній ітерації переобчислюється центр мас для кожного кластера, отриманого на попередньому кроці, потім вектори розбиваються на кластери знову відповідно до того, який з нових центрів виявився ближчим за обраною метрикою. Алгоритм завершується, коли на якийсь ітерації не відбувається зміни кластерів.

Для того, щоб порівнювати два об'єкти, необхідно мати критерій, на підставі якого буде відбуватися порівняння. Як правило, таким критерієм є відстань між об'єктами.

Даними для алгоритму k-means є матриця, що складається з відстаней між кожним об’єктом. Щоб визначити відстані між об’єктами, потрібно мати міру відстаней. Евклідова відстань - найпоширеніша метрика відстані. Це геометрична відстаннь в багатовимірному просторі. Якщо об’єкти визначаються багатовимірними точками, або мають багато характеристик (стимулів), $X_i = \{ x_{i1}, x_{i2}, x_{i3}, \dots, x_{in}\}$, де $i = 1 \dots n$, відстань може бути визначена відстанню між точками $d(X_i, X_j)$, де

$$d(X_i, X_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2} \quad (2.1)$$

Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера, тобто функції:

$$\sum_{i=1}^{N} d\left(x_i, m_j(x_i)\right)^2, \quad (2.2)$$

де d – метрика, x_i - і-тий об’єкт даних, а $m_j(x_i)$ – центр кластера, якому на j-тій операції присвоєний елемент x_i.

Принцип алгоритму полягає в пошуку таких центрів кластерів та наборів елементів кожного кластера при наявності деякої функції Ф(°), що виражає якість поточного розбиття множини на k кластерів, коли сумарне квадратичне відхилення елементів кластерів від центрів цих кластерів буде найменшим:

$$V = arg \min \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2 \quad (2.3)$$

де k – число кластерів, S_i - отримані кластери, $i = 1,2, \dots, k$, μ_i – центри мас векторів $x_j \in S_i$.

Метод k-середніх використовується для кластеризації даних на основі алгоритму розбиття векторного простору на заздалегідь визначену кількість

кластерів k. Алгоритм являє собою ітераційну процедуру, в якій виконуються наступні кроки:

1. Вибирається число кластерів k.
2. З вихідної множини даних випадковим чином вибираються k спостережень, які будуть служити початковими центрами кластерів.
3. Для кожного спостереження вихідної множини визначається найближчий до нього центр кластера (відстані вимірюються в метриці Евкліда). При цьому записи, «притягнуті» певним центром, утворюють початкові кластери.
4. Обчислюються центроїди - центри мас кластерів. Кожен центр мас - це вектор, елементи якого являють собою середні значення відповідних стимулів, обчислені за всіма елементами кластера.
5. Центр кластера зміщується в його центр ваги, після чого центр ваги стає центром нового кластера.
6. 3-й і 4-й кроки ітераційно повторюються. Очевидно, що на кожній ітерації відбувається зміна меж кластерів і зміщення їх центрів. В результаті мінімізується відстань між елементами всередині кластерів і збільшуються міжкластерні відстані.

Зупинка алгоритму проводиться тоді, коли границі кластерів і розташування центроїдів не перестануть змінюватися від ітерації до ітерації, тобто на кожній ітерації в кожному кластері буде залишатися один і той же набір об’єктів. На практиці алгоритм зазвичай знаходить набір стабільних кластерів за кілька десятків ітерацій.

Вибір числа кластерів є складним питанням. Якщо немає припущень щодо цього числа, рекомендують створити 2 кластера, потім 3, 4, 5 і т.д., порівнюючи отримані результати. На рисунку 2.1 наведено приклад роботи алгоритму k-середніх для k, рівного двом.

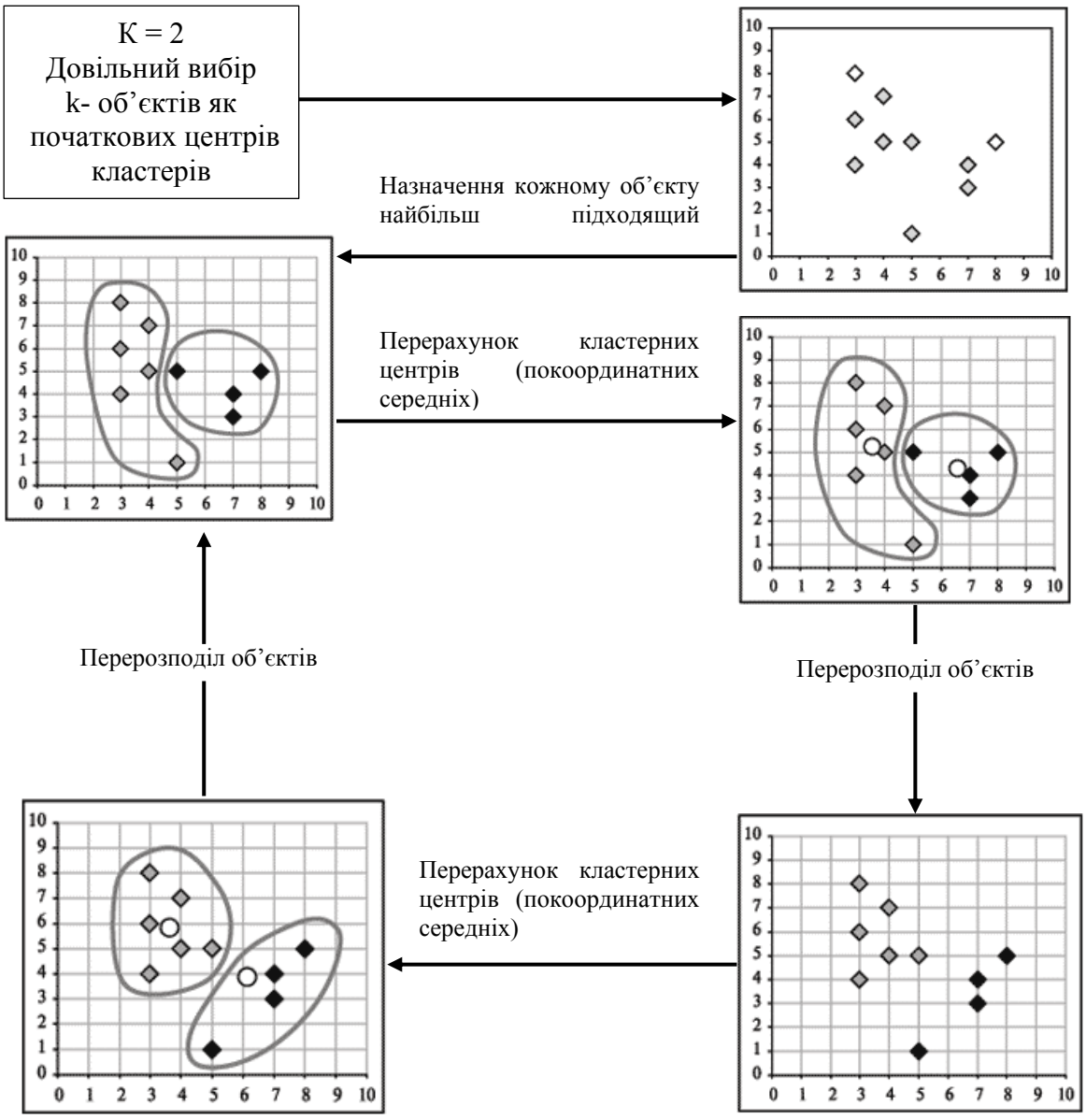


Рисунок 2.1 - Приклад роботи алгоритму k-середніх (k = 2)

На рисунку 2.2 зображена блок-схема алгоритму k-середніх (k-means)

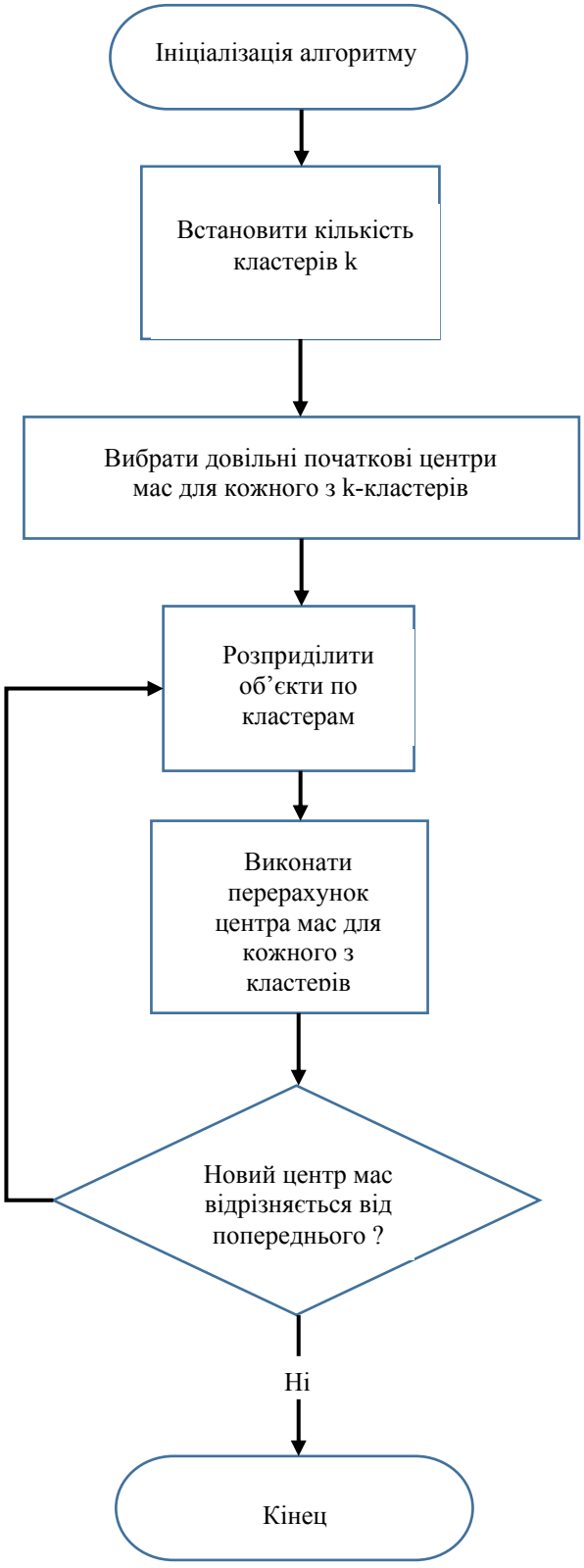


Рисунок 2.2 - Блок-схема алгоритму k-середніх (k-means)

Переваги алгоритму k-середніх (k-means):

- простота використання;
- швидкість використання;
- зрозумілість і «прозорість» алгоритму.

Недоліки алгоритму k-середніх (k-means):

- результат класифікації сильно залежить від випадкових початкових позицій кластерних центрів;
- алгоритм чутливий до викидів, які можуть викривлювати значення кластерних центрів (центроїдів);
- кількість кластерів повинна бути заздалегідь визначена.

Методи поділу (K-means, PAM-кластеризація) і ієрархічна кластеризація виконують пошук кластерів сферичної форми або опуклих кластерів. Іншими словами, вони підходять тільки для компактних і добре розділених кластерів. Більш того, на них також сильно впливає присутність шуму і викидів в даних. Дані з реального життя можуть містити неточності, наприклад:

- кластери можуть мати довільну форму, таку як показано на рисунку 2.3;
- дані можуть містити шум.

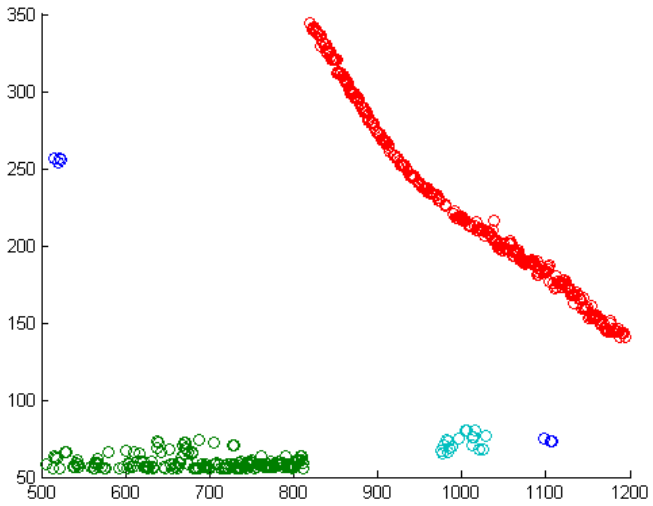


Рисунок 2.3 – Кластери не стандартної форми

Методи кластеризації на основі поділу і ієрархії дуже ефективні з кластерами нормальної форми. Однак коли справа доходить до кластерів довільної форми або виявлення викидів, методи на основі щільності більш ефективні. На рисунку вище показаний набір даних, що містить неопуклі кластери і викиди / шуми. З огляду на такі дані, алгоритм k-середніх має труднощі з ідентифікацією цих кластерів довільної форми.

Заснований на щільності кластерів простору ознак для додатків з шумами (англ. Density-based spatial clustering of applications with noise, DBSCAN) - це алгоритм кластеризації даних, заснований на щільності - якщо даний набір точок в деякому просторі, алгоритм групує разом точки, які тісно розташовані (точки з багатьма близькими сусідами), позначаючи як викиди точки, які віддалені і знаходяться в областях з малою щільністю (найближчі сусіди яких лежать далеко). DBSCAN є одним з найбільш часто використовуваних алгоритмів кластеризації [29]. Ідея, покладена в основу алгоритму, полягає в тому, що всередині кожного кластера спостерігається типова щільність точок (об'єктів), яка помітно вище, ніж щільність зовні кластера, а також щільність в областях з шумом нижче щільності будь-якого з кластерів, тобто, основна ідея DBSCAN полягає в тому, що точка належить кластеру, якщо вона знаходиться поруч з багатьма точками з цього кластера. Якщо точніше виражатись, то для кожної точки кластера її сусідство заданого радіуса має містити не менше певної кількості точок, це число точок задається граничним значенням.

DBSCAN містить два основних параметра:

- eps: визначає окіл навколо точки даних, тобто якщо відстань між двома точками менше або дорівнює «eps», то вони вважаються сусідами. Якщо значення eps вибрано дуже маленьким, велика частина даних буде вважатися викидами. Якщо eps обраний дуже великим, кластери об'єднаються, і більшість точок даних будуть в одних і тих же кластерах;

– minPts: мінімальна кількість сусідів (точок даних) в радіусі eps. Чим більший набір даних, тим більше значення minPts має бути вибрано. Як правило, мінімальна кількість minPts може бути отримана з кількості вимірів D в наборі даних, зокрема, minPts >= D + 1. Мінімальне значення MinPts має бути не менше 3.

На основі цих двох параметрів точки класифікуються як базова (основна) точка, гранична точка або викид (шум):

– базова точка: точка, в околі області з радіусом eps якої є не менше minPts кількості точок (включаючи саму точку);
– гранична точка: точка, яка досяжна з базової точки і в її околі менше minPts кількості точок;
– шум: точка, яка не є основною і недоступна з будь-яких основних точок.

Ці моменти можна краще пояснити за допомогою візуалізації. Наступний рисунок 2.4 взятий з Вікіпедії.

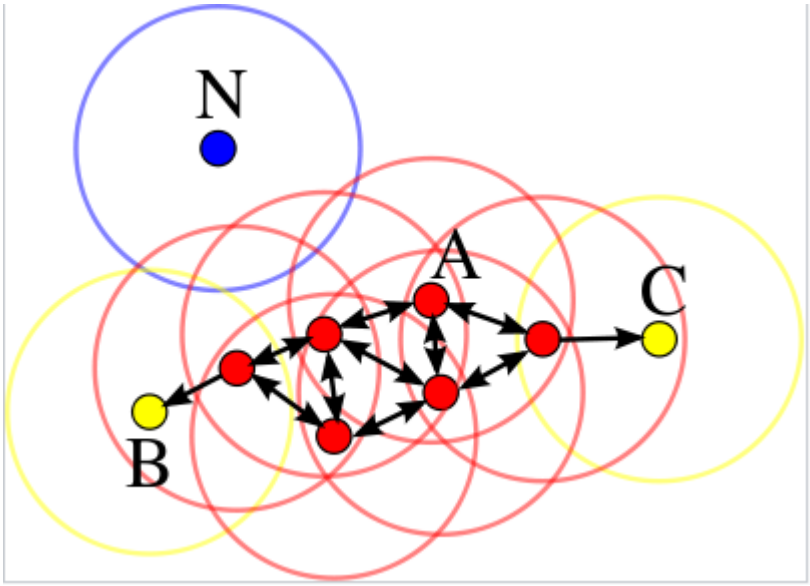


Рисунок 2.4 – діаграма розміщення точок

У випадку, зображеному на рисунку 2.4 minPts рівно 4. Червоні точки є основними, тому що в їх околах є як мінімум 4 точки з радіусом eps. Ця область

показана на малюнку кругами. Жовті точки є граничними, тому що вони досяжні з центральної точки і мають менше 4 точок в межах свого оточення. Досяжність означає знаходження в безпосередній близькості від основної точки. Точки B і C мають дві точки (включаючи саму точку) в межах своєї околиці (тобто навколишнього області з радіусом eps). Нарешті, N є викидом, тому що це не основна точка і не може бути досягнута з основної точки.

Алгоритм DBSCAN містить наступні кроки:

1. Визначається eps та minPts .
2. Знаходяться всі сусідні точки в межах eps і визначаються основні точки або точки, у яких кількість сусідів minPts.
3. Для кожної основної точки, якщо вона ще не призначена кластеру, створюється новий кластер.
4. Рекурсивно знаходяться всі пов'язані точки, які відповідають умові щільності і призначаються кластери, відповідні до основних точок.
5. Перебираються невідвідані точки в наборі даних. Ті точки, які не належать ні одного кластеру, є шумом.

Відстань між точками визначається з використанням того ж методу вимірювання відстані, як в алгоритмі k-середніх. Евклідова відстань - Найбільш часто використовуваний метод.

Переваги алгоритму DBSCAN:

- не потрібно заздалегідь вказувати кількість кластерів;
- добре працює з кластерами довільної форми;
- DBSCAN стійкий до викидів і здатний виявляти викиди.

Недоліки алгоритму DBSCAN:

- у деяких випадках визначити правильну відстань сусідства (eps) непросто і вимагає знання предметної області;
- якщо кластери сильно розрізняються по щільності, DBSCAN не підходить для визначення кластерів. Характеристики кластерів визначаються комбінацією параметрів eps-minPts. Оскільки

передається одна комбінація eps-minPts, вона не може добре узагальнюватися на кластери, в яких щільність сильно відрізняється.

Алгоритми DBSCAN та k-середніх, які були розглянуті вище, складають малу долю алгоритмів, які використовуються для кластеризації даних. Проте, важливим є той факт, що всі алгоритми кластеризації на початку своєї роботи отримують дані для подальшого аналізу. Варто зазначити, що вхідні дані алгоритму повинні бути нормалізовані. Під нормалізованістю вхідних даних розуміють те, що дані не повинні містити зайвої інформації, дані повинні бути інтерпретовані у правильну «форму» (зазвичай це числа).

Машинне навчання з використанням природної мови стикається з одною серйозною перешкодою - його алгоритми зазвичай працюють з числами, а природна мова - це, в загальному, текст. Тому потрібно перетворити цей текст в числа, або зробити векторизацію тексту, як прийнято зазначати [30]. Це фундаментальний крок в процесі машинного навчання для аналізу даних, і різні алгоритми векторизації істотно вплинуть на кінцеві результати. В даному випадку найбільш підходящий алгоритм векторизації текстових даних – TF-IDF (від англ. TF — term frequency, IDF — inverse document frequency). Цей алгоритм являє собою статистичний показник, який оцінює, наскільки слово релевантне документу в наборі документів. Це робиться шляхом множення двох показників: скільки разів слово зустрічається в документі, і зворотної частоти цього слова в наборі документів. Якщо термін часто використовується в певному тексті, але рідко в інших, то він має велику значимість для даного тексту. Таким чином, слова, які є загальними в кожному документі, такі як «це», «що» і «якщо», мають низький рейтинг, навіть якщо вони можуть зустрічатися багато разів, оскільки вони не мають великого значення для цього документа зокрема. Однак, якщо слово «помилка» зустрічається багато раз в документі і не часто зустрічається в інших, це, ймовірно, означає, що воно дуже актуальне.

Вдаючись в деталі алгоритму TF-IDF, варто зазначити, що TF (Term Frequency - частота слова) - означає наскільки часто термін зустрічається в

документі. Показує відношення кількості згадувань слова до суми всіх слів на сторінці, тобто частотність слова, зокрема:

$$TF = \frac{n_i}{\sum_k n_k}, \tag{2.4}$$

де n_i – число входжень слова в документ, $\sum_k n_k$ – загальна кількість слів в документі.

IDF (Inverse Document Frequency - зворотна частота документа) - відношення всього числа документів до тих, які мають задане слово. Зменшує вагу слова в залежності від його частоти і показує релевантність тексту ключовому запиту:

$$IDF = log \frac{|D|}{|d_i \in t_i|}, \tag{2.5}$$

де $|D|$ - кількість документів в наборі, $|d_i \in t_i|$ - кількість документів, в яких зустрічається слово t_i (коли $n_i \neq 0$).

В результаті отримаємо значимість конкретного слова в межах одного тексту:

$$TF - IDF(t, d, D) = TF(t, d) \cdot IDF(t, D), \tag{2.6}$$

Задля наглядності практичної простоти використання алгоритму TF-IDF, наведено приклад, зокрема: взято документ з 10000 символів в якому слово "делегат" зустрічається 25 разів, а колекція складається з 2 мільйонів документів, у 2000 з яких також зустрічається дане слово. Тоді: $TF = \frac{25}{10000} = 0.0025$; $IDF = lg \frac{2000}{2000000} = lg \frac{1}{1000} = -3$(lg – логарифм за основою 10); $TF - IDF = 0.0025 \cdot 3 = 0.075$.

Переваги методу TF-IDF:

– простота розрахунку;

- містить базову метрику для вилучення найбільш описових термінів в документі;
- за допомогою базової метрики легко вираховується схожість між двома документами.

Недоліки методу TF-IDF:

- TF-IDF заснований на моделі пакета слів (BoW), тому він не фіксує позицію в тексті, семантику, збіги в різних документах. Даний метод обчислює зважений термін слова в документі на основі його частоти;
- з вище перелічених причин TF-IDF тільки корисний як функція лексичного рівня. Даний алгоритм не може фіксувати семантику (наприклад, в порівнянні з тематичними моделями).

2.3 Інформаційна структура системи

Розроблюваний програмний продукт призначений для бінарної класифікації заданого корпусу даних. В даному випадку, корпус даних являє собою файли, які містять в собі інформацію, подану в текстовому вигляді. Для того, щоб можна було «працювати» з цим текстом, його потрібно очистити. В більшості випадків процес очищення текстових документів зводиться до наступних дій:

- токенізація виразів;
- видалення лишніх слів (шум, «stop words»);
- нормалізація слів;
- векторизація тексту.

Токенізація - це процес розбиття фрагментів тексту на менші шматки.

Шумові слова, або «stop words» - це слова, які можуть мати сенс для людського спілкування, але не мають ніякого змісту для машинного навчання.

Нормалізація трохи складніша, ніж токенізація. Завданням нормалізації є перетворення всіх форм слова в єдине представлення цього слова. Наприклад, «спостерігали», «спостерігати», «спостерігаючи», «спостереження» можна

нормалізувати до "спостерігати". Існує два основних методи нормалізації: стеммінг (від англ. «stemming»), лематизація (від англ. «lemmatization»).

При використанні стеммінгу, кожне слово розділяється на «stem» - найменша частинка слова, з якої можна створити похідні слова (слова, які містять цю ж частинку). Стеммінг просто усікає слово, використовуючи спільні закінчення, тому зв’язок між словами «бачив» та «бачиш», наприклад, буде пропущений.

Лематизація вирішує проблему стеммінгу, пов’язану із втратою зв’язку між подібними словами. Цей процес використовує структуру даних, яка пов’язує всі форми слова із найпростішою, або лемою.

Діаграма процесу очищення корпусу даних (базового тексту) зображена на рисунку 2.5

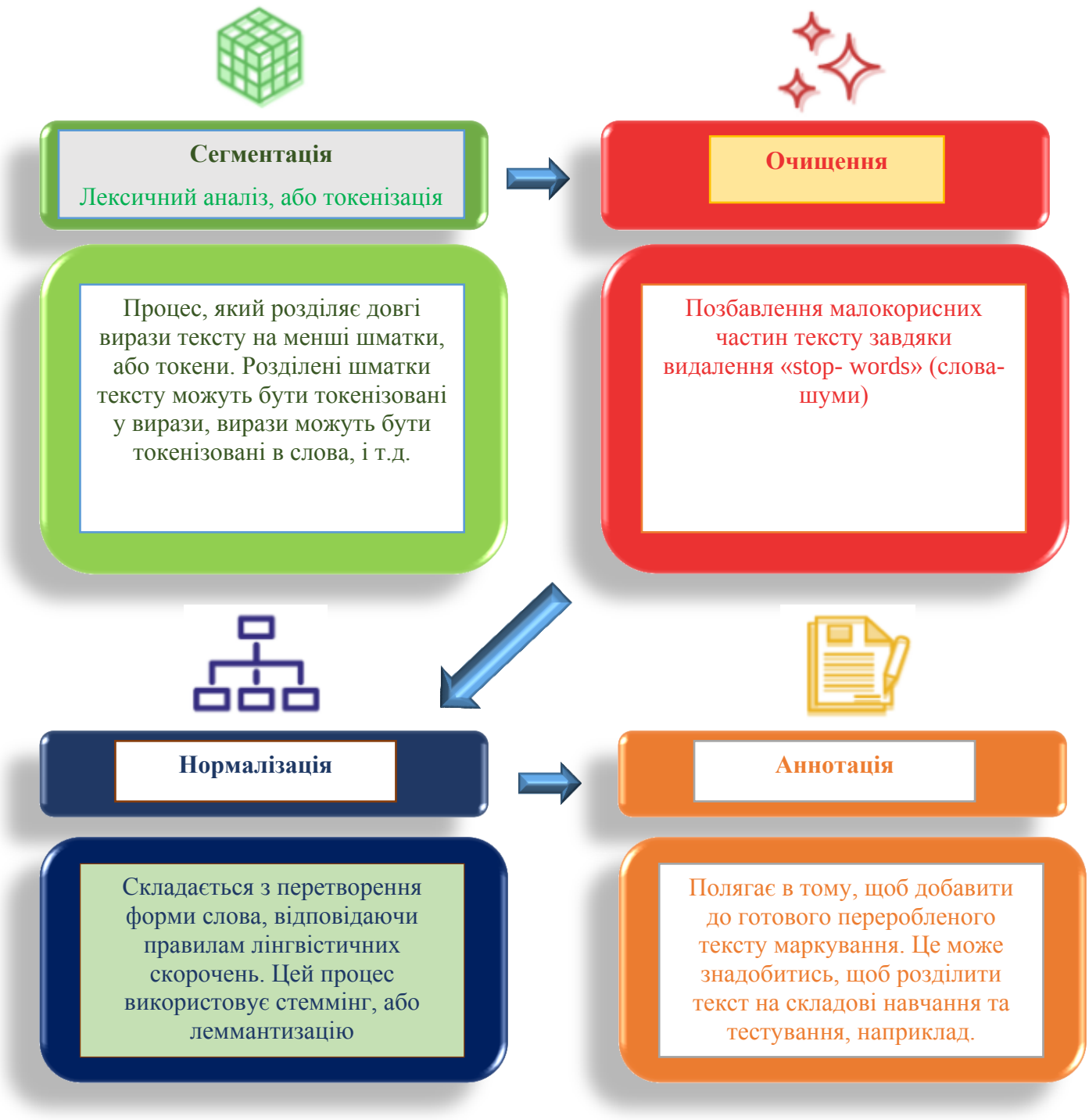


Рисунок 2.5 – Процес підготовки тексту (очищення)

В даному випадку, текст повинен класифікуватись на категорії «відноситься до заданої теми», «не відноситься до заданої теми». В якості базових класифікаторів використовуються наступні алгоритмічні моделі:

– Метод опорних векторів (Support vector machine);

- Наївний Баєсів класифікатор (Naive Bayes classifier);
- Метод k-найближчих сусідів (K-neighbors);
- Adaptive Boosting (AdaBoost);
- Випадковий ліс (Random forest);
- Логістична регресія (Logistic regression).

Метод опорних векторів (Support vector machine) - це алгоритм машинного навчання, який можна використовувати як для класифікації, так і для регресії. Однак він здебільшого використовується в задачах класифікації. В алгоритмі SVM кожен елемент даних інтерпретується як точка в n-мірному просторі (де n - кількість об’єктів, які в наявності), значення кожного об’єкта є значенням конкретної координати. Потім проводиться класифікація, знаходячи гіперплощину, яка добре диференціює два класи (рисунок 2.6).

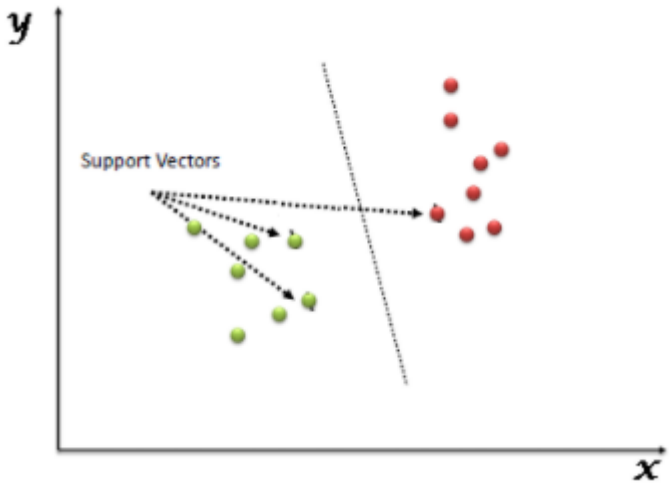


Рисунок 2.6 – результат роботи SVM

Наївний Баєсів класифікатор (Naive Bayes classifier) - це метод класифікації, заснований на теоремі Байєса з припущенням незалежності серед провісників. Простіше кажучи, класифікатор Байєса припускає, що наявність певної ознаки в класі не пов’язана з наявністю будь-якої іншої ознаки. Наприклад, фруктом можна вважати яблуко, якщо воно червоне, кругле і має діаметр близько 8 сантиметрів. Навіть якщо ці ознаки залежать одна від одної або від існування інших ознак, усі ці

властивості незалежно сприяють імовірності того, що цей фрукт є яблуком. Модель Наївного Байєса проста в побудові і особливо корисна для дуже великих наборів даних. Поряд із простотою, як відомо, цей підхід перевершує навіть дуже складні методи класифікації. Теорема Байєса забезпечує спосіб обчислення апостеріорної ймовірності $P\ (c \mid x)$ з $P\ (c)$, $P\ (x)$ та $P\ (x \mid c)$. Формула наведена нижче:

$$P\ (c \mid x)\ =\ \frac{P\ (x \mid c)P\ (c)}{P\ (x)}, \tag{2.7}$$

де $P\ (c \mid x)$ - це апостеріорна ймовірність класу (c, цілі), заданого предиктором (x, атрибути);

$P\ (c)$ - апріорна ймовірність класу;

$P\ (x \mid c)$- це ймовірність, яка є ймовірністю передбачувача даного класу;

$P\ (x)$ - апріорна ймовірність предиктора.

З формули (2.7) можна вивести наступну закономірність:

$$P\ (c \mid x) =\ P\ (x_1 \mid c)\ \times\ P\ (x_2 \mid c) \times \ldots \times\ P\ (x_n \mid c)\ \times\ P\ (c) \tag{2.8}$$

Метод k-найближчих сусідів (K-neighbors) визначає схожість між новим випадком / даними та наявними кейсами, після чого відносить новий кейс до категорії, найбільш схожої на наявні категорії. Алгоритм KNN на етапі навчання просто зберігає набір даних, і коли він отримує нові дані, то класифікує ці дані за категорією, яка схожа на нові дані.

Роботу K-NN можна пояснити на основі наведеного нижче алгоритму:

1. Вибирається число К сусідів.
2. Обчислюється евклідову відстань К кількості сусідів;
3. Вибирається К найближчих сусідів відповідно до розрахункової евклідової відстані.

4. Серед цих k сусідів підраховується кількість точок даних у кожній категорії.
5. Призначаються нові точки даних тій категорії, для якої кількість сусідів максимальна.

Припустимо, у нас є нова точка даних, і нам потрібно віднести її до потрібної категорії, рисунок 2.7.

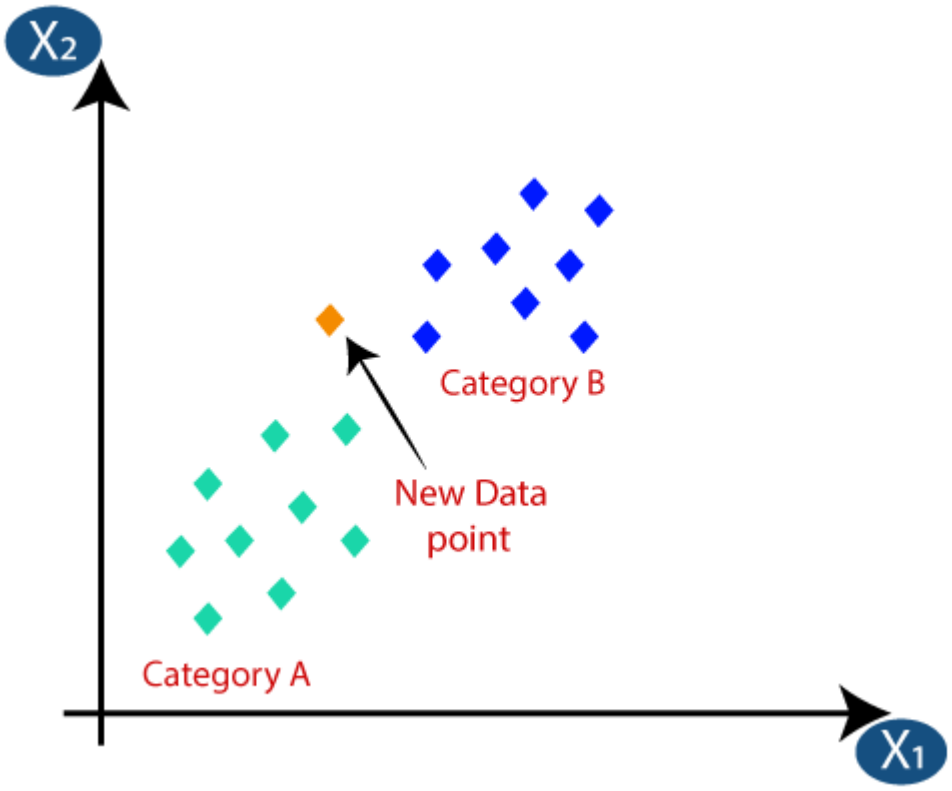


Рисунок 2.7 – Приклад даних

По-перше, виберемо кількість сусідів k = 5.

Далі ми обчислимо евклідову відстань між точками даних. Евклідова відстань - це відстань між двома точками, яка була наведена вище у цьому розділі.

Розрахувавши евклідову відстань, ми отримали найближчих сусідів: 3 найближчих сусіда категорії A та два найближчих сусіда категорії B, рисунок 2.8.

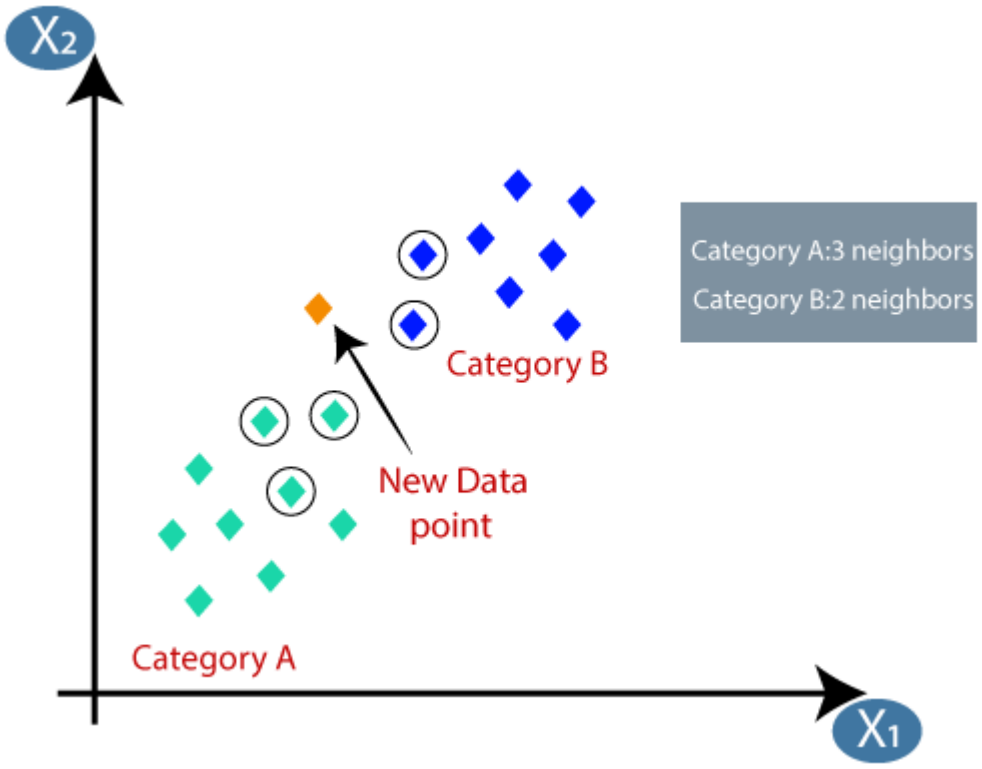


Рисунок 2.8 – кількість найближчих сусідів нової точки

Як ми бачимо, 3 сусіди належать до категорії А, 2 до категорії В, отже, ця нова точка даних повинна належати до категорії А.

Adaptive Boosting (AdaBoost) - це один із підсилювальних ансамблів класифікаторів, запропонованих Йоавом Фрейндом та Робертом Шапіром у 1996 році. Він поєднує кілька класифікаторів для підвищення точності класифікації. AdaBoost - це ітеративний метод ансамблю. Класифікатор AdaBoost створює сильний класифікатор, поєднуючи кілька неефективних класифікаторів. Основна концепція Adaboost полягає у встановленні ваги класифікаторів та підготовці вибірки даних у кожній ітерації таким чином, щоб вона забезпечувала точні прогнози незвичних спостережень. Будь-який алгоритм машинного навчання може бути використаний як базовий класифікатор, якщо він приймає ваги на навчальному наборі. Adaboost повинен відповідати двом умовам:

– класифікатор повинен навчатись інтерактивно на різних прикладах навчання із різними ваговими коефіцієнтами;

– на кожній ітерації повинне забезпечуватись тренування на вхідних даних так, щоб мінімізувати помилки в навчанні.

Алгоритм роботи класифікатора AdaBoost можна описати наступними кроками:

1. Спочатку Adaboost довільно підбирає навчальну підмножину.
2. AdaBoost ітеративно навчається, вибираючи навчальний набір на основі точного прогнозу останнього навчання.
3. Неправильно класифікованим спостереженням призначаються вищі вагові коефіцієнти, так що в наступній ітерації ці спостереження отримають високу ймовірність класифікації.
4. Крім того, вага тренованого класифікатора призначається в кожній ітерації відповідно до точності класифікатора. Більш точний класифікатор отримає більший ваговий коефіцієнт вагу.
5. Цей процес повторюється до тих пір, поки класифікація навчених моделей на навчальних даних не досягне максимуму точності, або до досягнення вказаної максимальної кількості ітерацій навчання.

Схема роботи алгоритму Adaptive Boosting зображена на рисунку 2.9.

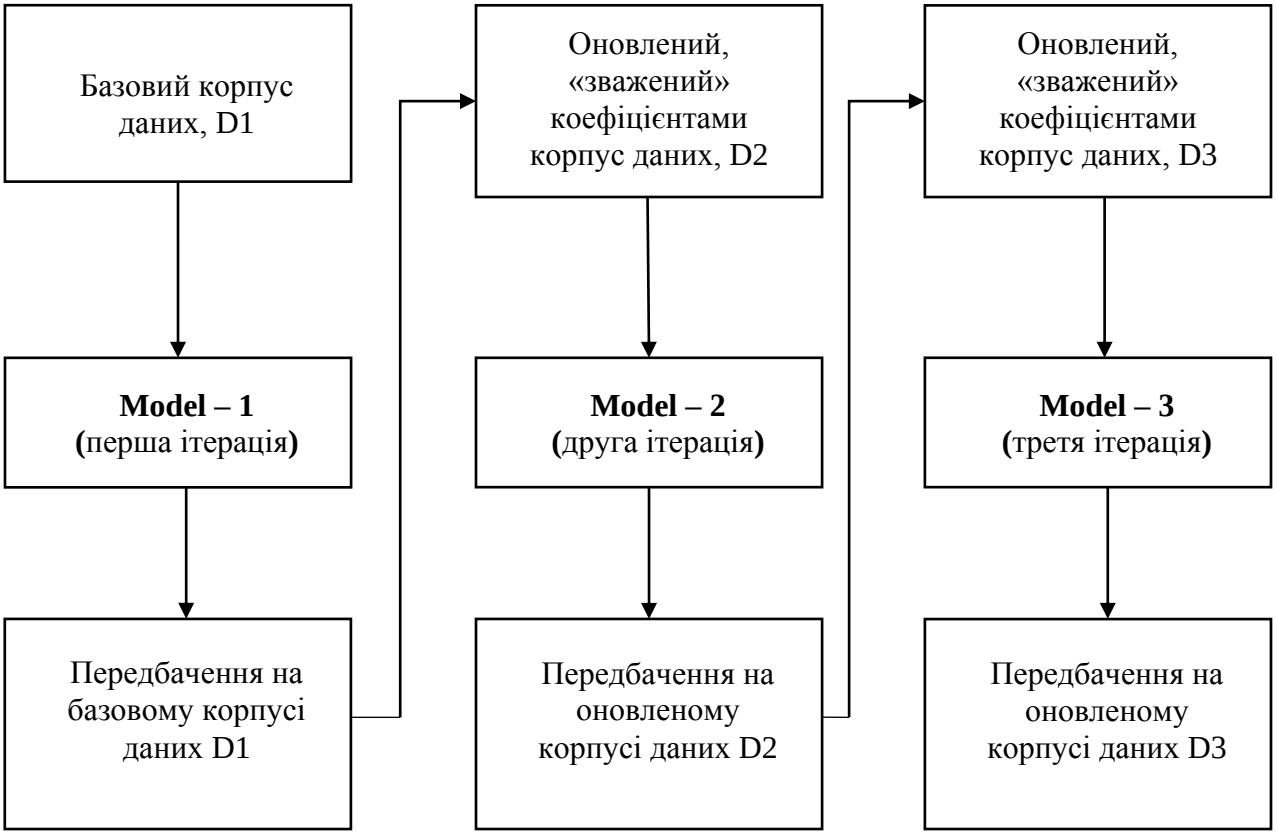


Рисунок 2.9 – Схема роботи класифікатора AdaBoost

Випадковий ліс (Random forest) є класифікатором, який містить ряд дерев рішень для різних підмножин заданого набору даних, він бере середнє значення рішень дерев для підвищення точності прогнозування цього набору даних. Він базується на концепції ансамблевого навчання, що являє собою процес поєднання декількох класифікаторів для вирішення складної проблеми та підвищення ефективності роботи моделі. Випадковий ліс бере прогноз з кожного дерева, після чого на основі більшості голосів прогнозів, він передбачає кінцевий результат. Більша кількість дерев у лісі призводить до вищої точності.

Діаграма, яка зображена на рисунку 2.10, показує принцип роботи класифікатора Random forest.

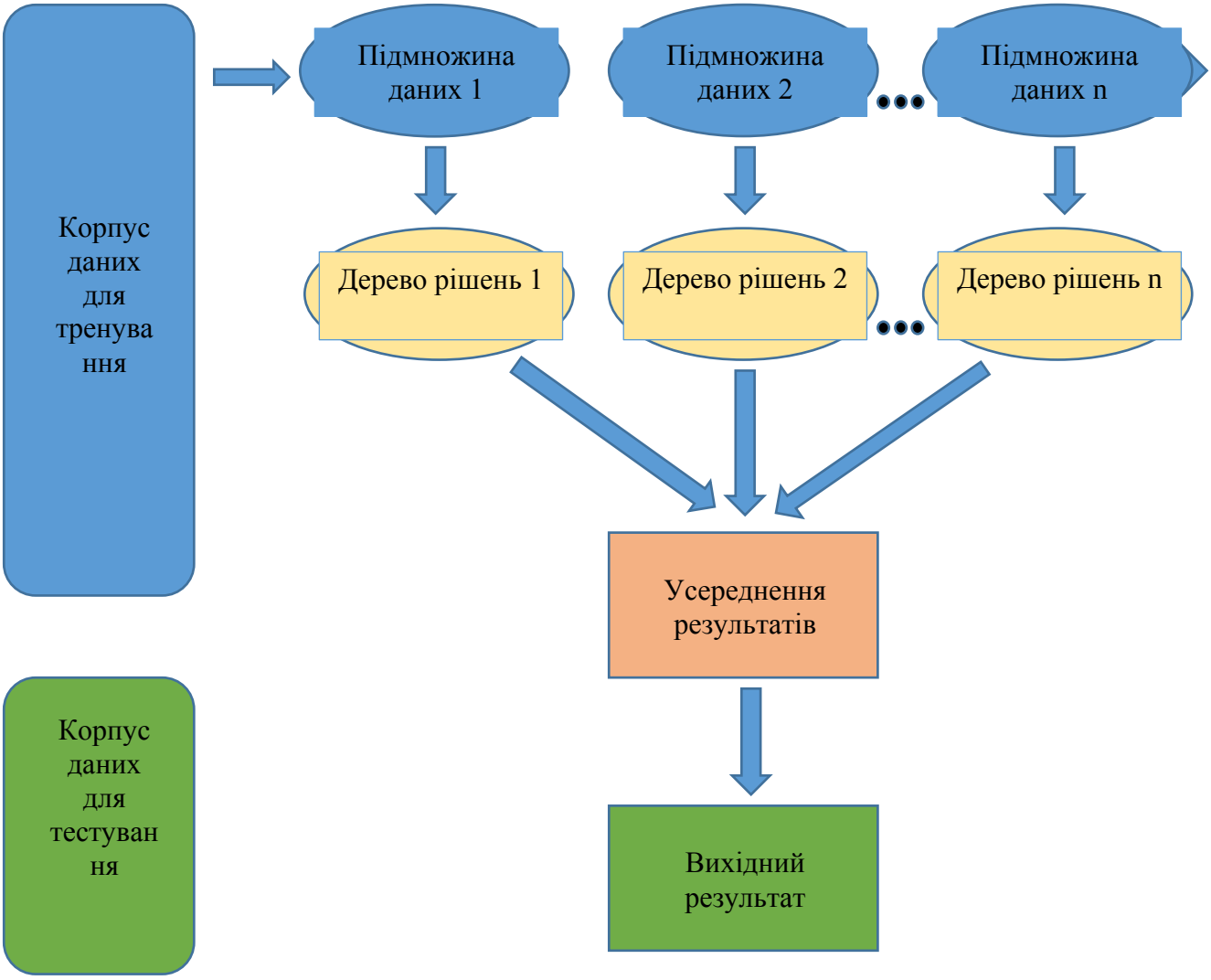


Рисунок 2.10 - принцип роботи класифікатора Random forest

Оскільки випадковий ліс поєднує кілька дерев для прогнозування класу набору даних, можливо, що деякі дерева рішень можуть передбачати правильний результат, а інші - ні. Але разом усі дерева прогнозують правильний результат. Отже, нижче є два припущення щодо кращого класифікатора:

- у змінній ознаки набору даних повинні бути деякі фактичні значення, щоб класифікатор міг передбачити точні результати, а не приблизні;
- передбачення кожного дерева повинні мати дуже низькі кореляційні зв'язки.

Random forest працює в два етапи: спочатку створюються випадковий ліс шляхом об’єднання N дерев рішень, потім робиться прогноз для кожного дерева, створеного на першому етапі.

Робочий процес даного алгоритму можна описати наступними кроками:

1. Вибираються випадкові точки даних K з набору тренувань.
2. Створюються дерева рішень, які пов’язані із вибраними точками даних (підмножини).
3. Вибирається число N дерев рішень, які потрібно побудувати;
4. Повторення кроків 1 та 2.
5. Для нових точок даних робляться прогнози кожного дерева рішень, після цього нові точки даних відносяться до категорії, яка набирає більшість голосів.

Логістична регресія (Logistic regression) вимірює взаємозв'язок між залежною змінною (наша мітка, те, що ми хочемо передбачити) та однією або кількома незалежними змінними (властивості навчального набору даних), оцінюючи ймовірності, використовуючи логістичну функцію, що лежить в основі. Результатом цього класифікатора є ймовірності відношення одиниці даних до одного чи іншого кластеру, ці ймовірності повинні бути перетворені у двійкові значення, щоб зробити конкретний прогноз. Це завдання логістичної функції, яку також називають сигвидною функцією. Сигмовидна функція - це S-подібна крива, яка може приймати будь-яке дійсне число та відображати його у значення між діапазоном від 0 до 1. Потім ці значення між 0 та 1 будуть перетворені в 0 або 1 за допомогою порогового значення.

На рисунку 2.11 зображені кроки, через які проходить алгоритм логістичної регресії, щоб видати результат класифікації.

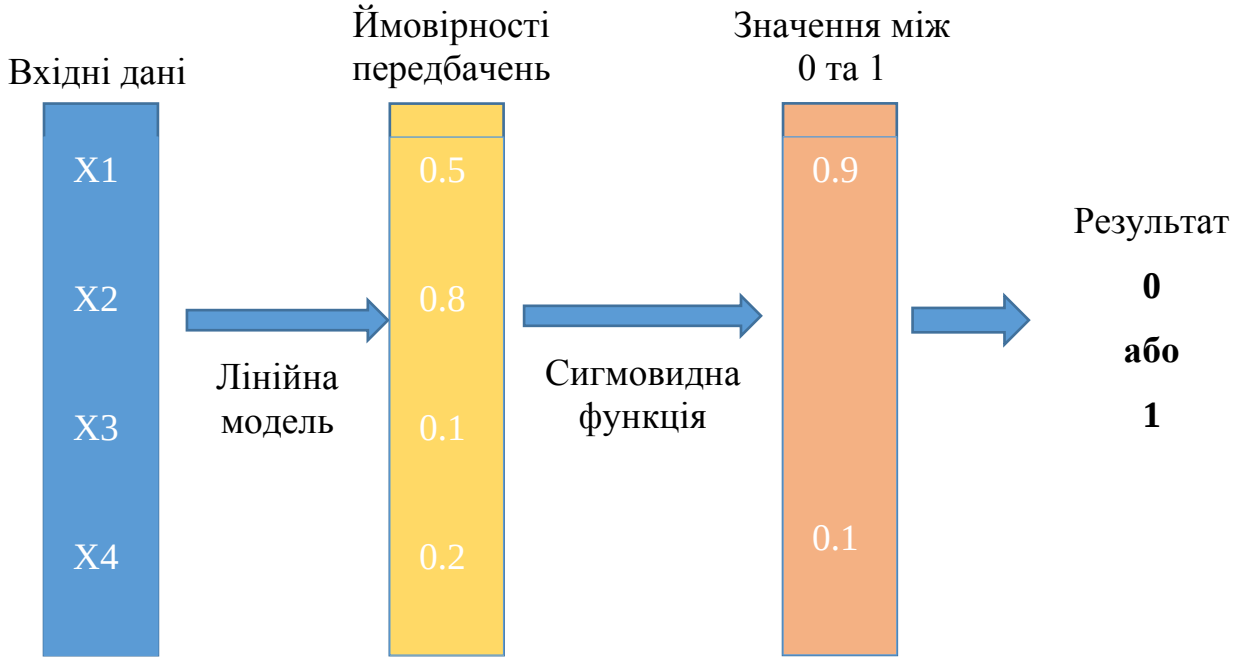


Рисунок 2.11 - Кроки, через які проходить алгоритм логістичної регресії, щоб видати результат класифікації

Не зважаючи на те, що вище були описані базові класифікатори, які використовуються в поточному дослідженні застосування агрегативних підходів для класифікації на базі ансамблевих моделей, застосунок реалізований так, що використання базових класифікаторів може бути змінюване, тобто алгоритми, які застосовуються для класифікації тексту можуть бути змінені на інші, або до базового набору алгоритмів класифікації уможливлено додавання нових. Опція заміни, або додавання нових «примітивів» в ансамбль дозволяє проводити експерименти, ціль який полягає в тому, щоб знайти найбільш оптимальний набір базових моделей, при якому точність всього ансамблю (мета алгоритму) буде найвища.

На рисунку 2.12 зображена загальна схема роботи програми.

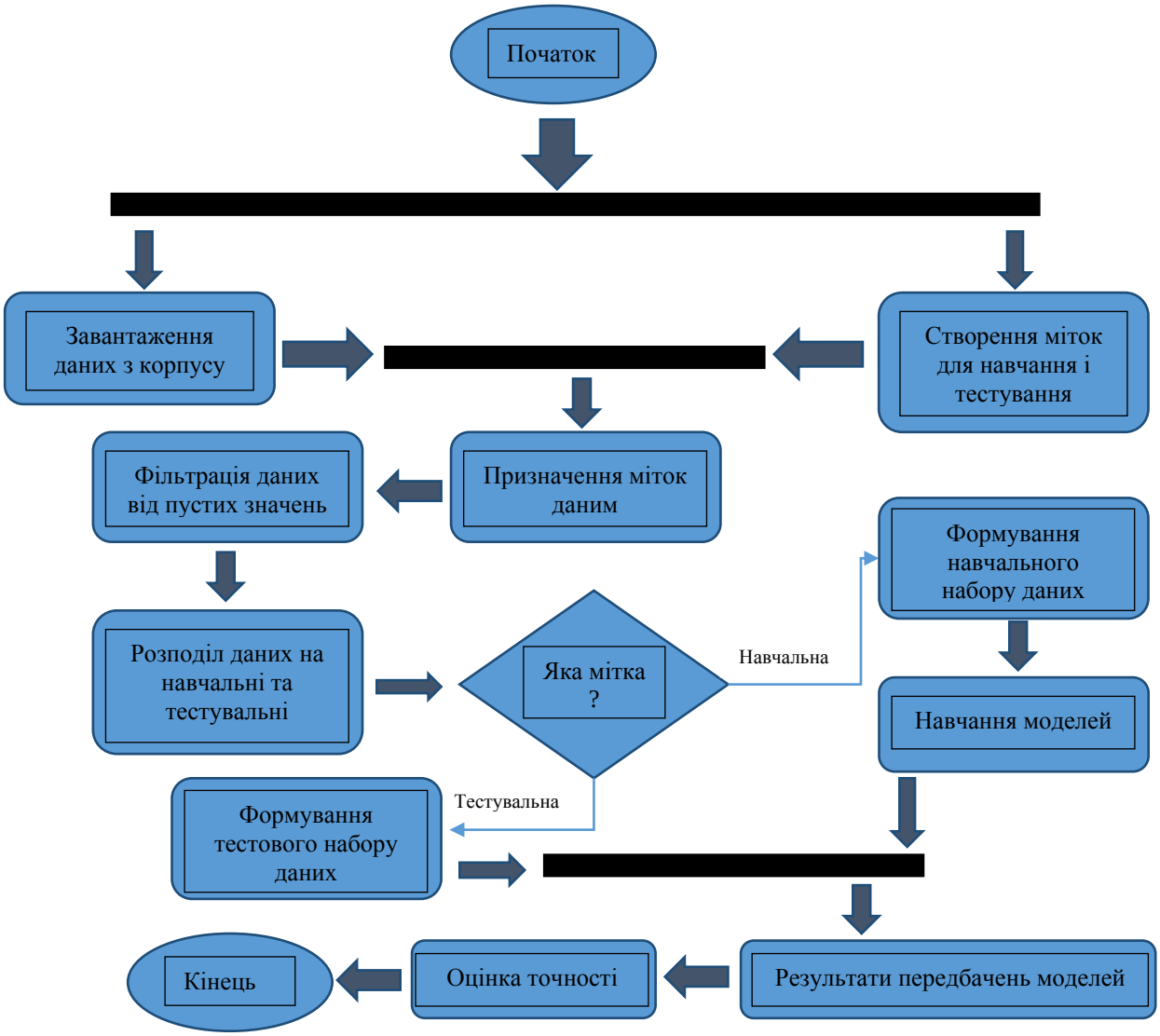


Рисунок 2.12 – Узагальнена схема роботи програми

Завданням системи є не тільки класифікація на основі передбачень базових алгоритмів класифікації, потрібно щоб кінцевий результат порівняння істинних значень даних з корпусу і передбачень моделей був якомога точніший.

Одним із підходів, які використовуються для підвищення кваліфікації базових алгоритмів, є агрегатування. Задля оптимізації рішень базових моделей машинного навчання, створюються унікальні комбінації класифікаторів. Кожна комбінація - це по суті набір алгоритмів класифікації, суть полягає в тому, щоб не було однакових наборів. Залежність кількості комбінацій від кількості базових моделей зображена на рисунку 2.13

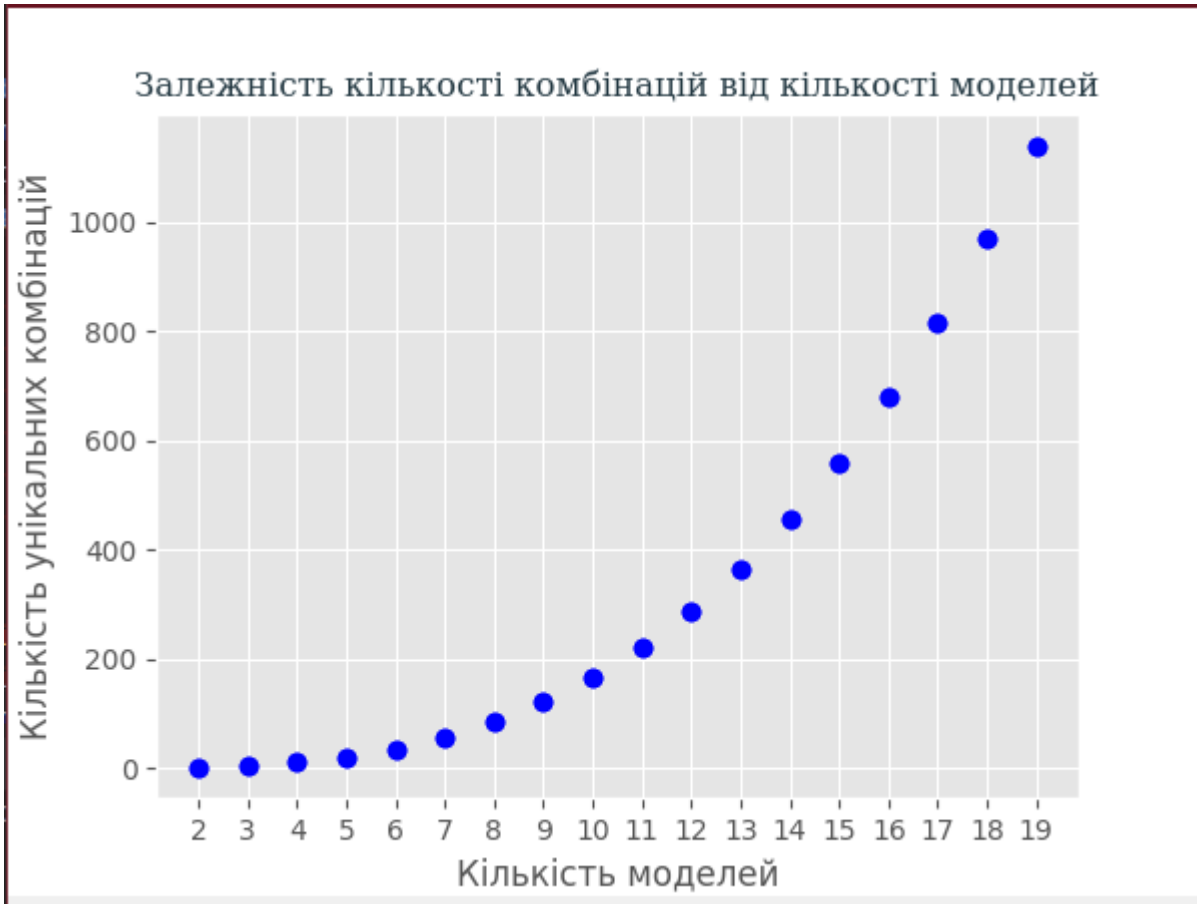


Рисунок 2.13 – Кількість комбінацій

Після того, як унікальні набори базових класифікаторів створені, потрібно добавити модуль оптимізації передбачень для кожного набору. Тобто, рішення кожної базової моделі в наборі (комбінації) повинно порівнятись із рішеннями інших моделей в даному наборі. В кінцевому результаті, оптимізація буде проводитись тільки над тими передбаченнями, які не були однакові для заданої комбінації моделей – цей процес називають також як підвищення точності.

Підвищення точності кожного набору моделей проводиться з допомогою такого алгоритма кластеризації, як k-середніх (k-means) – цей алгоритм описується вище в поточному розділі. Передбачення, які не були однакові для заданої комбінації моделей, використовуються в якості вхідних параметрів алгоритму k-means, далі відбувається знаходження центроїдів кластерів для заданих вхідних даних. Кількість кластерів встановлена в кількості двох. Залежно від величини

середнього значення центроїдів, встановлюється відповідна відповідь для даного набору моделей.

Однією з головних цілей дослідження є візуалізація результатів. В якості результатів повинні бути наочно представлені основні метрики для даної області класифікації. Для того, щоб відображати всі ці метрики, добавлено модуль побудови графіків у двох та трьох-вимірному просторі. Даний модуль також використовується для того, щоб відображати коефіцієнти кореляцій між корпусом тестових даних та передбаченнями базових моделей.

Коефіцієнт кореляції є статистичним показником сили взаємозв'язку між відносними закономірностями двох змінних. Значення коливаються від -1,0 до 1,0. Розраховане число більше 1,0 або менше -1,0 означає, що під час розрахунку кореляції виникла помилка. Кореляція -1,0 показує ідеальну негативну кореляцію, тоді як кореляція 1,0 - ідеальну позитивну кореляцію. Кореляція 0,0 показує відсутність лінійної залежності між закономірностями двох змінних.

Негативна кореляція - це зв’язок між двома змінними, в яких одна змінна збільшується в міру зменшення іншої, і навпаки.

Позитивна кореляція - це взаємозв'язок між двома змінними, в яких обидві змінні рухаються в тандемі - тобто в одному напрямку. Можна сказати, що при позитивній кореляції одна змінна зменшується, коли інша змінна зменшується, або одна змінна збільшується, коли інша збільшується.

Міцність зв'язку варіюється, залежить від значення коефіцієнта кореляції. Наприклад, значення 0,2 показує, що існує позитивна кореляція між двома змінними, але вона є слабкою і, ймовірно, неважливою. Аналітики деяких галузей дослідження не вважають кореляцію важливою, поки значення не перевищить щонайменше 0,8. Однак коефіцієнт кореляції з абсолютним значенням 0,9 або більше буде дуже сильним взаємозв’язком.

Виходячи з вище написаного, коефіцієнт кореляції являється необхідною метрикою, яка показує, наскільки «толковою» є базова модель. Доцільність використання класифікатора оцінюється міцністю кореляції між його

передбаченнями і правильними відповідями, зокрема, чим вищий коефіцієнт кореляції, тим точнішим являється класифікатор.

2.4 Вибір засобів розробки інформаційної системи

2.4.1 Вибір мови програмування

Виходячи із поставленого завдання, обрана мова програмування Python [31]. Це популярна мова програмування, і отримала вона свою популярність завдяки тому, що за допомогою цієї мови можна вирішувати наступні проблеми:

- WEB-розробка;
- розробка програмного забезпечення;
- реалізація математичних моделей;
- машинне навчання;
- Deep-Learning.

Проекти ШІ відрізняються від традиційних програмних проектів. Відмінності полягають у наборі технологій, навичках, необхідних для проекту на основі ШІ, та необхідності глибоких досліджень. Для реалізації необхідних функцій штучного інтелекту слід використовувати мову програмування, яка є стабільною, гнучкою та має доступні інструменти. Python пропонує все це, саме тому сьогодні існує багато проектів AI на Python.

Від розробки до розгортання та обслуговування проекту Python допомагає розробникам бути продуктивними та впевненими в тому, яке програмне забезпечення вони створюють. Переваги, які роблять Python найкращим чином придатним для машинного навчання та проектів на основі ШІ, включають простоту та послідовність, доступ до чудових бібліотек та фреймворків для ШІ та машинного навчання (ML), гнучкість, незалежність від платформи та широке співтовариство. Це додає загальної популярності мови.

Причини, чому Python є найкращою мовою для машинного навчання:

1) це легка мова для розуміння, яка дозволяє швидко перевіряти дані;
2) Python має чудову екосистему бібліотек;

3) низький бар'єр для входу;
4) це дуже гнучка мова;
5) програми, написані на Python, легко читати, оскільки ця мова високорівнева

Єдиним суттєвим недоліком мови програмування Python для застосування його в області математичного аналізу та статистики, які часто застосовуються в алгоритмах машинного навчання є проблеми з багато потоковими програмами: Python не підтримує потокову передачу через Global Interpreter Lock, тобто GIL, який є мьютексом; це дозволяє виконувати одночасно лише один потік. Багато потокові програми, пов'язані з процесором, можуть бути повільнішими, ніж однопотокові; це питання можна вирішити шляхом впровадження багатопроцесорних програм замість багато потокових програм.

Судячи із вище написаного, Python – це та мова, яка відповідає всім вимогам, які поставлені в даному дослідженні, тому питання вибору вважається закритим.

2.4.2 Вибір засобу підтримки алгоритмів машинного навчання

Задля роботи ансамблю, мета якого покращити точність передбачень, він повинен містити в собі низку базових (примітивних) алгоритмів класифікації. Реалізація базових класифікаторів міститься в бібліотеці scikit-learn. Sklearn або scikit-learn у Python на сьогоднішній день є однією з найкорисніших бібліотек із відкритим кодом, які ви можете використовувати для машинного навчання в Python. Бібліотека scikit-learn - це вичерпна колекція найефективніших інструментів статистичного моделювання та машинного навчання. Деякі з цих інструментів включають регресію, класифікацію, зменшення розмірності та кластеризацію.

Переваги Sklearn:

- бібліотека поширюється за ліцензією BSD, що робить її безкоштовною з мінімальними юридичними та ліцензійними обмеженнями;
- проста у використанні;

- бібліотека scikit-learn дуже універсальна та зручна і служить реальним цілям, таким як прогнозування поведінки споживачів, створення нейрозображень тощо;
- Scikit-learn підтримується та оновлюється численними авторами, співавторами та величезною міжнародною спільнотою;
- веб-сайт scikit-learn забезпечує розгорнуту документацію щодо API для користувачів, які хочуть інтегрувати алгоритми зі своїми платформами.

Недоліки:

- це не найкращий вибір для розробки продуктів, які використовують «глибоке» навчання

Зваживши переваги та недоліки бібліотеки, описаної вище, можна сказати, що це найкращий вибір серед наявних у мові програмування Python.

2.4.3 Вибір засобу візуалізації результатів дослідження

Візуалізація інформації завжди ставиться у високий пріоритет в будь-якому дослідженні, оскільки складні зв’язки між елементами даних найкраще прослідковуються в наочному поданні. В даному дослідженні застосування агрегативних підходів потрібно реалізувати систему відображення основних метрик класифікаційної системи у двох-вимірному просторі у вигляді графіків. Додатковим завданням виступає модуль відображення кореляційних властивостей базових моделей класифікації із тестовими даними. Коефіцієнти кореляцій повинні відображатись у формі гістограм для двох та трьох-вимірного простору.

Для поставлених завдань було вирішено використовувати графічний рушій, який міститься у бібліотеці для Python, Matplotlib [32].

Matplotlib - це між платформна бібліотека для візуалізації даних та графічного побудови графіків для Python та її числового розширення NumPy. Таким чином, ця бібліотека пропонує життєздатну альтернативу MATLAB з відкритим

кодом. Розробники також можуть використовувати API matplotlib (Інтерфейси прикладного програмування) для вбудовування графіків у графічні програми.

Сценарій matplotlib Python побудований таким чином, що кілька рядків коду - це все, що потрібно в більшості випадків для створення візуального графіку даних. Шар сценарію matplotlib накладає два API:

- API pyplot - це ієрархія об’єктів коду Python, які містить matplotlib.pyplot;
- колекція об'єктів об'єктно-орієнтованого API, яка може бути зібрана з більшою гнучкістю, ніж pyplot. Цей API забезпечує прямий доступ до внутрішніх рівнів Matplotlib.

API pyplot має зручний інтерфейс з відстеженням стану в стилі MATLAB. Фактично, matplotlib спочатку був написаний як альтернатива MATLAB з відкритим вихідним кодом. ОО (об’єктно орієнтоване) API і його інтерфейс має більше параметрів і потужний, ніж pyplot, але вважається більш складним у використанні. В результаті інтерфейс pyplot використовується частіше.

Розуміння API pyplot matplotlib є ключовим для розуміння того, як працювати з графіками:

- matplotlib.pyplot.figure: Figure - контейнер верхнього рівня. Він включає все, що візуалізується в графіку, включаючи одну або кілька Осей.
- matplotlib.pyplot.axes: Axes містять більшість елементів у графіку: Axis, Tick, Line2D, Text, тощо, та встановлює координати. Це область, на яку наносяться дані. До Axes належать також Вісь Axes X, Axes Y і, можливо, Axes Z.

Оскільки під час роботи інформаційної системи класифікації даних створюється велика кількість комбінацій базових алгоритмів, взаємозв’язки кожного класифікатора у кожній комбінації повинні відображатись також. Щоб розмістити таку велику кількість графіків, яка дорівнює кількості комбінацій моделей, потрібно використовувати полотно, на якому будуть розміщені ці всі графіки, в якості полотна використовується об’єкт Tkinter [33].

Розділ 3

Програмна реалізація інформаційної системи

3.1 Структура і функціональне призначення модулів системи

Під час реалізації програмного засобу, було реалізовано структуру, яка відповідає поставленому завданню. Діаграма класів зображена на рисунку 3.1

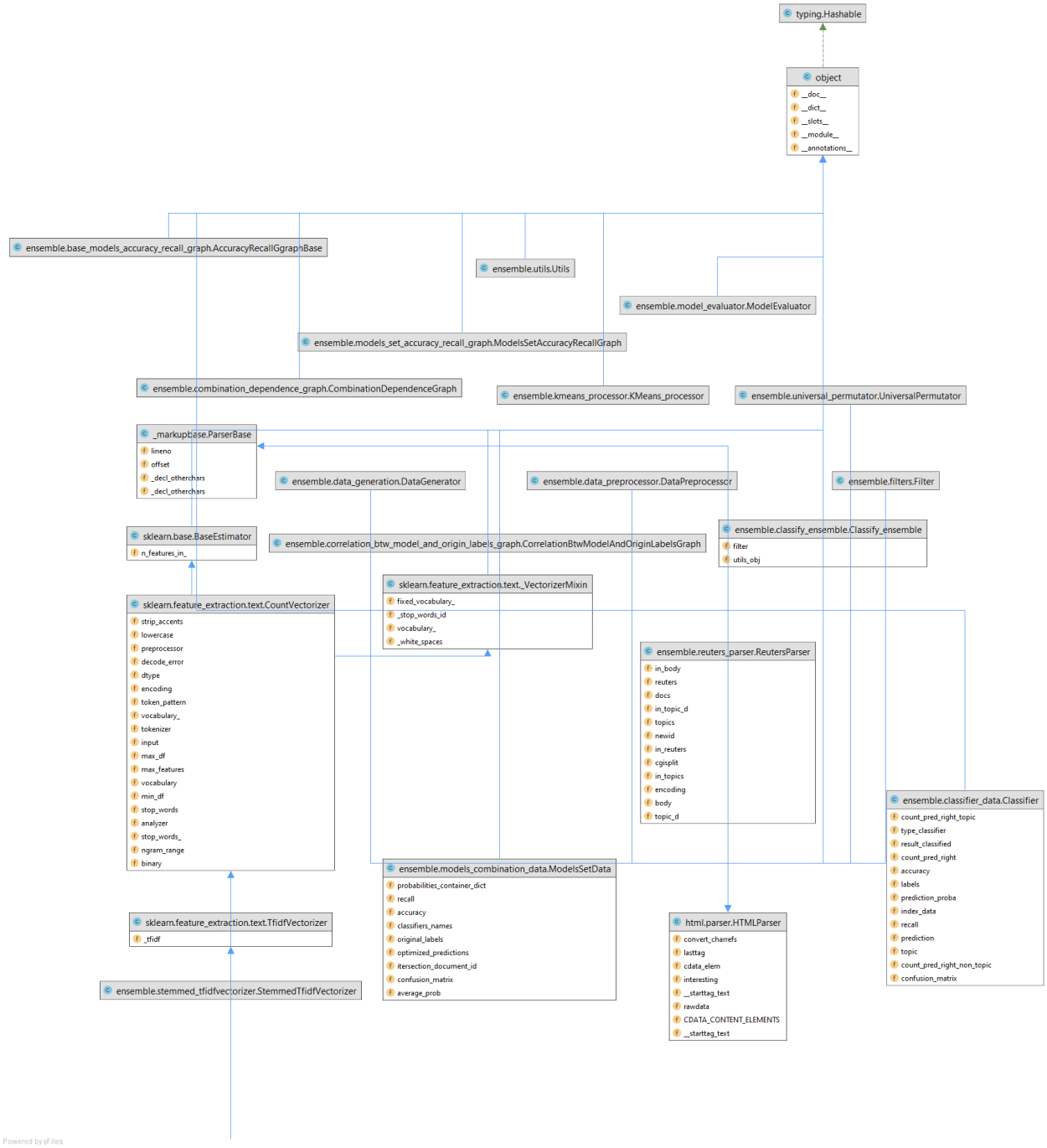


Рисунок 3.1 – діаграма класів

Клас «ReutersParser», який наслідується від класу «HTMLParser», дозволяє завантажувати дані з корпусу даних (файлів) і розбивати їх на окремі сегменти. Це зроблено для того, щоб вибрати тільки ті документи, в яких є необхідні блоки тексту.

Класи «DataGenetator», «DataPreprocessor» відфільтровують завантажені дані, розбивають вхідні дані на вибірки для навчання та тестування моделей, навчають «vectorizer», «tf-idftransformer»[14][15].

Клас «Filter» розмічає вхідні дані по темах, та видаляє пусті документи.

Клас «ClassifyEnsemble» являється одним із найбільших в програмному продукті, від відповідіє за навчання моделей, тестування моделей.

Клас «KMeans_processor» використовується в якості модуля кластеризації передбачень базових моделей. Саме в цьому класі відбувається оптимізація передбачень класифікаторів.

Клас «ModelEvaluator» призначений для оцінювання основних метрик класифікаторів, таких як точність, повнота, матриця конфузій [16].

Класи, які являються контейнерами для даних окремих класифікаторів, або контейнерами для даних комбінації моделей: «ClassifierData», «Model», «ModelsCombinationData».

За створення графіків залежності комбінацій від кількості базових моделей, точності базових моделей, точності комбінацій, та графіків, які показують коефікаєнти кореляції у двох, трьох-вимірному просторі, відповідають наступні класи: «AccuracyRecallGgraphBase», «CombinationDependenceGraph»,«CorrelationBtwModelAndOriginLabelsGraph», «ModelsSetAccuracyRecallGraph».

Важливо зауважити, що коефіцієнти кореляцій у двох, трьох-вимірному просторі будуються для обох випадків: статичний, динамічний. Коефіцієнт кореляції в статиці обраховується відносно сталого набору документів для кожного класифікатора, а от динамічний коефіцієнт кореляції обраховується на тих документах, які не входять до «ядра» передбачень кожного набору моделей. Ядром передбачень вважається така множина документів, на яких була дана однакова

відповідь класифікаторів у наборі. Отож, «ядро» передбачень змінюється відносно кожного універсального набору класифікаторів.

Узагальнена діаграма проекту зображена на рисунку 3.2.

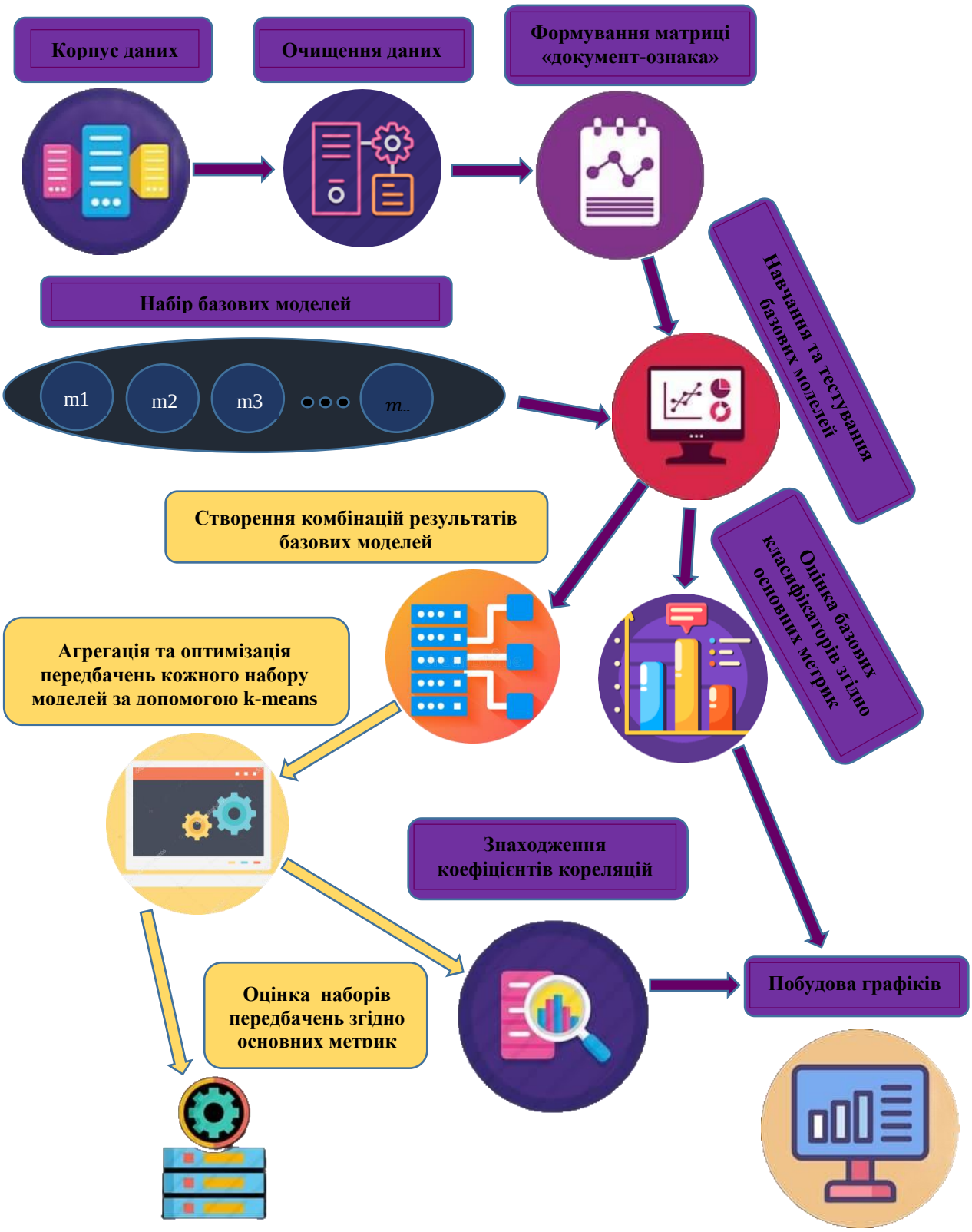


Рисунок 3.2 – Узагальнена діаграма проекту

3.2 Тестування інформаційної системи

Для перевірки коректної роботи застосунку було проведене його модульне тестування (unit testing). Модульне тестування - це вид тестування програмного забезпечення, де тестуються окремі блоки або компоненти програмного забезпечення. Метою є перевірити, що кожна одиниця програмного коду працює належним чином. Модульне тестування проводиться під час розробки (фази кодування) програми розробниками. Модульні тести ізолюють розділ коду та перевіряють його правильність. Одиницею може бути окрема функція, метод, процедура, модуль або об’єкт.

Модульне тестування важливе. Існує міф, що розробники програмного забезпечення іноді намагаються заощадити час, виконуючи мінімальне модульне тестування – це не достовірна інформація, тому що невідповідне модульне тестування призводить до високої вартості виправлення дефектів під час тестування системи, тестування інтеграції і навіть бета-тестування після створення програми. Якщо належне модульне тестування проводиться на ранній стадії розробки, в кінцевому підсумку це економить час і гроші.

Ось основні причини для проведення модульного тестування в програмній інженерії:

- модульні тести допомагають виправити помилки на початку циклу розробки та заощадити витрати.;
- це допомагає розробникам зрозуміти базу коду тестування та дозволяє швидко вносити зміни;
- хороші модульні тести служать проектною документацією;
- модульні тести допомагають з повторним використанням коду. Можна перенести код і тести в новий проект, після чого Змінювати код, поки тести не запустяться знову.

Інформаційна система класифікації даних на базі ансамблевий моделей включає в себе чимало модулів, для основних з яких були зроблені наступні модульні тести:

1) тест модуля завантаження даних;
2) тест фільтрування даних;
3) тест навчання базових класифікаторів;
4) тест дампу навчених моделей;
5) тест завантаження дампу;
6) тест модуля тестування навчених класифікаторів;
7) тест модуля створення комбінацій;
8) тест модуля оцінки наборів класифікаторів;
9) тест функцій побудови двох, трьох-вимірних графіків (декілька різних тестів).

Окрім модульних тестів, система також підтримує логування (logging) для зручного відслідковування процесу виконання. Результат зображений на рисунках 3.3, 3.4.

```
With remove stopwords and stemming
Teaching of classifiers starts

---------- Labels  earn non-earn ----------
Using svm Classifier for teach

Training with svm...
Training time 80.95 seconds.
Using naive-bayes Classifier for teach

Training with naive-bayes...
Training time 0.03 seconds.
Using k-neighbors Classifier for teach

Training with k-neighbors...
Training time 0.02 seconds.
Using ada-boost Classifier for teach

Training with ada-boost...
Training time 28.53 seconds.
Using random-forest Classifier for teach
```

Рисунок 3.3 – Фрагмент логування

```
Teaching of classifiers takes 283.77 seconds.

Testing of classifiers starts

---------- Labels  earn non-earn ----------
Using svm Classifier for test

Testing time of svm takes 7.26 seconds.
Saved to file...
Using naive-bayes Classifier for test

Testing time of naive-bayes takes 2.51 seconds.
Saved to file...
Using k-neighbors Classifier for test

Testing time of k-neighbors takes 4.06 seconds.
Saved to file...
Using ada-boost Classifier for test

Testing time of ada-boost takes 2.39 seconds.
Saved to file...
Using random-forest Classifier for test
```

Рисунок 3.4 - Фрагмент логування

Результат проходження модульних тестів зображений на рисунку 3.5

```
>>> ____Start__Testing____
... test_data_generation (__main__.Unit_Testing) ... ok
... test_filters (__main__.Unit_Testing) ... ok
... test_teach_classifiers (__main__.Unit_Testing) ... ok
... test_dump_teached_data_save (__main__.Unit_Testing) ... ok
... test_dump_teached_data_load (__main__.Unit_Testing) ... ok
... test_test_classifiers (__main__.Unit_Testing) ... ok
... test_permutate_universally (__main__.Unit_Testing) ... ok
... test_evaluate_by_important_metrics (__main__.Unit_Testing) ... ok
... test_build_static_correlation_graphs2d (__main__.Unit_Testing) ... ok
... test_build_dynamic_correlation_graphs2d (__main__.Unit_Testing) ... ok
... test_build_dynamic_correlation_graphs3d (__main__.Unit_Testing) ... ok
... test_build_static_correlation_graphs3d (__main__.Unit_Testing) ... ok
... test_build_accuracy_recall_graph (__main__.Unit_Testing) ... ok
... test_build_combination_dependence_graph (__main__.Unit_Testing) ... ok
... ----------------------------------------------------------------------
... Ran 14 tests in 38 seconds
...
... OK
```

Рисунок 3.5 – Результати проходження тестів

3.3 Вимоги до апаратних та програмних засобів

Отримана система класифікацій текстового контенту складається з багатьох модулів. Для того, щоб реалізований програмний продукт виконувався коректно, необхідно, щоб на персональному комп'ютері користувача були встановлені наступні програмні компоненти:

1) мова програмування : Python 3.9;
2) бібліотека matplotlib (pip install matplotlib);
3) бібліотека tkinter (pip install tkinter);
4) бібліотека scipy (pip install scipy);
5) бібліотека pandas (pip install pandas);
6) бібліотека numpy (pip install numpy);
7) бібліотека scikit-learn (pip install sklearn);
8) бібліотека pickle (pip install pickle);
9) бібліотека nltk (pip install nltk);
10) бібліотека shutil (pip install shutil);

Вимоги до апаратного забезпечення користувача наступні:

Мінімальні:

- ОС: Windows / 7
- процесор: x86, 800GH
- пам'ять (ОЗУ): 256 MB
- накопичувач: HDD
- дисплей: 1024x768

Рекомендовані:

- ОС: Windows 10
- процесор: x64, Intel Core i5- 7300 HQ
- пам'ять (ОЗУ): 4000 MB
- накопичувач: SSD
- дисплей: 1920x1080

3.4 Результати дослідження

В ході виконання поставленого завдання було з’ясовано, що застосування агрегативних підходів для класифікації на базі ансамблевих моделей «дає свої плоди». В якості оцінювальних метрик використовуються точність «accuracy», повнота «recall», та матриця невідповідностей «confusion matrics».

Повнота показує, яку частину об’єктів, які реально відносяться до істинного класу, модель передбачила правильно. На прикладі: повнота - це скільки з божевільних людей, яких ми перевірили, посадили в психлікарню.

«confusion matrics».являє собою матрицю розміром 2 на 2. На рисунку 3.6 зображена структура цієї матриці.

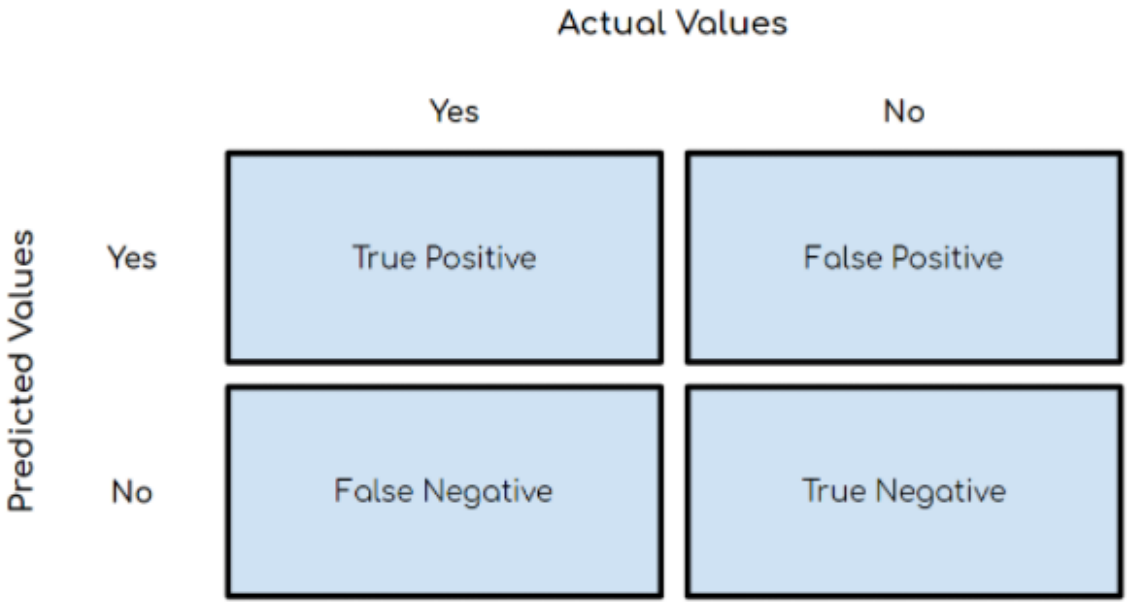


Рисунок 3.6 – Структура матриці невідповідностей

У таблиці, зображеній на рисунку 3.6 міститься інформація скільки разів система прийняла вірне і скільки разів невірне рішення за документами заданого класу. А саме:

– TP – істинно позитивне рішення;
– TN – істинно негативне рішення;
– FP – хибно позитивне рішення;
– FN – хибно негативне рішення.

На рисунку 3.7 зображений графік точності та повноти, побудований на основі оцінювання базових класифікаторів. В даному випадку – це «svm», «naive-bayes», «k-neighbors», «ada-boost», «random-forest», «logistic-regression». На рисунку 3.8 зображений графік точності та повноти, побудований на основі оцінювання комбінацій (наборів) моделей.

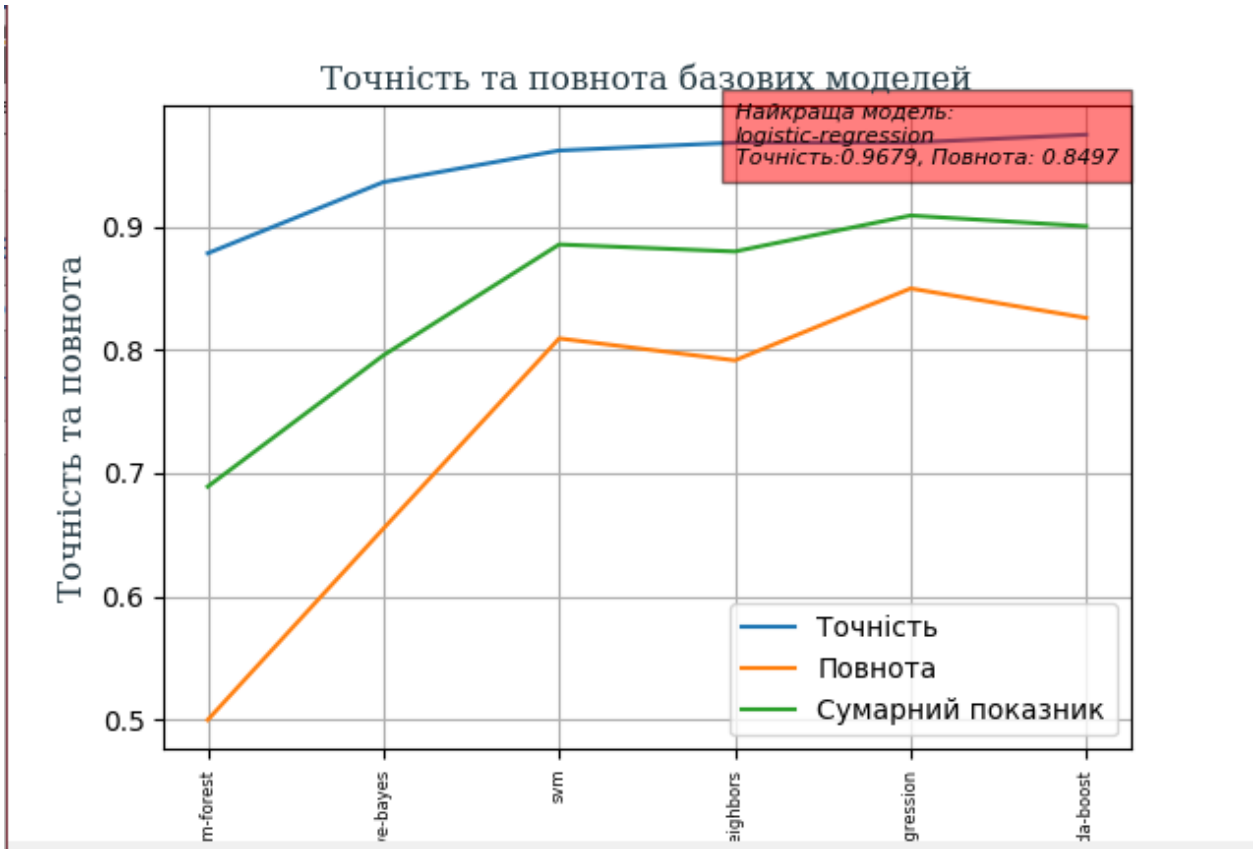


Рисунок 3.7 – Графік , побудований на основі оцінювання базових класифікаторів

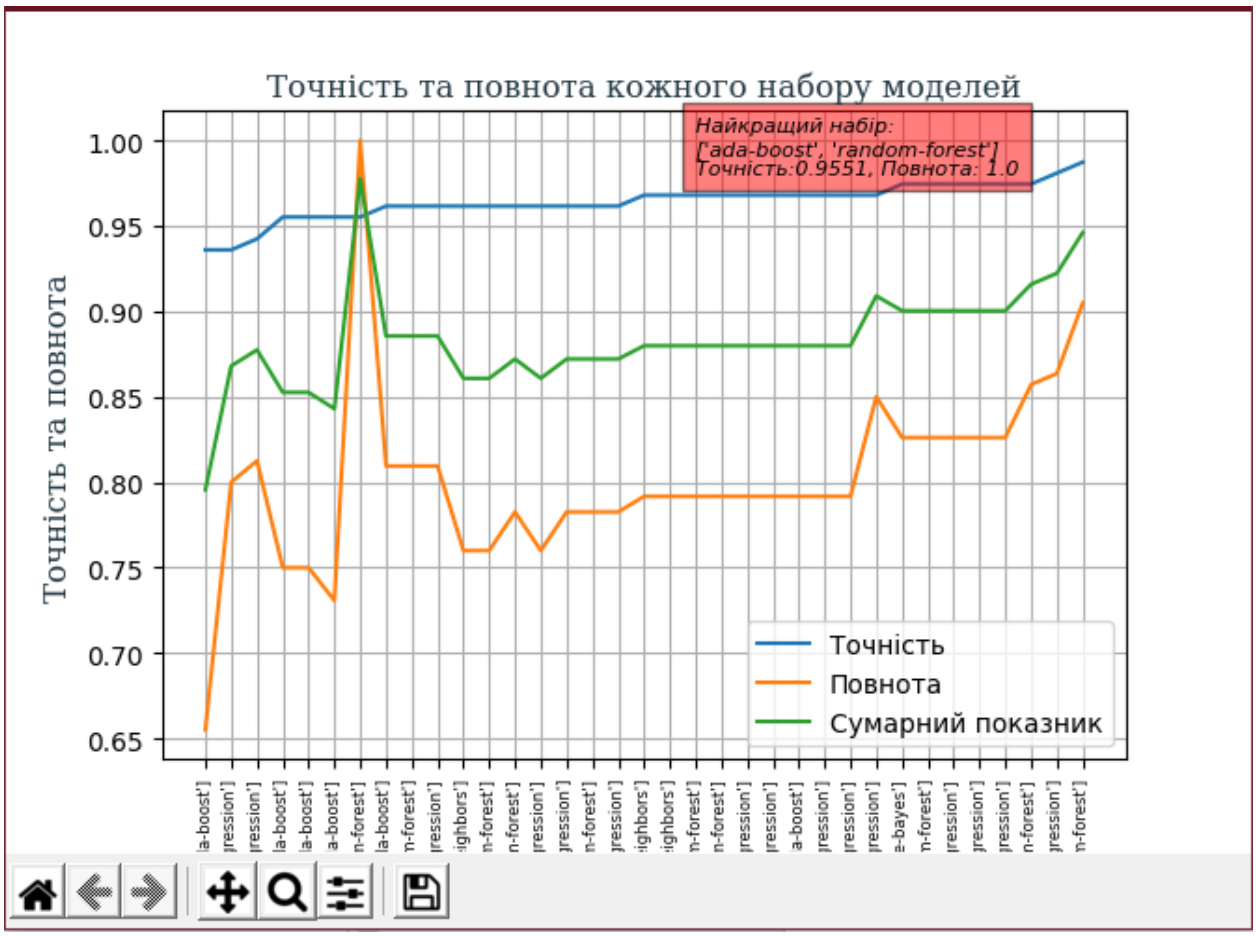


Рисунок 3.8 - Графік , побудований на основі оцінювання комбінацій (наборів) моделей

Як видно з графіків, зображених на рисунках 3.7-3.8, точність та повнота більша при застосуванні агрегативного підходу, аніж при застосуванні базових моделей. Цікаво примітити, що повнота комбінованих наборів класифікаторів значно перевищує повноту при звичайній класифікації.

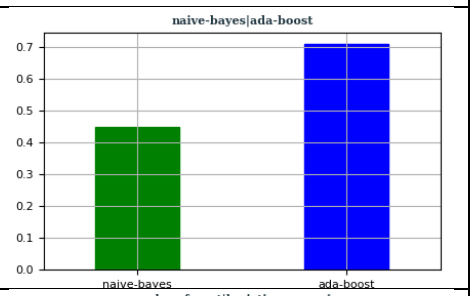
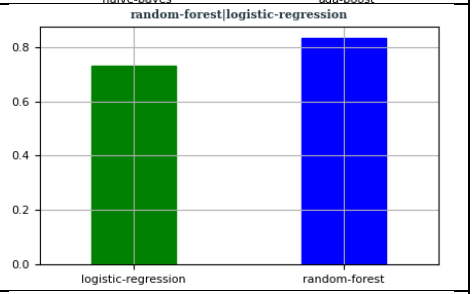
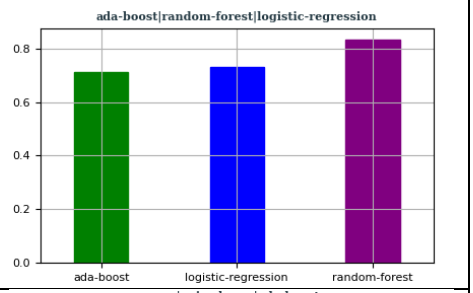
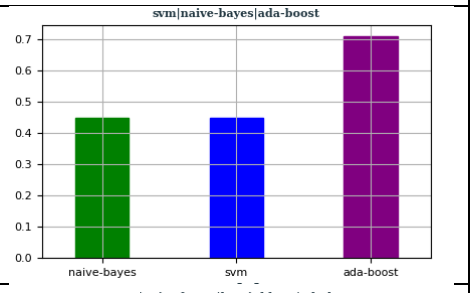
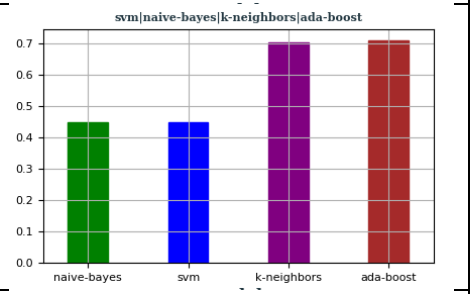
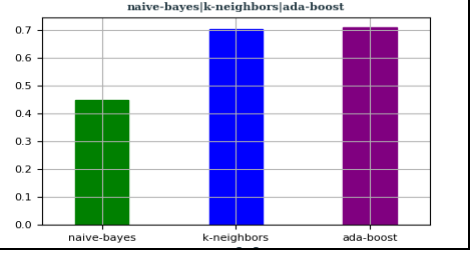
В таблиці 3.1 наведені результати базових класифікаторів, оцінених згідно базових метрик.

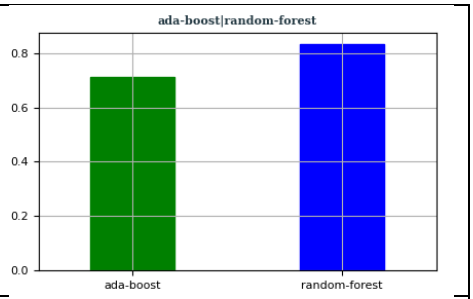
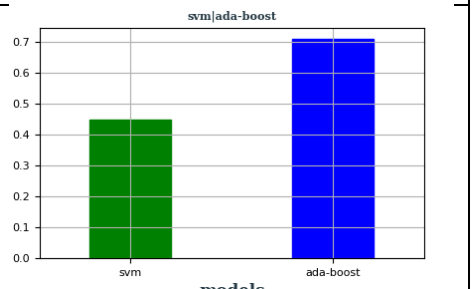
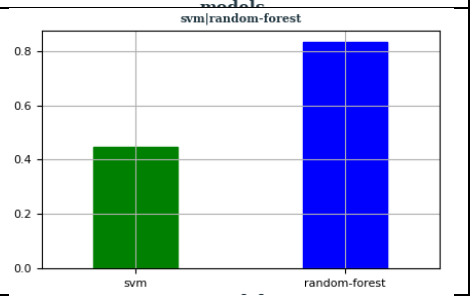
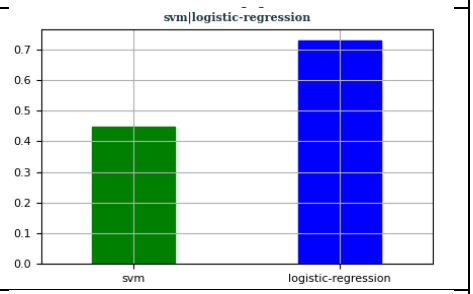
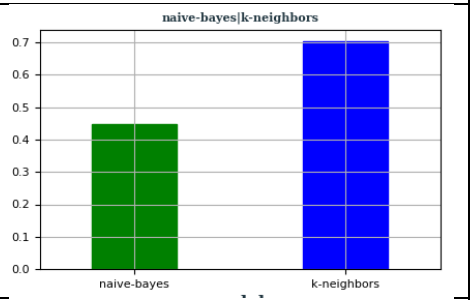
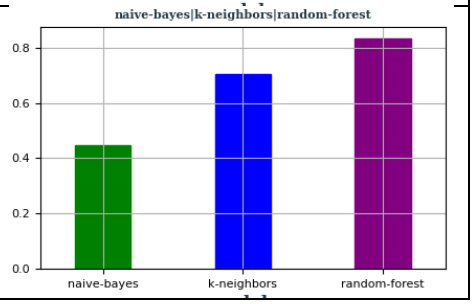
Таблиця 3.1 – результати базових класифікаторів

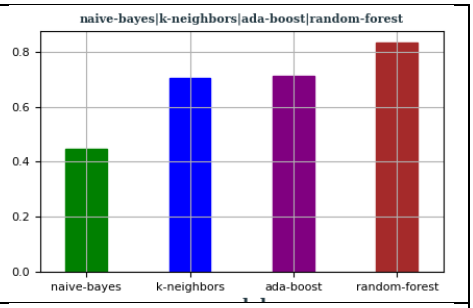
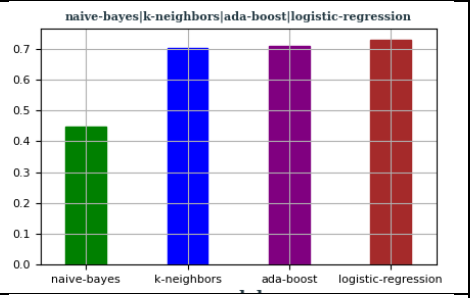
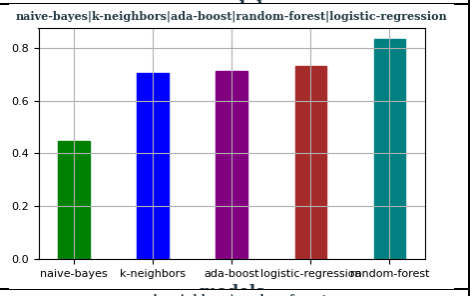
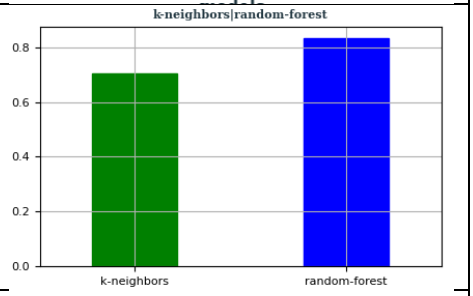
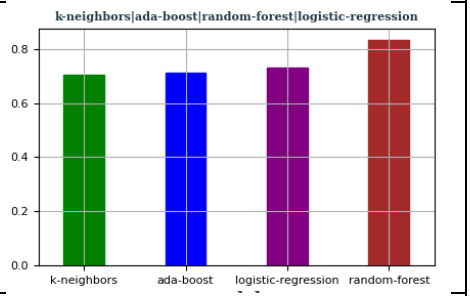
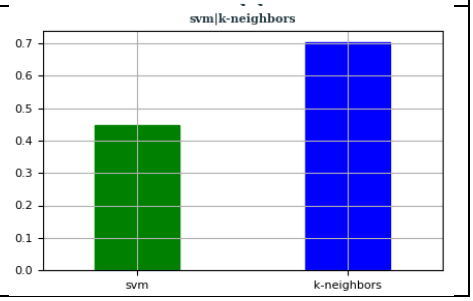
№	Назва класифікатора	Точність	Повнота	Матриця невідповідностей
1	«random-forest»	0.8782	0.5	TP – 0 FP - 19 FN – 0 TN - 137
2	«naive-bayes»	0.9359	0.6551	TP – 19 FP - 0 FN – 10 TN -127
3	«svm»	0.9615	0.7914	TP – 17 FP - 2 FN – 4 TN -133
4	«logistic-regression»	0.9679	0.8497	TP – 17 FP - 2 FN – 3 TN -124
5	«k-neighbors»	0.9679	0.7914	TP – 19 FP - 0 FN – 5 TN -132
6	«ada-boost»	0.9744	0.8258	TP – 19 FP - 0 FN – 4 TN -133

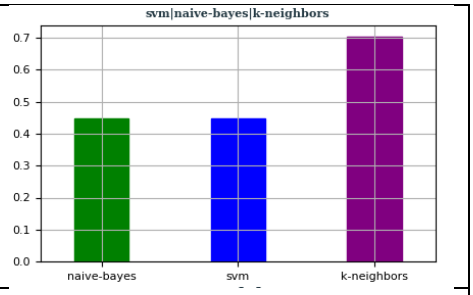
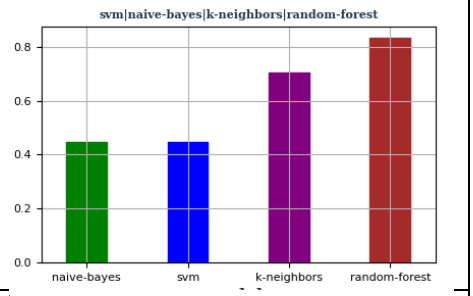
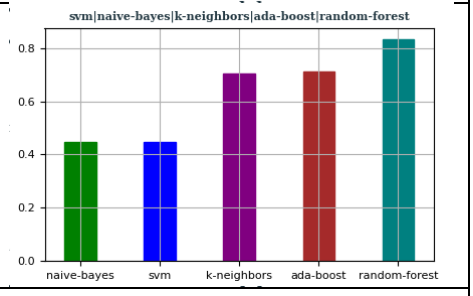
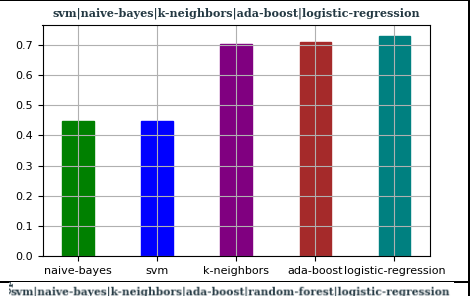
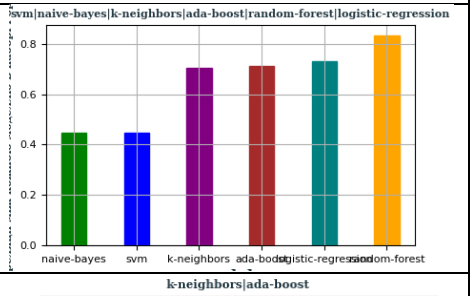
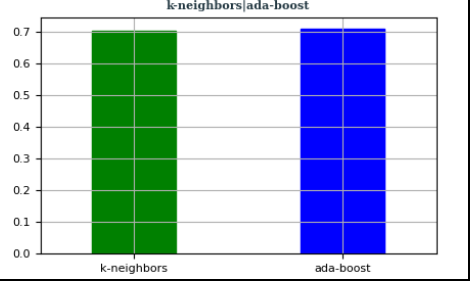
В таблиці 3.2 наведені результати наборів класифікаторів, оцінених згідно базових метрик. Слід зазначити, що в цій таблиці також наведені графіки коефіцієнтів кореляцій, які показують, наскільки істинні значення документів (оригінальні надписи – теми) пов’язані із передбаченням кожного класифікатора.

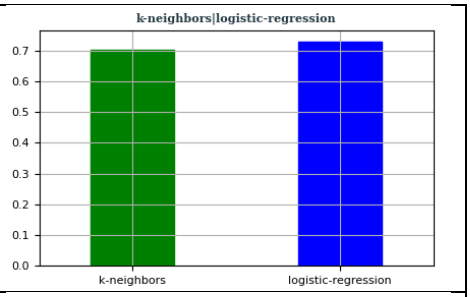
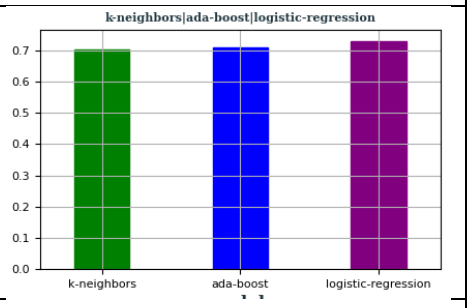
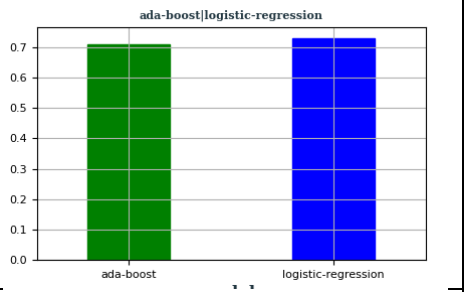
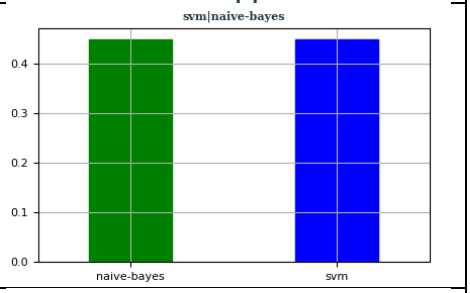
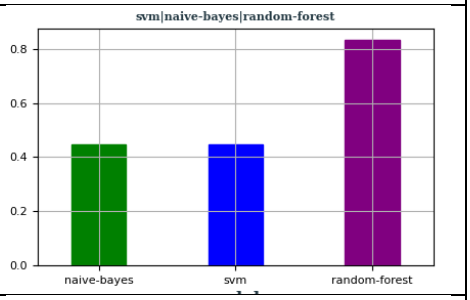
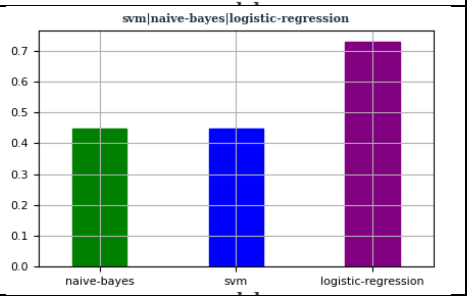
Таблиця 3.2 - результати комбінацій моделей

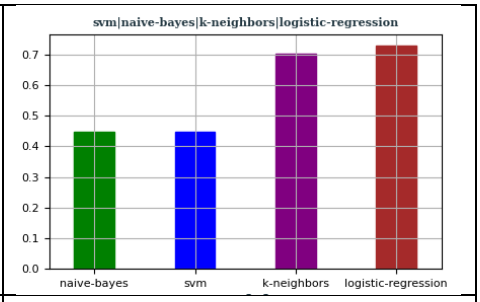
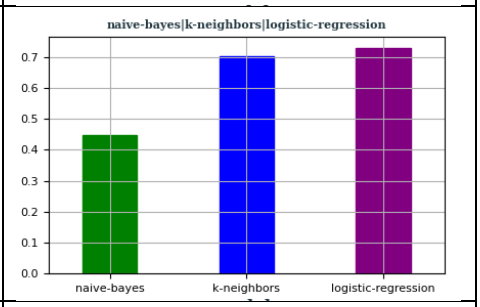
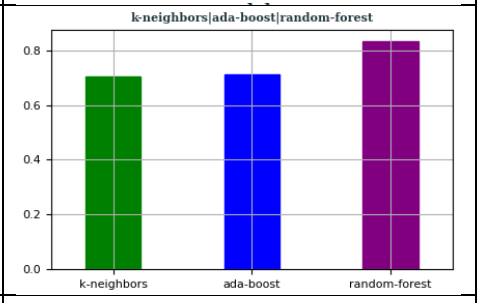
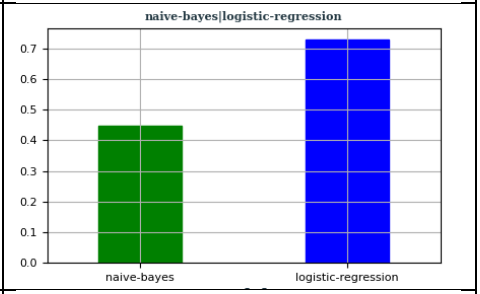
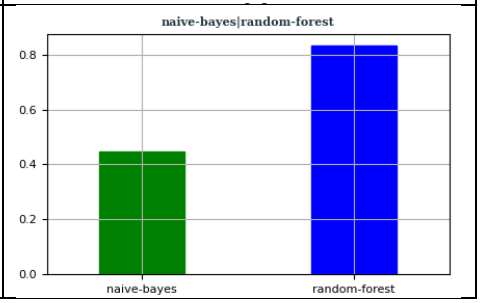
№	Назва моделей в комбінації	Точність	Повнота	Матриця невідповідностей	
1	'naive-bayes|ada-boost'	0.9359	0.6552	TP – 19 FP - 0 FN – 10 TN - 127	
2	'random-forest|logistic-regression'	0.9359	0.8	TP – 12 FP - 7 FN – 3 TN - 134	
3	'ada-boost|random-forest|logistic-regression'	0.9423	0.8125	TP – 13 FP - 6 FN – 3 TN – 134	
4	'svm|naive-bayes|ada-boost'	0.9551	0.75	TP – 18 FP - 1 FN – 6 TN - 131	
5	'svm|naive-bayes|k-neighbors|ada-boost'	0.9551	0.75	TP – 18 FP - 1 FN – 6 TN - 131	
6	'naive-bayes|k-neighbors|ada-boost'	0.9551	0.73	TP – 19 FP - 0 FN – 7 TN - 130	

7	'ada-boost|random-forest'	0.9551	1	TP – 12 FP - 7 FN – 0 TN – 137	
8	'svm|ada-boost'	0.9615	0.8095	TP – 17 FP - 2 FN – 4 TN - 133	
9	'svm|random-forest'	0.9615	0.8095	TP – 17 FP - 2 FN – 4 TN - 133	
10	'svm|logistic-regression'	0.9615	0.8095	TP – 17 FP - 2 FN – 4 TN -133	
11	'naive-bayes|k-neighbors'	0.9615	0.76	TP –19 FP - 0 FN – 6 TN -131	
12	'naive-bayes|k-neighbors|random-forest'	0.9615	0.76	TP – 19 FP - 0 FN – 6 TN -131	

13	'naive-bayes|k-neighbors|ada-boost|random-forest'	0.9615	0.7826	TP – 18 FP - 1 FN – 5 TN -132	
14	'naive-bayes|k-neighbors|ada-boost|logistic-regression'	0.9615	0.76	TP –19 FP - 0 FN – 6 TN -131	
15	'naive-bayes|k-neighbors|ada-boost|random-forest|logistic-regression'	0.9615	0.7826	TP – 18 FP - 1 FN – 5 TN -132	
16	'k-neighbors|random-forest'	0.9615	0.7826	TP – 18 FP - 1 FN – 5 TN -132	
17	'k-neighbors|ada-boost|random-forest|logistic-regression'	0.9615	0.7826	TP – 18 FP - 1 FN – 5 TN -132	
18	'svm|k-neighbors'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	

19	'svm|naive-bayes|k-neighbors'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
20	'svm|naive-bayes|k-neighbors|random-forest'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
21	'svm|naive-bayes|k-neighbors|ada-boost|random-forest'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
22	'svm|naive-bayes|k-neighbors|ada-boost|logistic-regression'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
23	'svm|naive-bayes|k-neighbors|ada-boost|random-forest|logistic-regression'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
24	'k-neighbors|ada-boost'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	

25	'k-neighbors|logistic-regression'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
26	'k-neighbors|ada-boost|logistic-regression'	0.9679	0.7917	TP – 19 FP - 0 FN – 5 TN -132	
27	'ada-boost|logistic-regression'	0.9679	0.85	TP – 17 FP - 2 FN – 3 TN -134	
28	'svm|naive-bayes'	0.9744	0.8261	TP – 19 FP - 0 FN – 4 TN -133	
29	'svm|naive-bayes|random-forest'	0.9744	0.8261	TP – 19 FP - 0 FN – 4 TN -133	
30	'svm|naive-bayes|logistic-regression'	0.9744	0.8261	TP – 19 FP - 0 FN – 4 TN -133	

31	'svm|naive-bayes|k-neighbors|logistic-regression'	0.9744	0.8261	TP – 19 FP - 0 FN – 4 TN -133	
32	'naive-bayes|k-neighbors|logistic-regression'	0.9744	0.9744	TP – 19 FP - 0 FN – 4 TN -133	
33	'k-neighbors|ada-boost|random-forest'	0.9744	0.8571	TP – 18 FP - 1 FN – 3 TN -134	
34	'naive-bayes|logistic-regression'	0.9808	0.8606	TP – 19 FP - 0 FN – 3 TN -134	
35	'naive-bayes|random-forest'	0.9872	0.9048	TP – 19 FP - 0 FN – 2 TN -135	

На рисунках 3.9-3.10 зображені графіки гістограм в трьохвимірному просторі, які показують коефіцієнти кореляцій в статиці між передбаченням кожної моделі в наборі і оригінальним значенням документа (тема). Тобто, ці коефіцієнти кореляцій визначались відносно сталого набору документів.

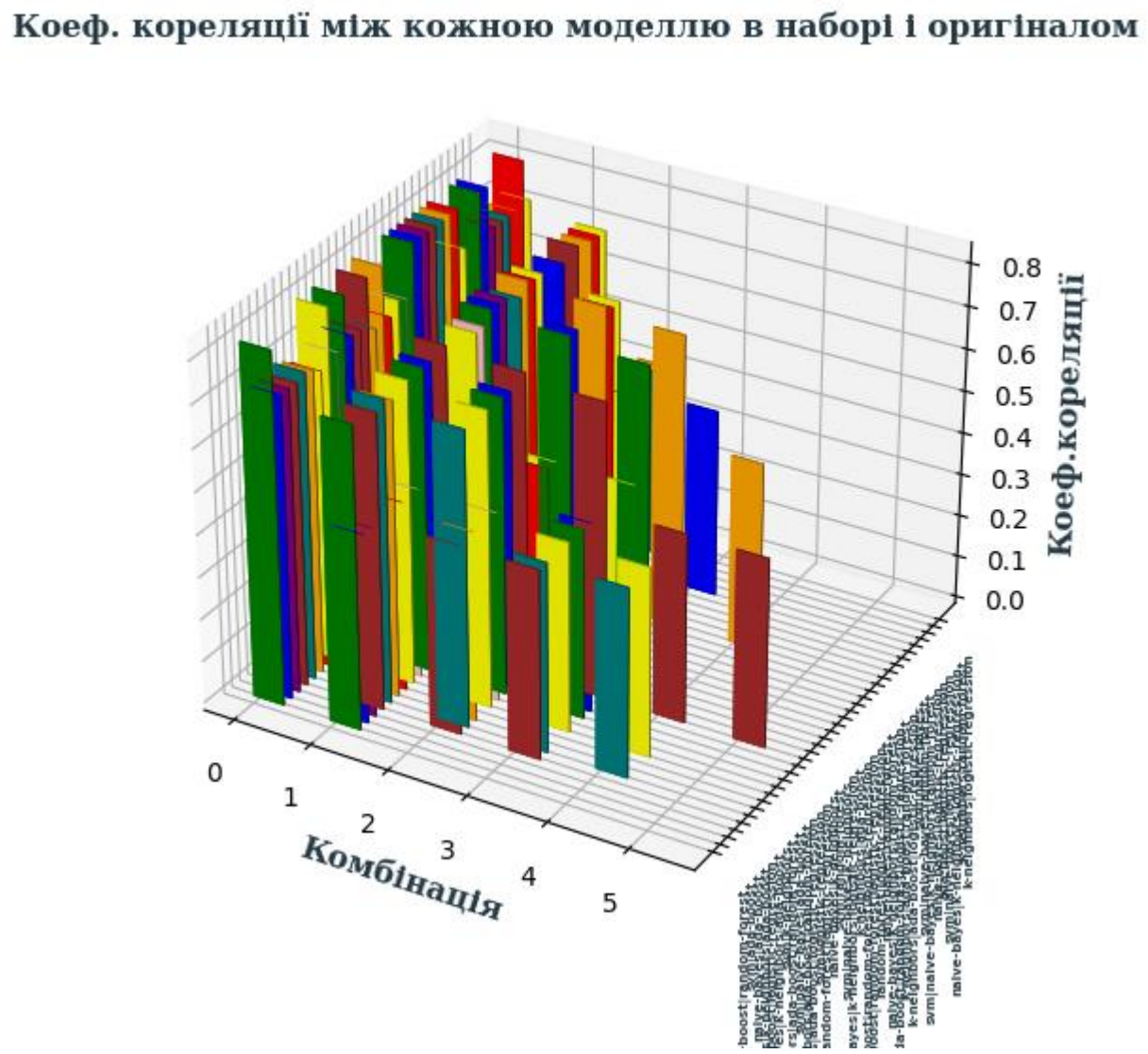


Рисунок 3.9 – Графіки гістограм в трьох вимірному просторі для кореляцій в статиці

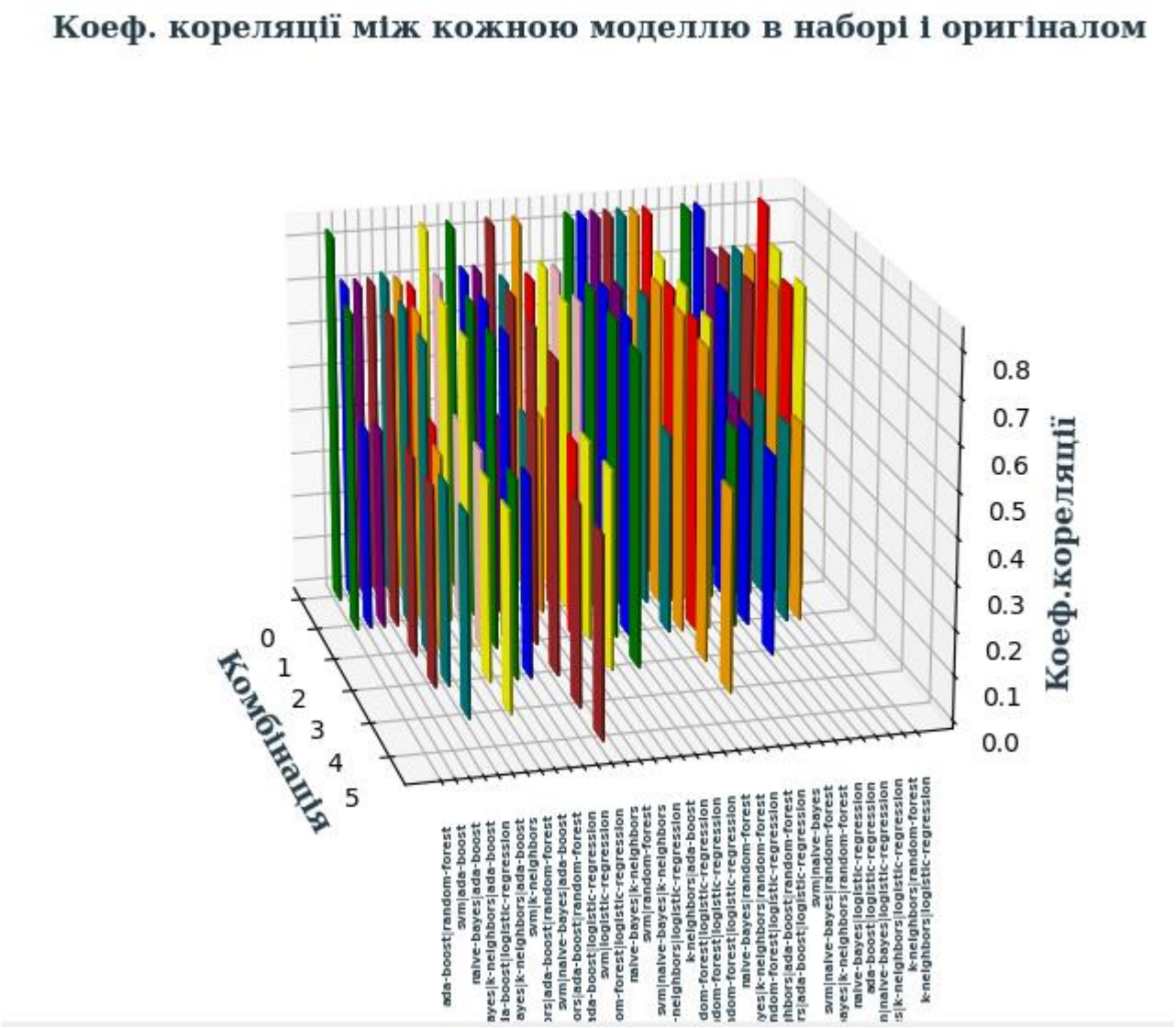


Рисунок 3.10 – Графіки гістограм в трьох вимірному просторі для кореляцій в статиці

На рисунку 3.11 зображені графіки гістограм в трьохвимірному просторі, які показують коефіцієнти кореляцій в динаміці між передбаченням кожної моделі в наборі і оригінальним значенням документа (тема). Тобто, ці коефіцієнти кореляцій визначались відносно динамічного набору документів.

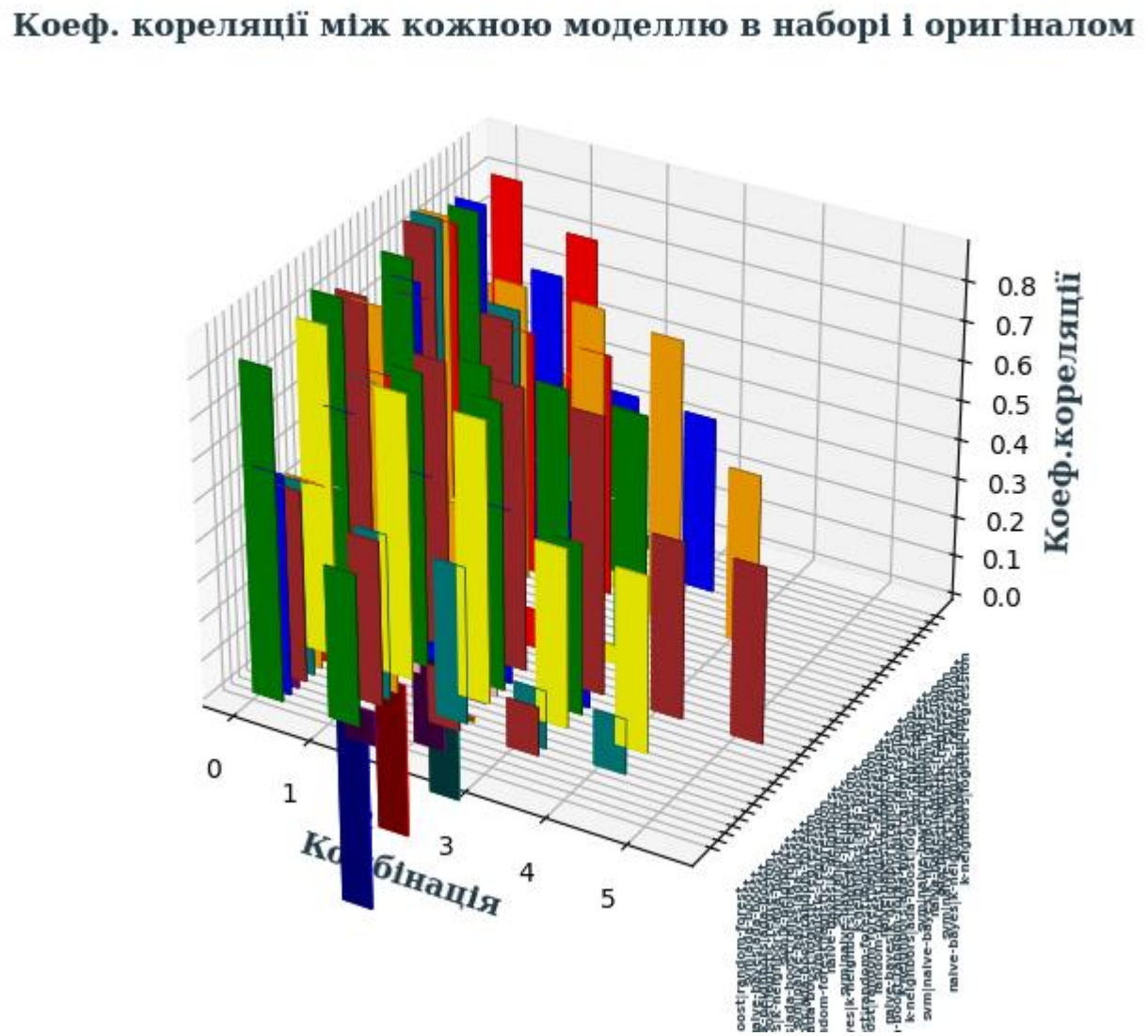


Рисунок 3.11 – Графіки гістограм в трьох вимірному просторі для кореляцій в динаміці

Висновки

Під час розроблення кваліфікаційної роботи було проведено аналіз предметної області машинного навчання. В ході виконання поставленого технічного завдання було проведено дослідження застосування агрегативних підходів для класифікації на базі ансамблевих моделей.

Протягом періоду розробки було зроблено аналіз сучасних ансамблевих підходів, реалізовано програмну систему з використанням інформаційної технології класифікації на базі кластерної оптимізації.

Результатом кваліфікаційної роботи є застосунок, призначений для класифікації на заданому корпусі даних. Реалізований програмний продукт використовує базові алгоритми класифікації, та агрегує їх. За допомогою агрегатування досягається вища кваліфікація системи класифікації, або, іншими словами, вища точність передбачень ансамблевої моделі.

Кінцевим результатом реалізованого програмного продукту є:

— графіки точностей моделей;
— графік залежності кількості комбінацій від кількості базових класифікаторів;
— графіки коефіцієнтів кореляцій між передбаченнями моделей та оригіналом;
— графіки коефіцієнтів кореляцій в трьох вимірному просторі.

Відповідно до отриманого результату, можна зробити висновок, що розроблений додаток працює вірно, а вимоги технічного завдання виконанні в повному обсязі.

Одним із шляхів вдосконалення реалізованої системи класифікації є підбір базових класифікаторів із множини на основі вирахуваних коефіцієнтів кореляцій між передбаченнями класифікаторів і правильними відповідями. Такий підхід допоможе в подальшому ще більше оптимізувати ансамблеву модель, оскільки ансамбль буде складатись із тих моделей, які показують найвищі метрики точності.

Перелік посилань

1. Машинне навчання. [Електронний ресурс]. – Режим доступу: https://futurenow.com.ua/shho-take-mashynne-navchannya-prosto-pro-skladne-za-5-hvylyn/

2. Wikipedia: Класифікація даних. [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Задача_класифікації

3. Кластеризація даних. [Електронний ресурс]. – Режим доступу: https://dl.acm.org/doi/10.1145/331499.331504

4. Самокерований автомобіль Google. [Електронний ресурс]. – Режим доступу: https://waymo.com/

5. YouTube. [Електронний ресурс]. – Режим доступу: https://www.youtube.com/?gl=UA&tab=w1

6. Netflix. [Електронний ресурс]. – Режим доступу: https://www.netflix.com/ua-ru/

7. Amazon. [Електронний ресурс]. – Режим доступу: https://www.amazon.com/

8. Twitter. [Електронний ресурс]. – Режим доступу: https://twitter.com/?lang=uk

9. Wikipedia: Контрольоване навчання (Supervised learning). [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Навчання_з_учителем

10. Завдання класифікації. [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/classical-machine-learning.html

11. Завдання регресії. [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/classical-machine-learning.html

12. Wikipedia: Неконтрольоване навчання (Unsupervised learning). [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Навчання_без_учителя

13. Кластерний аналіз, завдання кластеризації. [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/getting-started-machine-learning/

14. Ансамблеве моделювання. . [Електронний ресурс]. – Режим доступу:https://www.sciencedirect.com/topics/computer-science/ensemble-modeling

15. Комітети (голосування, Voting Ensembles). [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789136609/2/ch02lvl1sec16/max-voting

16. Wikipedia: Бегінг (Bagging). [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Бэггинг

17. Випадковий ліс (Random Forest). [Електронний ресурс]. – Режим доступу:https://dyakonov.org/2016/11/14/%d1%81%d0%bb%d1%83%d1%87%d0%b0%d0%b9%d0%bd%d1%8b%d0%b9-%d0%bb%d0%b5%d1%81-random-forest/

18. Бустінг (Boosting). [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/ensembles.html

19. Стекінг (Stacking). [Електронний ресурс]. – Режим доступу: https://dyakonov.org/2017/03/10/c%d1%82%d0%b5%d0%ba%d0%b8%d0%bd%d0%b3-stacking-%d0%b8-%d0%b1%d0%bb%d0%b5%d0%bd%d0%b4%d0%b8%d0%bd%d0%b3-blending/

20. Uclassify. [Електронний ресурс]. – Режим доступу: https://uclassify.com/

21. Wikipedia: Наївний баєсів класифікатор. [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

22. Теорема Байєса і не тільки. [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/naive-bayes-classifiers/

23. ARM NN. [Електронний ресурс]. – Режим доступу: https://developer.arm.com/ip-products/processors/machine-learning/arm-nn

24. Python Scikit-Learn. [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/

25. Accord.Net. [Електронний ресурс]. – Режим доступу: http://accord-framework.net/

26. Wikipedia: NLP. [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/NLP

27. Wikipedia: Ієрархічні методи кластерного аналізу. [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Иерархическая_кластеризация

28. Алгоритм k-means. [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/427761/

29. DBSCAN. [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

30. Векторизація тексту. [Електронний ресурс]. – Режим доступу: https://monkeylearn.com/blog/beginners-guide-text-vectorization/

31. Python. [Електронний ресурс]. – Режим доступу: https://www.python.org/

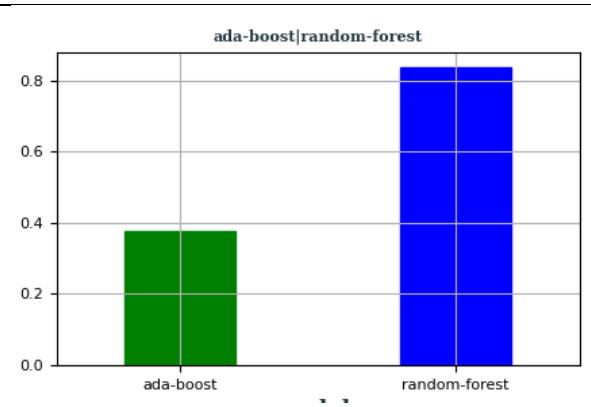
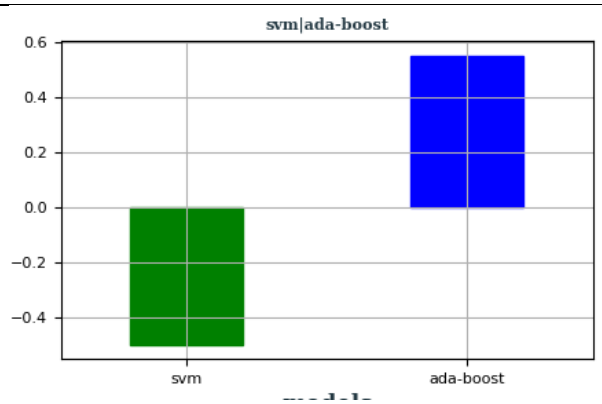
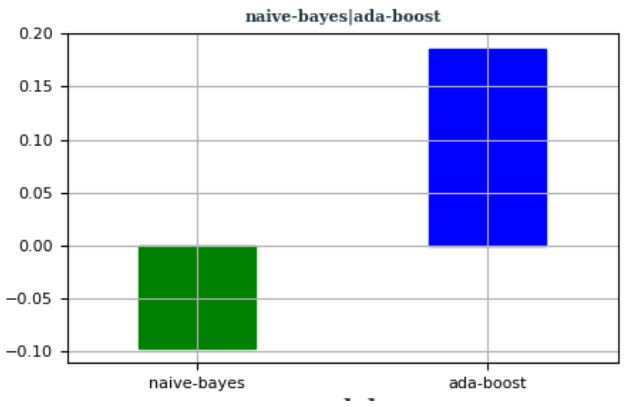
32. Matplotlib. [Електронний ресурс]. – Режим доступу: https://matplotlib.org/

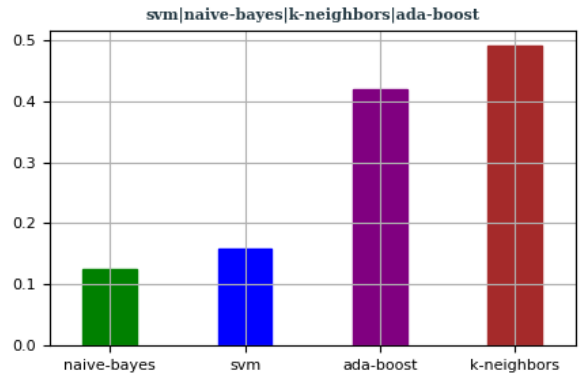
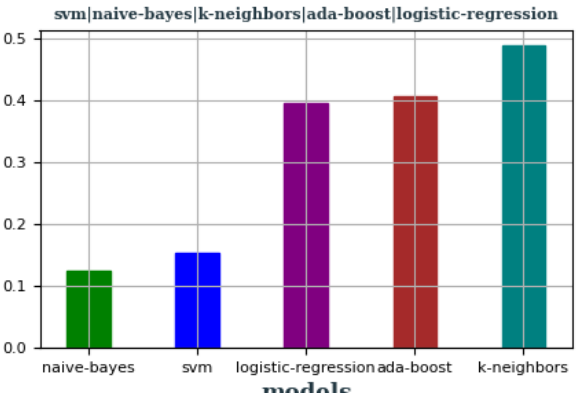
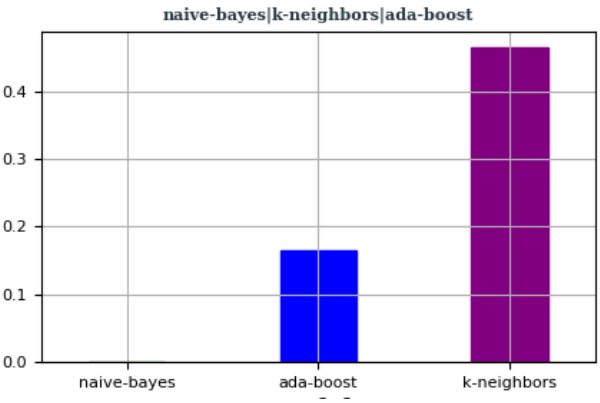
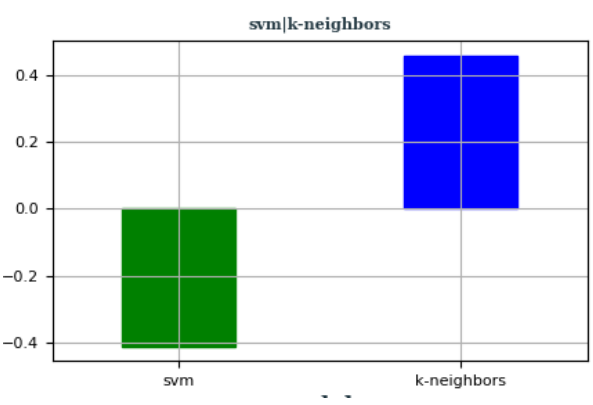
33. Tkinter. [Електронний ресурс]. – Режим доступу: https://pythonru.com/uroki/obuchenie-python-gui-uroki-po-tkinter.

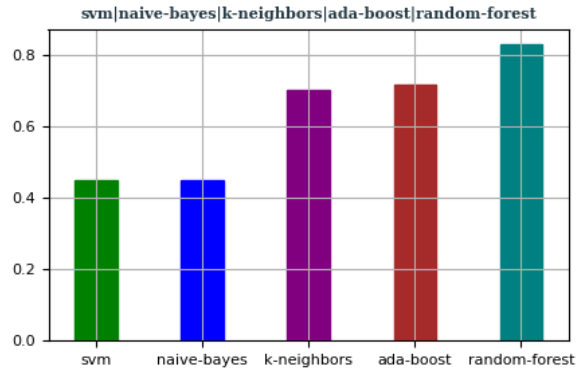
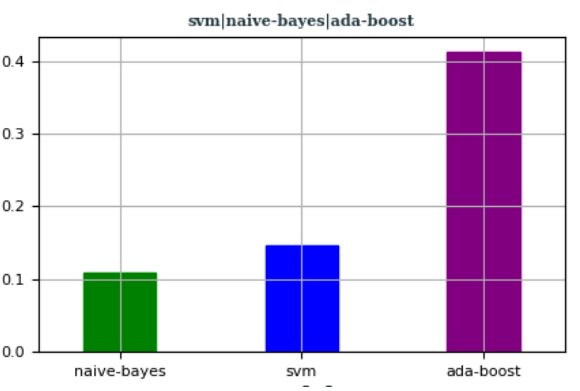
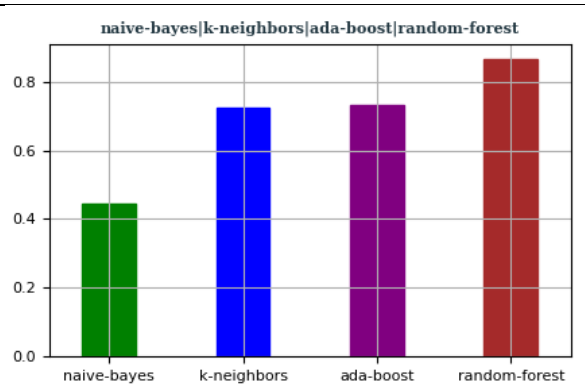
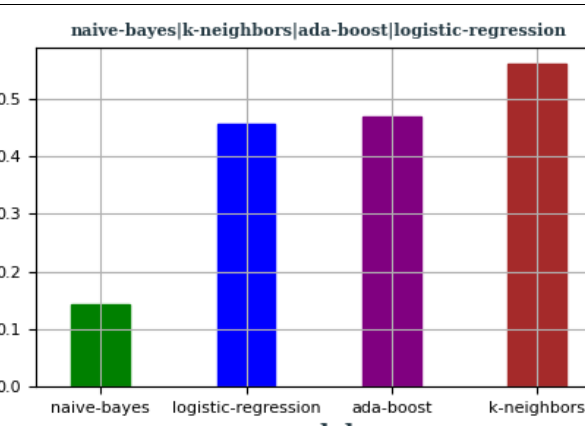
ДОДАТКИ

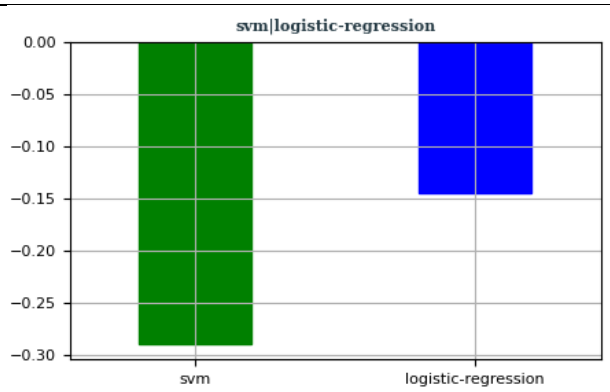
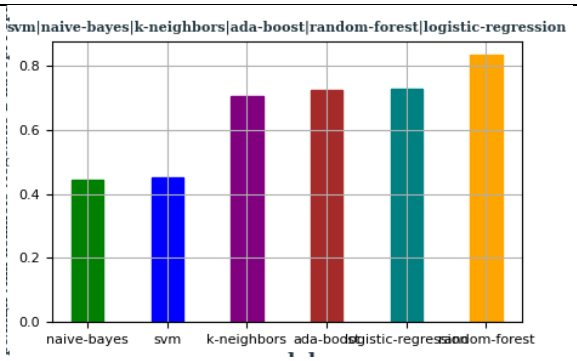
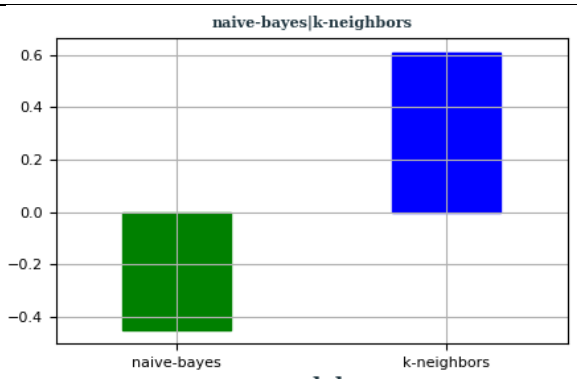
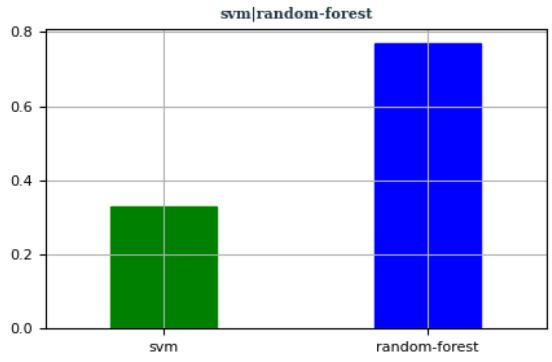
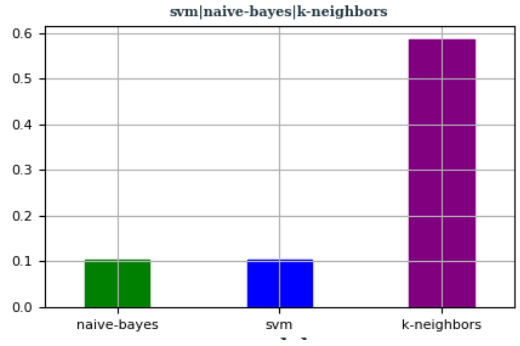
Додаток А

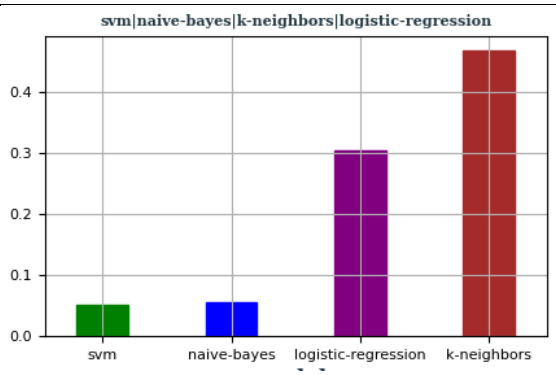
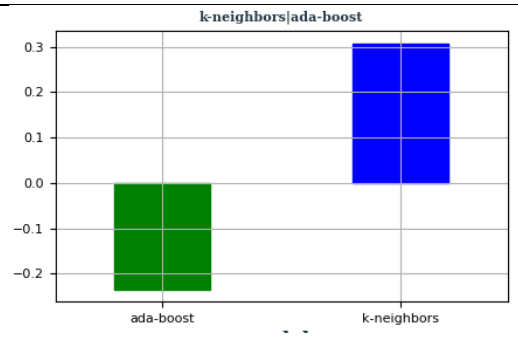
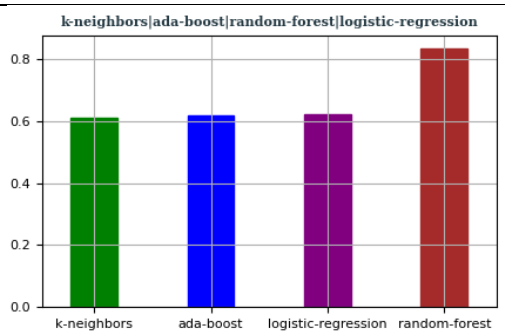
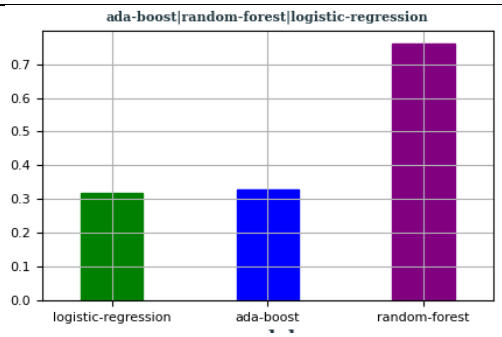
Динамічні коефіцієнти кореляцій для кожної моделі в кожному універсальному наборі

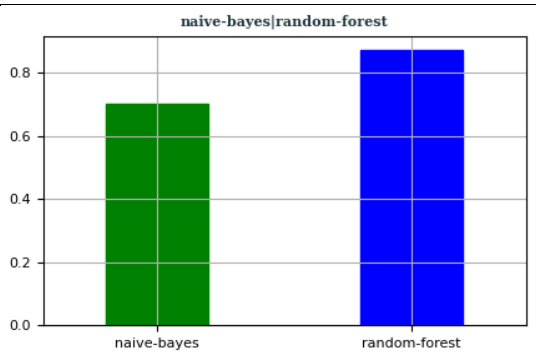
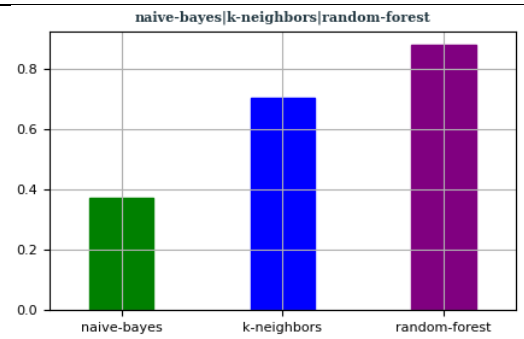
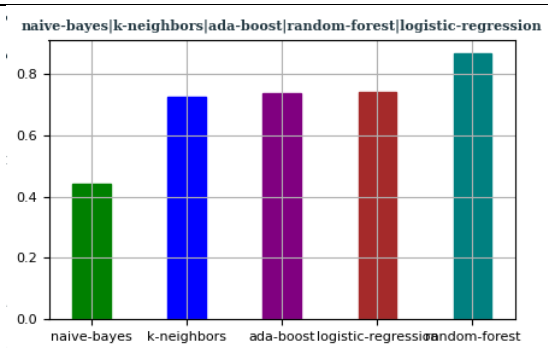
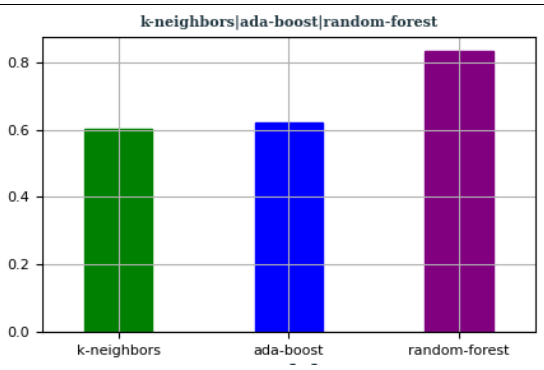
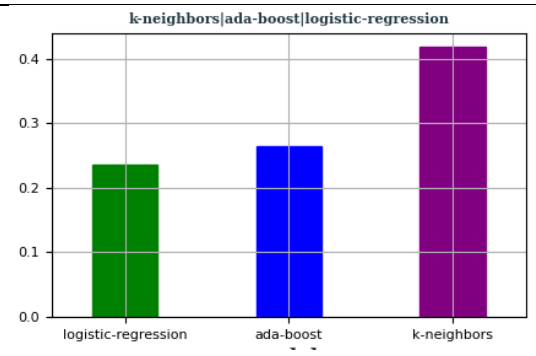
№	Назва комбінації‘	Коефіцієнт кореляції
1	'ada-boost|random-forest'	
2	'svm|ada-boost'	
3	'naive-bayes|ada-boost'	

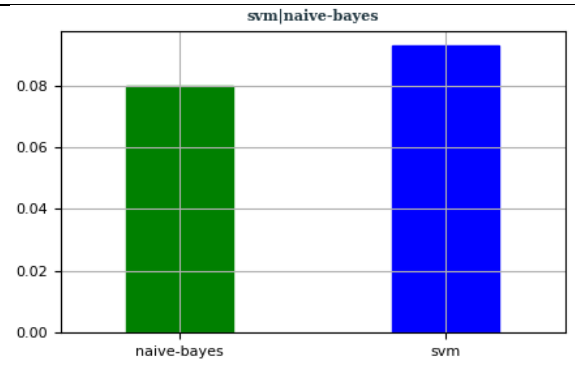
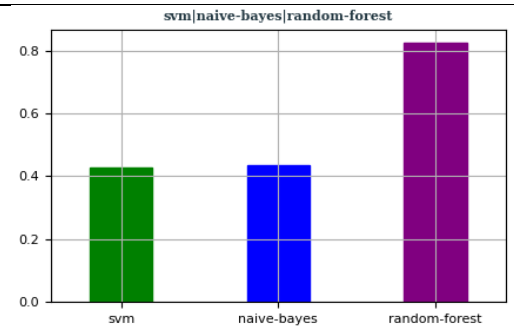
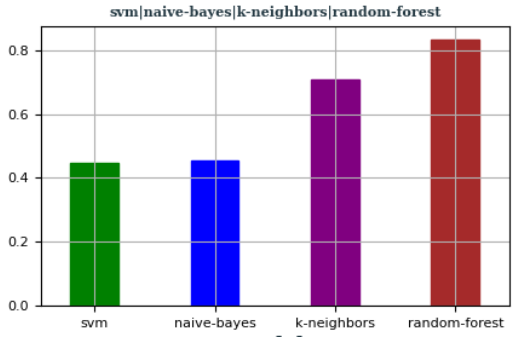
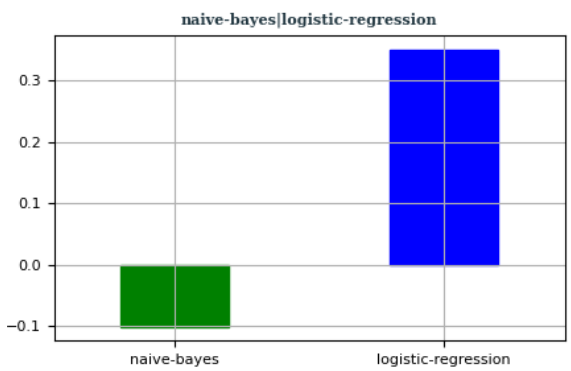
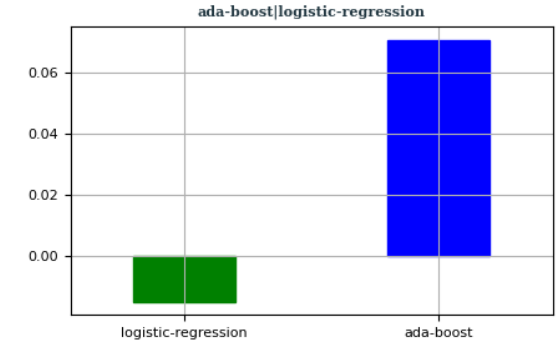
4	'svm|naive-bayes|k-neighbors|ada-boost'	
5	'svm|naive-bayes|k-neighbors|ada-boost|logistic-regression'	
6	'naive-bayes|k-neighbors|ada-boost'	
7	'svm|k-neighbors'	

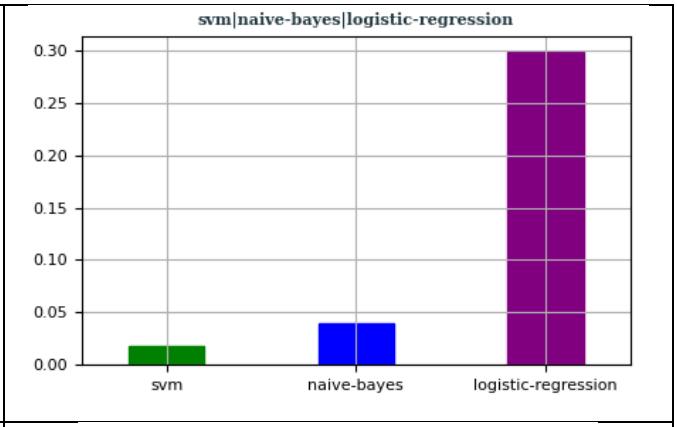
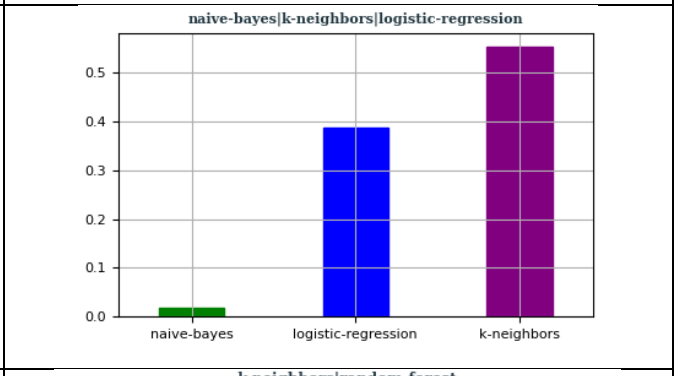
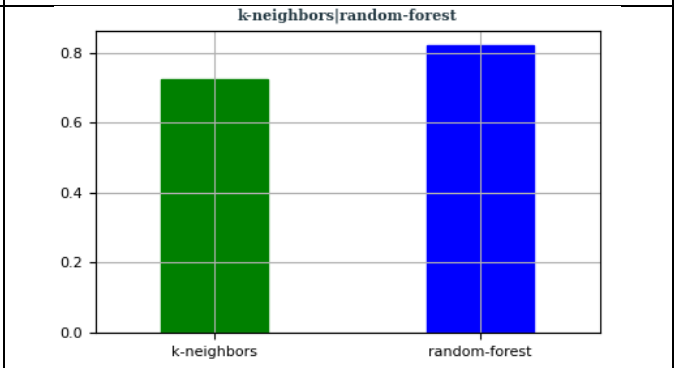
8	'svm|naive-bayes|k-neighbors|ada-boost|random-forest'	
9	'svm|naive-bayes|ada-boost'	
10	'naive-bayes|k-neighbors|ada-boost|random-forest'	
11	'naive-bayes|k-neighbors|ada-boost|logistic-regression'	

12	'svm|logistic-regression'	
13	'svm|naive-bayes|k-neighbors|ada-boost|random-forest|logistic-regression'	
14	'naive-bayes|k-neighbors'	
15	'svm|random-forest'	
16	'svm|naive-bayes|k-neighbors'	

17	'svm|naive-bayes|k-neighbors|logistic-regression'	
18	'k-neighbors|ada-boost'	
19	'k-neighbors|ada-boost|random-forest|logistic-regression'	
20	'ada-boost|random-forest|logistic-regression'	
21	'random-forest|logistic-regression'	

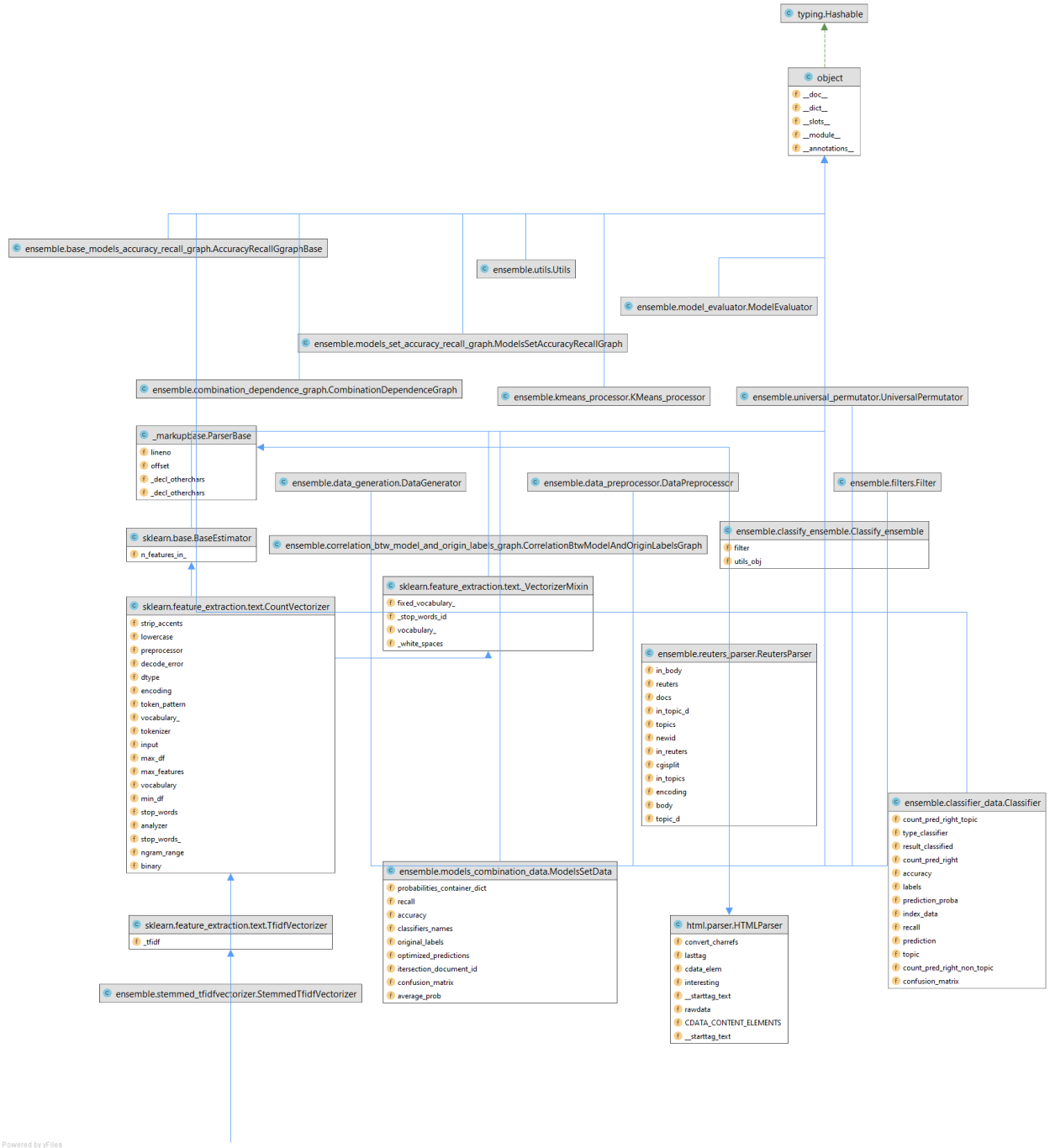
22	'naive-bayes|random-forest'	
23	'naive-bayes|k-neighbors|random-forest'	
24	'naive-bayes|k-neighbors|ada-boost|random-forest|logistic-regression'	
25	'k-neighbors|ada-boost|random-forest'	
26	'k-neighbors|ada-boost|logistic-regression'	

27	'svm|naive-bayes'	
28	'svm|naive-bayes|random-forest'	
29	'svm|naive-bayes|k-neighbors|random-forest'	
30	'naive-bayes|logistic-regression'	
31	'ada-boost|logistic-regression'	

32	'svm|naive-bayes|logistic-regression'	
33	'naive-bayes|k-neighbors|logistic-regression'	
34	'k-neighbors|random-forest'	
35	'k-neighbors|logistic-regression'	

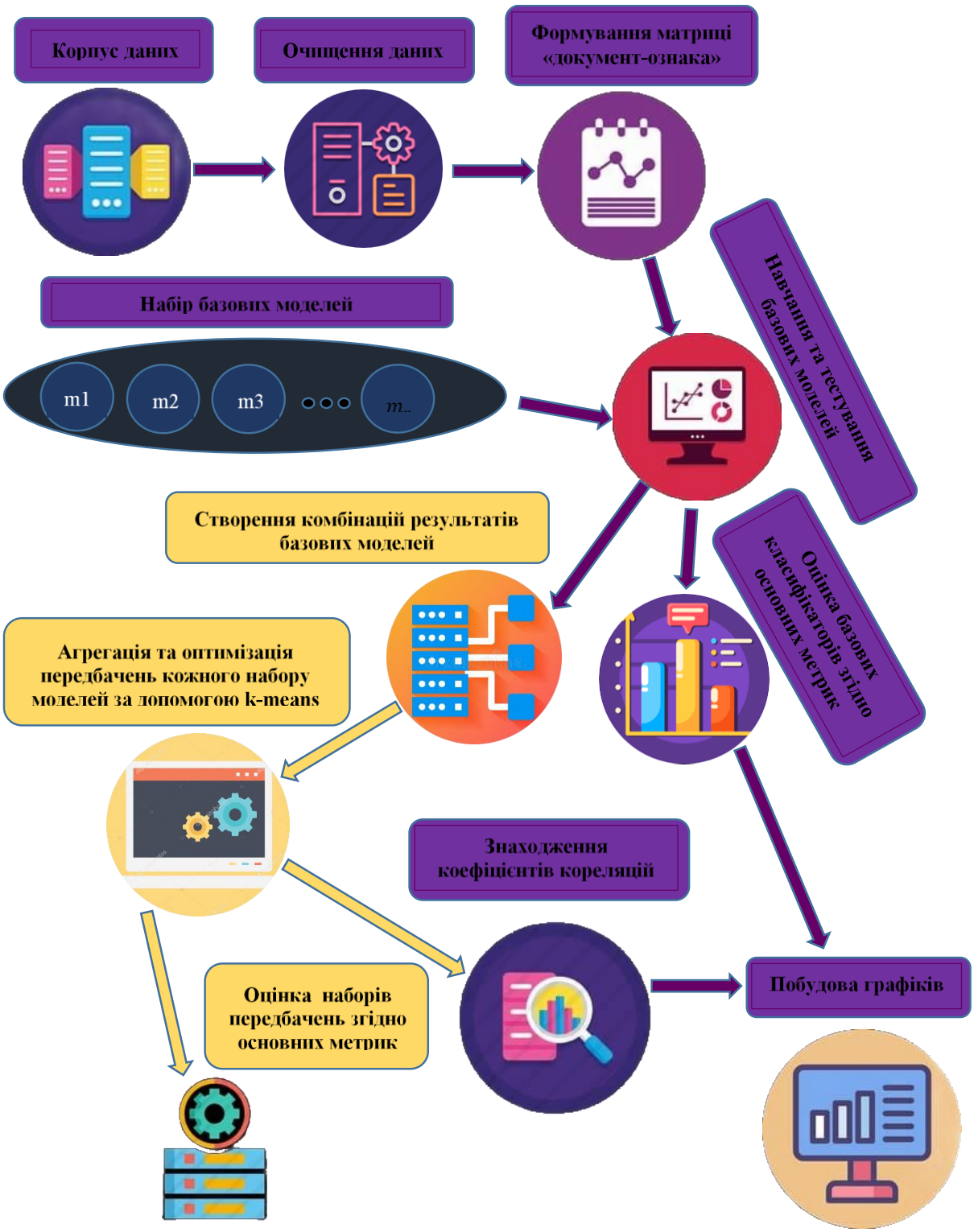
Додаток Б

Діаграма класів інформаційної системи класифікації даних



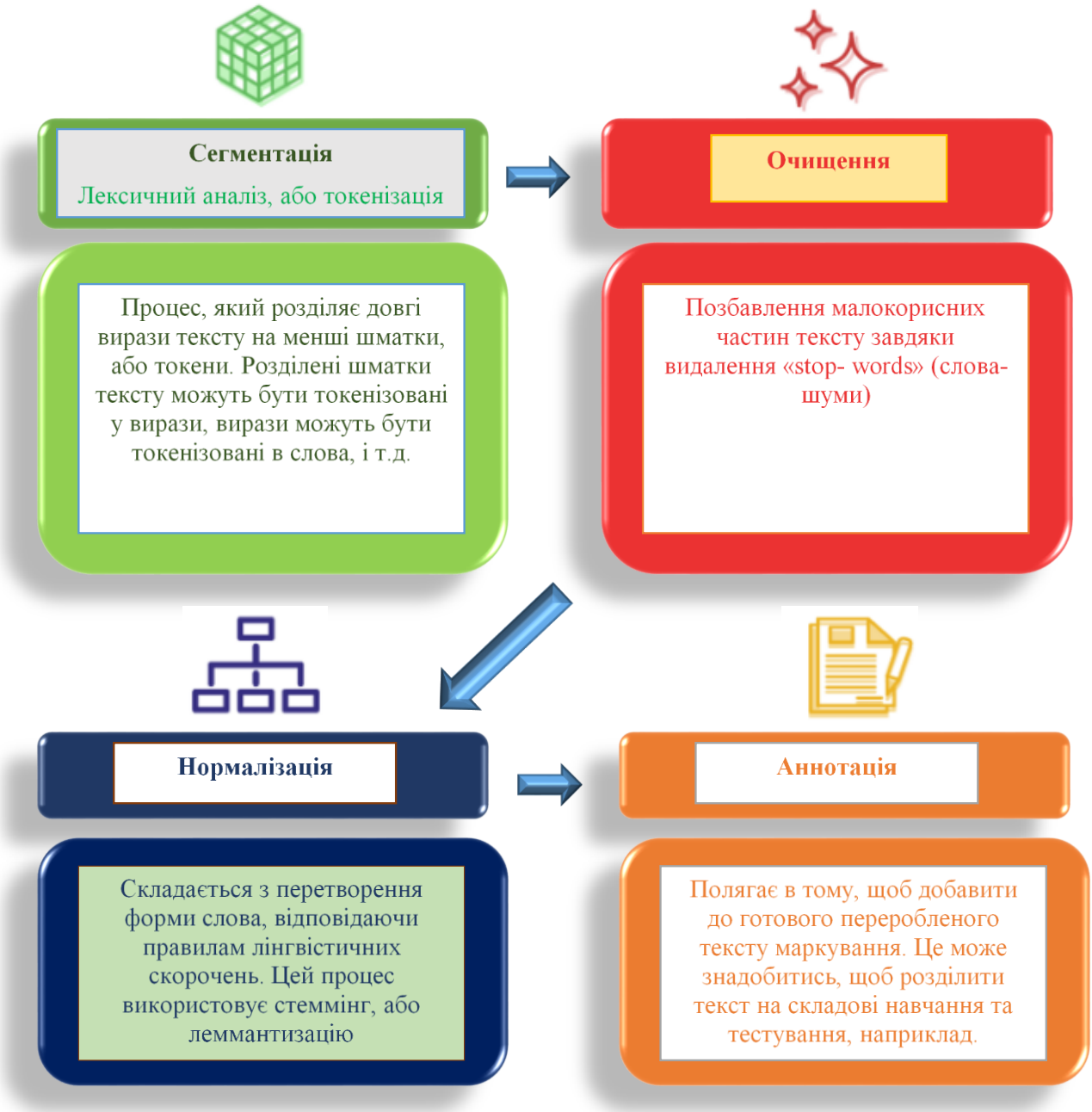
Додаток В

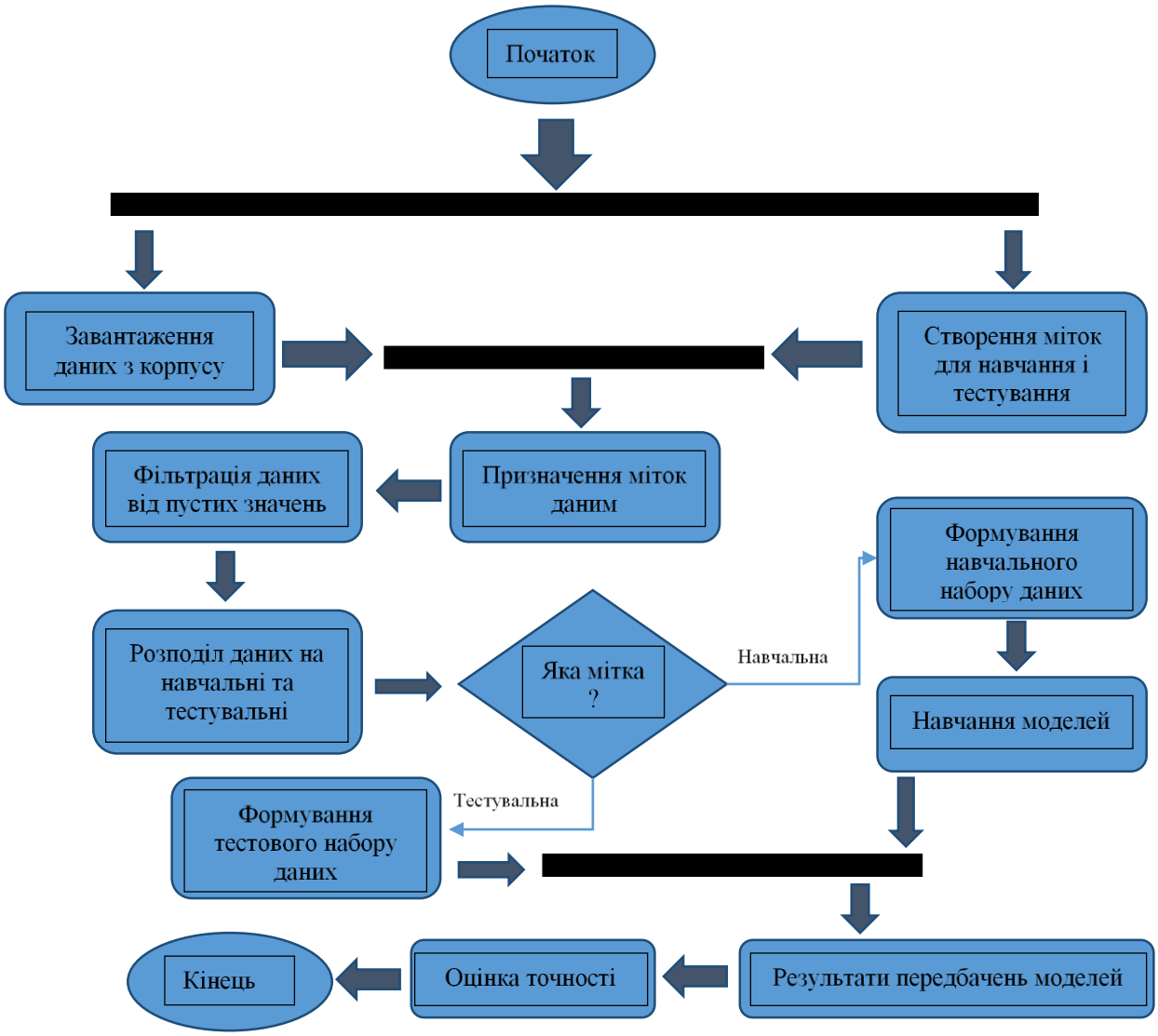
Узагальнена діаграма проекту

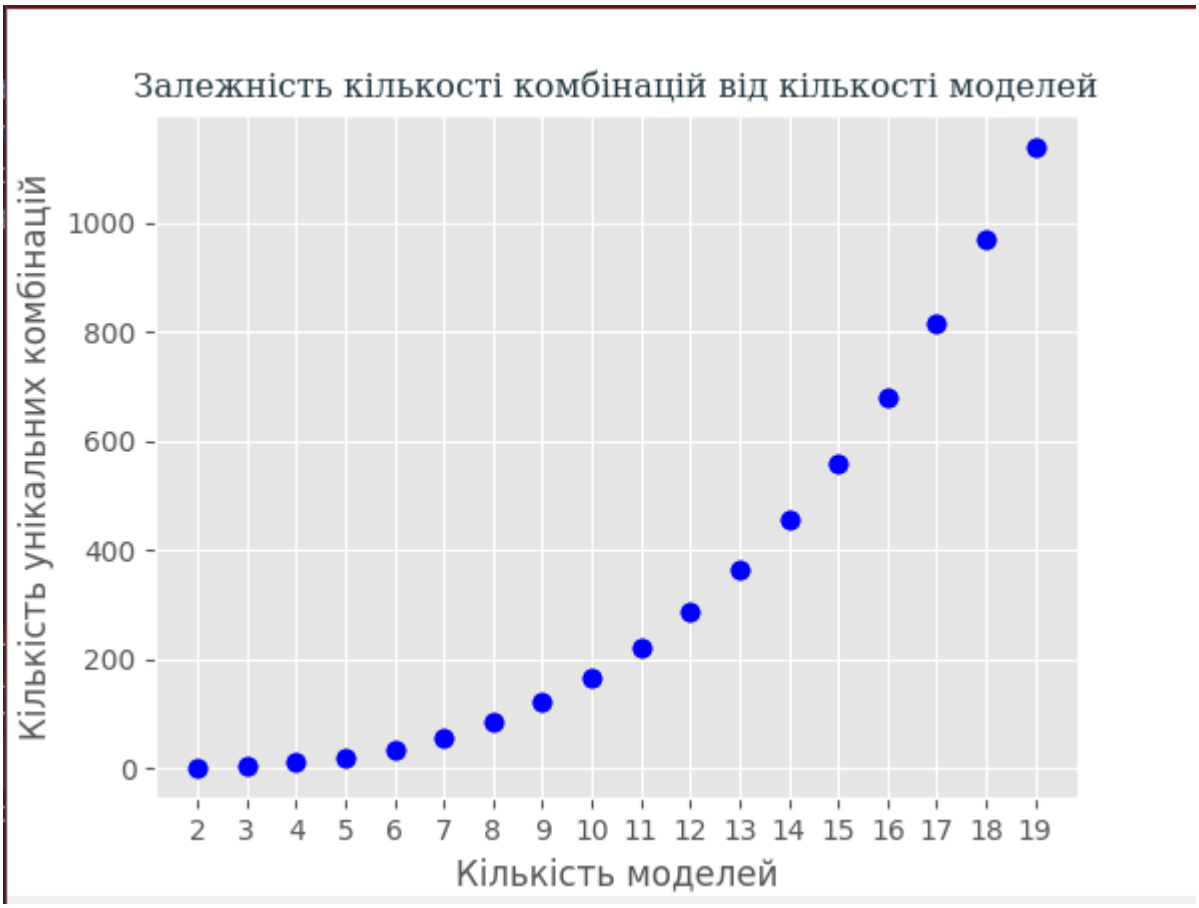


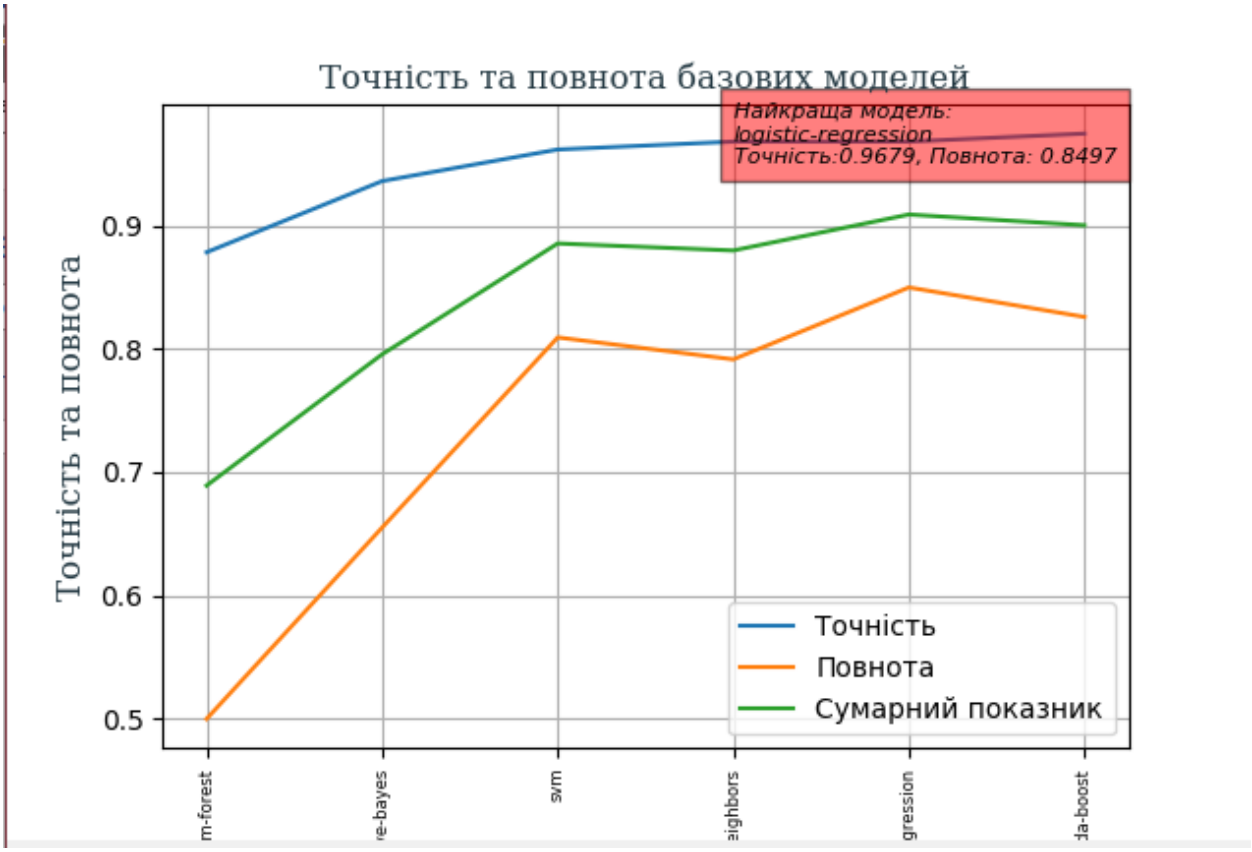
Додаток Г

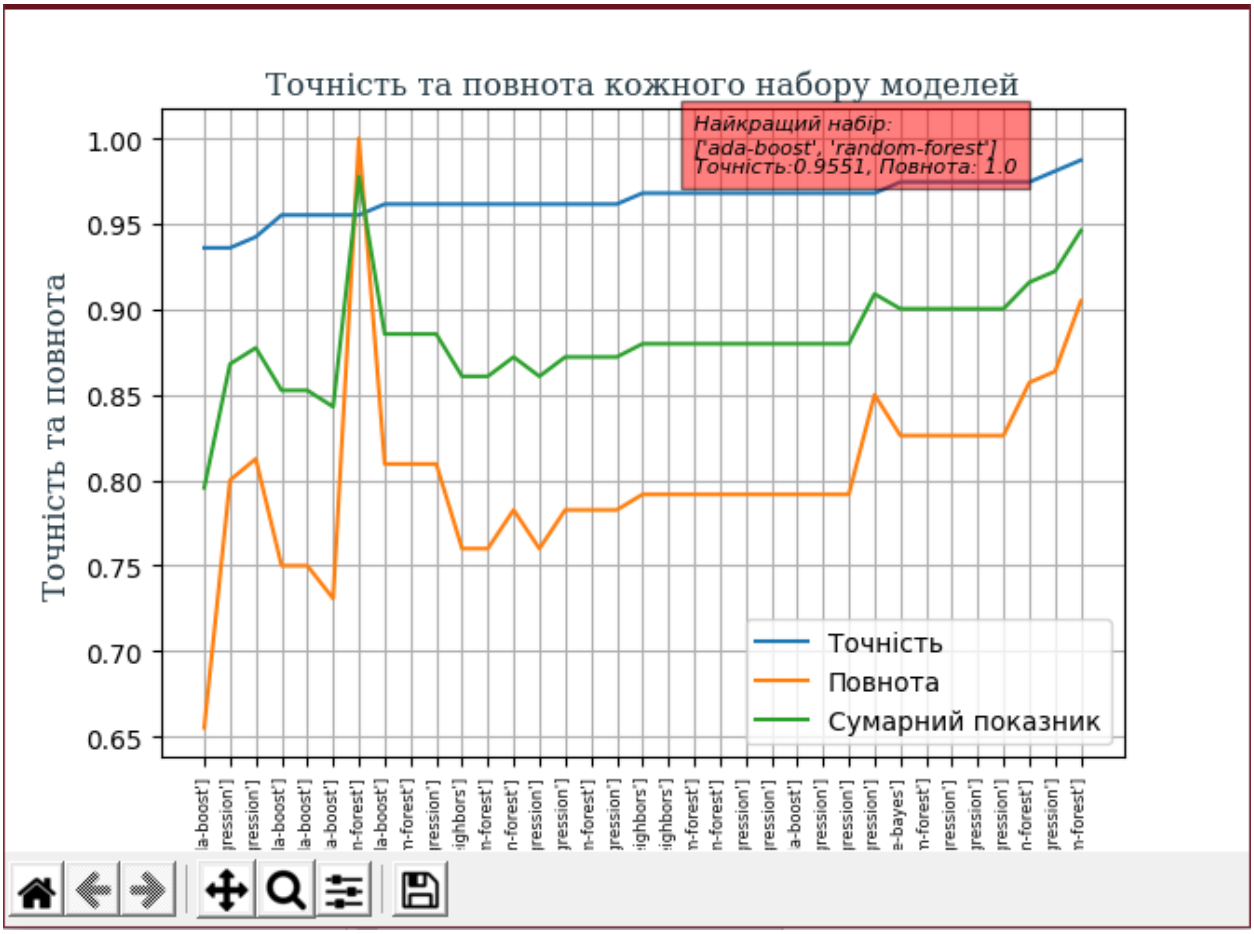
Презентаційний матеріал



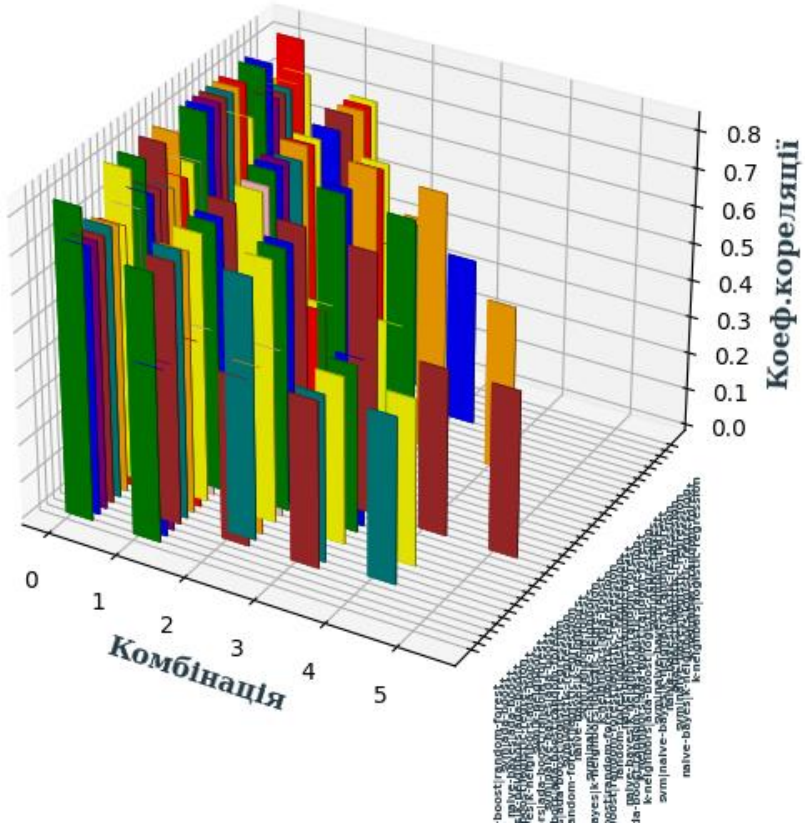

Початок
Завантаження даних з корпусу
Створення міток для навчання і тестування
Фільтрація даних від пустих значень
Призначення міток даним
Розподіл даних на навчальні та тестувальні
Яка мітка ?
Навчальна
Тестувальна
Формування навчального набору даних
Навчання моделей
Формування тестового набору даних
Результати передбачень моделей
Оцінка точності
Кінець

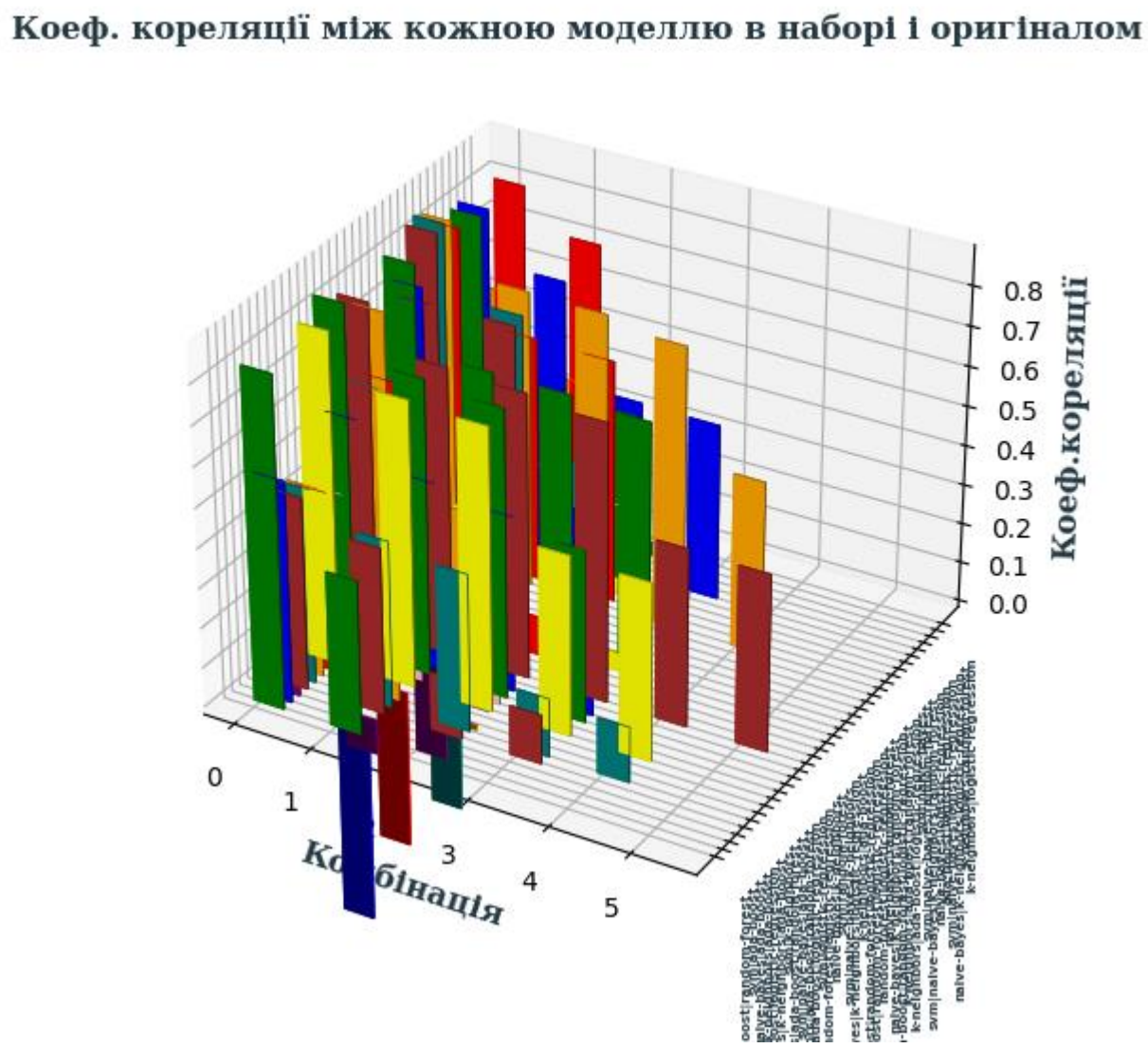

Залежність кількості комбінацій від кількості моделей
Кількість унікальних комбінацій
0
200
400
600
800
1000
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Кількість моделей


Точність та повнота базових моделей
Найкраща модель:
logistic-regression
Точність:0.9679, Повнота: 0.8497
Точність та повнота
0.5
0.6
0.7
0.8
0.9
Точність
Повнота
Сумарний показник
m-forest
e-bayes
svm
ighbors
gression
la-boost



Коеф. кореляції між кожною моделлю в наборі і оригіналом




Коеф. кореляції між кожною моделлю в наборі і оригіналом
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Коеф.кореляції
0
1
3
4
5
Комбінація

Додаток Д

Програмні коди

Лістинг № 1 classify_ensemble.py

```
import sys
import time
import os
from copy import deepcopy
import matplotlib.pyplot as plt
import data_generation
import filters
import utils
from data_preprocessor import get_test_docs, get_train_docs
from models_data.classifier_data import Classifier
from kmeans_processor import kmeans_process
from models_data.models import classifiers
from universal_permutator import permutate_universally
from model_evaluator import evaluate_by_important_metrics
from graph_building.correlation_btw_model_and_origin_labels_graph import build_correlation_graphs2d , build_correlation_graphs3d
from graph_building.models_set_accuracy_recall_graph import build_accuracy_recall_graph
from graph_building.base_models_accuracy_recall_graph import build_accuracy_recall_graph_base
from graph_building.combination_dependence_graph import build_combination_dependence_graph

def prediction_function(vectorizer, tfidf_transformer, classifier_data, classifier_name, corpus_test):
    x_test_counts = vectorizer.transform(corpus_test)
    x_test = tfidf_transformer.transform(x_test_counts)

    # Масив передбачень для тестового корпусу даних
    prediction = classifier_data.predict(x_test)
    # Масив ймовірностей передбачень для тестового корпусу даних
    predict_proba = classifier_data.predict_proba(x_test)
    # Вибір істинної ймовірності з масиву, який містить значення ймовірності відношення до topic, non-topic
    filter_proba = [max(pair_value) for pair_value in predict_proba]

    return prediction , filter_proba


def test_and_save(topics, ref_docs, vectorizer,tfidf_transformer, classifier_data, classifier_name):
    start_time = time.time()
    corpus_test = []
    labels = []
    indexed_data_newid = []
    for doc in ref_docs:
        corpus_test.append(doc[1])
        labels.append(doc[0])
        indexed_data_newid.append(doc[3])
    prediction, probabilities = prediction_function(vectorizer,tfidf_transformer , classifier_data,classifier_name, corpus_test)
    print("Testing time of " + classifier_name + " takes %.2f seconds." % (time.time() - start_time))
    status_prediction = []
    count_pred_right = 0
    count_pred_right_topic = 0
    count_pred_right_non_topic = 0
    for i in range(0, len(prediction)):
        if labels[i] == prediction[i]:
            status_prediction.append(1)
            count_pred_right += 1
            if labels[i] == topics[0]:
                count_pred_right_topic += 1
            if labels[i] == topics[1]:
                count_pred_right_non_topic += 1
```

```
        else:
            status_prediction.append(0)

    cls = Classifier(
        type_classifier=classifier_name,
        topic=topics[0],
        index_data=indexed_data_newid,
        prediction=prediction,
        result_classified=status_prediction,
        count_pred_right = count_pred_right,
count_pred_right_topic=count_pred_right_topic,
count_pred_right_non_topic=count_pred_right_non_
topic,
        prediction_proba= probabilities,
        labels = labels
    )

    outdir_in = 'test'

    utils.save_indexed_data_to_file(cls,
outdir_in)
    return cls

def         train(docs_train,         vectorizer,
classifier_name, classifier):

    y = [d[0] for d in docs_train]
    corpus = [d[1] for d in docs_train]
    X = vectorizer.transform(corpus)
    print("Training      with      %s..."      %
(classifier_name))
    start_time = time.time()
    classifier.fit(X, y)
    print("Training   time   %.2f   seconds."   %
(time.time() - start_time))
    return classifier



def   teach_classifiers(docs_train,   vectorizer,
topic, classifiers):
    start_time = time.time()
    # Вказуємо labels як topic і non-topic
    topics = [topic, "non-" + topic]
    print("Teaching of classifiers starts\n")
    print("----------",  "Labels  ",  topics[0],
topics[1], "----------")
    # Ініціалізація словника класифікаторів
    classifier_name_classifier_data            =
{classifier_name:classifier_name             for
classifier_name in classifiers.keys()}
    teached_models = dict()
    for    classifier_name,    classifier    in
classifiers.items():
        print("Using %s Classifier for teach\n"
% classifier_name)
        classifier_data                        =
train(docs_train,vectorizer,    classifier_name,
classifier)

classifier_name_classifier_data[classifier_name]
= classifier_data


    teached_models                             =
deepcopy(classifier_name_classifier_data)

    print("Teaching  of  classifiers  takes  %.2f
seconds.\n" % (time.time() - start_time))
    return teached_models

def                  test_classifiers(docs_test,
vectorizer,tfidf_transformer,             topic,
classifiers):
    start_time = time.time()
    #  створюється  контейнер  для  результатів
передбачень класифікаторів
    classifiers_results = []
    # Вказуємо labels як topic і non-topic
    topics = [topic, "non-" + topic]
    print("Testing of classifiers starts\n")
    print("----------",  "Labels  ",  topics[0],
topics[1], "----------")
    for    classifier_name,    classifier    in
classifiers.items():
        print("Using %s Classifier for test\n" %
classifier_name)
        cls  =  test_and_save(topics,  docs_test,
vectorizer,    tfidf_transformer,    classifier,
classifier_name)
        classifiers_results.append(cls)
```

```
    print("Testing of classifiers takes %.2f seconds.\n" % (time.time() - start_time))
    return classifiers_results


def teach_test_conveyor(docs_filtered, all_topics, k_train_tag,k_test_tag,
                        trained_vectorizer, trained_tfidf_transformer):
    # for topic in all_topics:
    # розмічає данні на два типи 'topic' та 'non-topic'
    docs, count_topic, count_non_topic = filters.labeled_docs_by_topics(all_topics[0], docs_filtered)

    # корпус для тренування
    train_docs = get_train_docs(docs, k_train_tag)
    # корпус для тестування
    test_docs = get_test_docs(docs, k_test_tag)
    # тренування
    if os.path.isfile('dump/list_models.pkl') == False:
        Dict_models = teach_classifiers(train_docs, trained_vectorizer, all_topics[0], classifiers)
        utils.dump_teached_data_save(Dict_models, 'dump/list_models.pkl')
        dict_models = utils.dump_teached_data_load('dump/list_models.pkl')
    else:
        dict_models = utils.dump_teached_data_load('dump/list_models.pkl')
    # тестування
    classifiers_results = test_classifiers(test_docs, trained_vectorizer,
                                           trained_tfidf_transformer, all_topics[0], dict_models)

    return classifiers_results


def main(argv):
    start_time = time.time()
    utils.del_directiry('test')
    # завантажує дані, завантажує категорії, завантажує мітки даних (для навчання і для тренування),
    # відфільтровує дані від пустих значень, вчить vectorizer, tfidf_transformer на відфільтрованих даних
    types, all_topics, docs_filtered , trained_vectorizer , trained_tfidf_transformer = data_generation.get_data(argv)
    k_train_tag = types[0]
    k_test_tag = types[1]

    # "Навчання і тестування моделей по найбільш популярним темам"
    classifiers_results = teach_test_conveyor(docs_filtered , all_topics, k_train_tag, k_test_tag,
                                              trained_vectorizer, trained_tfidf_transformer)

    original_labels = classifiers_results[0].labels
    every_combination_models_predictions = permutate_universally(classifiers_results, kmeans_process_func=kmeans_process)

    every_combination_models_predictions_evaluated = evaluate_by_important_metrics(original_labels, every_combination_models_predictions)

    build_correlation_graphs2d(every_combination_models_predictions_evaluated)
    # build_correlation_graphs3d(every_combination_models_predictions_evaluated)
    # build_accuracy_recall_graph(every_combination_models_predictions_evaluated)
    # build_accuracy_recall_graph_base(classifiers_results)
    # build_combination_dependence_graph()
```

```
    plt.show()
    print("Total runtime %.2f seconds.\n" %
(time.time() - start_time))


if __name__ == "__main__":
    main(sys.argv[0:])
```

Лістинг № 2 data_generation.py

```
import os
import pickle

from filters import filter_docs_from_voids
from data_preprocessor import
get_vectorizer_tfidf_transformer
from reuters_parser import ReutersParser

def get_most_important_topics():
    topics =
open("data/most_popular_topics.txt",
"r").readlines()
    topics = [t.strip() for t in topics]
    for i in range(0, len(topics)):
        topics[i] = topics[i].lower()
    return topics

def obtain_topic_tags():
    """
    Open the topic list file and import all of
the topic names
    taking care to strip the trailing "\n" from
each word.
    """
    topics = open("data/all-topics-
strings.lc.txt", "r").readlines()
    topics = [t.strip() for t in topics]
    for i in range(0, len(topics)):
        topics[i] = topics[i].lower()
    return topics

def obtain_train_test_tags():
    types = open("data/all-cgisplit.txt",
"r").readlines()
    types = [t.strip() for t in types]
    for i in range(0, len(types)):
        types[i] = types[i].lower()
    return types

def load_data():
    # Create the list of Reuters data and create
the parser
    if os.path.isfile('dump/docs_parsed.pkl'):
        input_f = open('dump/docs_parsed.pkl',
'rb')
        docs = pickle.load(input_f)
        input_f.close()
    else:
        #                       joblib.dump(svm,
"saved_data/saved_trained_data.pkl")
        # files = ["data/reut2-%03d.sgm" % r for
r in range(0, 22)]
        files = ["data/reut2-%03d.sgm" % r for r
in range(9, 16)]
        parser = ReutersParser()
        # Parse the document and force all
generated docs into
        # a list so that it can be printed out
to the console
        print("Parsing training data...\n")
        docs = []
        for fn in files:
            for d in parser.parse(open(fn,
'rb')):
                docs.append((d[0], d[1], d[2],
int(d[3])))

        output = open('dump/docs_parsed.pkl',
'wb')
        pickle.dump(docs, output)
        output.close()
    return docs

def get_data(argv):
    # завантажує дані
    all_docs = load_data()
    # завантажує категорії
    all_topics = get_most_important_topics()
    # завантажує мітки даних (для навчання і для
тренування)
    types = obtain_train_test_tags()
```

```
    # відфільтровуєм дані від пустих значень
    docs_filtered = filter_docs_from_voids(types, all_docs)
    # вчить vectorizer, tfidf_transformer на відфільтрованих даних
    trained_vectorizer, trained_tfidf_transformer = get_vectorizer_tfidf_transformer(docs_filtered, argv)

    return types, all_topics, docs_filtered , trained_vectorizer , trained_tfidf_transformer
```

Лістинг № 3 data_preprocessor.py

```
from stemmed_tfidfvectorizer import StemmedTfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

def get_train_docs(docs, train_tag):
    docs_train = [doc for doc in docs if doc[2].decode() == train_tag]
    return docs_train

def get_test_docs(docs, test_tag):
    docs_test = [doc for doc in docs if doc[2].decode() == test_tag]
    return docs_test

def get_vectorizer_tfidf_transformer(docs, argv):
    """
    Creates a document corpus list (by stripping out the
    class labels), then applies the TF-IDF transform to this
    list.

    The function returns both the class label vector (y) and
    the corpus token/feature matrix (X).
    """
    # фільтрується списки за типом запису, залишаються тільки ті, які відповідають doc_type
    # docs = [doc for doc in docs if doc[2].decode() == doc_type]
    # Create the training data class labels
    # створюється масив міток чи відноситься до теми чи ні

    # Create the document corpus list
    # створюється масив документів без міток довжина відповідає масиву y.
    corpus = [d[1] for d in docs]

    # Create the TF-IDF vectoriser and transform the corpus
    if len(argv) == 1:
        print("\nWith remove stopwords and stemming")
        vectorizer = StemmedTfidfVectorizer(stop_words="english", \
analyzer="word", \
min_df=1)
    if len(argv) > 1:
        if "--no-stemming" == argv[1]:
            print("Without stemming")
            vectorizer = TfidfVectorizer(stop_words="english", \
analyzer="word", \
min_df=1)
        else:
            if "--no-stopwords" == argv[1]:
                print("Without removing stopwords")
                vectorizer = TfidfVectorizer(analyzer="word", min_df=1)
            else:
                print("With remove stopwords and stemming")
                vectorizer = StemmedTfidfVectorizer(stop_words="english", \
analyzer="word", \
```

```
min_df=1)

    x = vectorizer.fit_transform(corpus)
    tfidf_transformer = TfidfTransformer()
    x_train_tfidf                                =
tfidf_transformer.fit_transform(x)

    return vectorizer, tfidf_transformer
```

Лістинг № 4 filters.py

```
def labeled_docs_by_topics(topic, docs):
    # розмічає дані на два типи label
    # ті, які відносяться до теми і ті, які не
відносяться
    count_topic = 0
    count_non_topic = 0
    ret_docs = []
    for doc in docs:
        if len(doc) < 3:
            continue
        label = "non-" + topic
        for topic_entry in doc[0]:
            if topic_entry == topic:
                label = topic
                count_topic += 1
                break
        ret_docs.append((label,  doc[1],  doc[2],
doc[3]))
    count_non_topic = len(docs) - count_topic
    return         ret_docs,         count_topic,
count_non_topic

def filter_docs_from_voids(types, docs):
    # відфільтровуєм від пустих значень
    # d[1] == ''
    # відфільтровує ті, які не мають незви теми
(topic),
    # d[0] == ""
    # відфільтровує ті, в яких тип не відповідає
заданому типу (types) (для тренування чи
перевірки-тестування)
    # d[2].decode() not in types
    ret_docs = []
    for d in docs:
        if d[0] == [] or d[0] == "":
            continue
        if d[1] == '':
            continue
        if d[2].decode() not in types:
            continue

        ret_docs.append((d[0],    d[1],    d[2],
d[3]))

    return ret_docs
```

Лістинг № 5 kmeans_process.py

```
from sklearn.cluster import KMeans
from  models_data.models_combination_data  import
ModelsSetData

def
select_non_intersectioned_predictions(classifier
s_results):
    non_intersection_predictions_dict          =
{classifier.type_classifier:      list()      for
classifier in classifiers_results}
    non_intersection_probabilities_dict        =
{classifier.type_classifier:      list()      for
classifier in classifiers_results}
    #   Запис   результатів   класифікаторів   в
dictionary
    predictions_dict                           =
{classifier.type_classifier:
classifier.prediction    for    classifier    in
classifiers_results}
    probabilities_dict                         =
{classifier.type_classifier:
classifier.prediction_proba  for  classifier  in
classifiers_results}
    # контейнер для id документів, передбачення
над якими не збіглися у моделей
    non_intersection_document_id = []
    #  контейнер істинних відповідей, містить
тільки ті елементи,
```

```
    # індекси яких такі ж як у id документів, передбачення над якими не збіглися у моделей
    non_intersection_original_labels = []
    results_count = len(classifiers_results[0].result_classified)
    for i in range(0, results_count):
        one_result_of_classifiers = []
        for results in predictions_dict.values():
            one_result_of_classifiers.append(results[i])
        if len(set(one_result_of_classifiers)) == 1:
            continue
        else:
            non_intersection_document_id.append(classifiers_results[0].index_data[i])
            non_intersection_original_labels.append(classifiers_results[0].labels[i])
            for classifier_name in non_intersection_predictions_dict.keys():
                non_intersection_predictions_dict[classifier_name].append(predictions_dict.get(classifier_name)[i])
                non_intersection_probabilities_dict[classifier_name].append(probabilities_dict.get(classifier_name)[i])

    return distribute_probabilities(non_intersection_predictions_dict,non_intersection_probabilities_dict), \
           non_intersection_document_id, non_intersection_original_labels


def distribute_probabilities(non_intersection_predictions_dict,
non_intersection_probabilities_dict):
    # розділяємо ймовірності на додатні та від'ємні для відповідних класів (non-topic: -;topic: +;)
    predictions_container = [predictions for predictions in non_intersection_predictions_dict.values()]
    probabilities_container = [probabilities for probabilities in non_intersection_probabilities_dict.values()]

    sub_str = "non"
    for i in range(0, len(probabilities_container[0])):
        for j in range(0, len(probabilities_container)):
            if sub_str in predictions_container[j][i]:
                probabilities_container[j][i] = -1 * probabilities_container[j][i]
    return non_intersection_probabilities_dict


def kmeans_process(classifiers_results):
    # створюємо модель для кластеризації даних
    model = KMeans(n_clusters=2, random_state=0)
    # контейнер для передбачень класифікаторів, які не пересіклись
    probabilities_container_dict, documents_id_container, original_labels_container = select_non_intersectioned_predictions(classifiers_results)
    # створюємо контейнер для центроїдів
    centroids_container = [[], []]
    probabilities_container = [probabilities_lst for probabilities_lst in probabilities_container_dict.values()]
    for i in range(0, len(probabilities_container[0])):
        tmp_prediction = []
        cluster_centers = []
        index = 0
        for prediction in probabilities_container:
            tmp_prediction.append([])
            tmp_prediction[index].append(prediction[i])
            index += 1

        model.fit(tmp_prediction)
```

```
        cluster_centers = model.cluster_centers_
centroids_container[0].append(cluster_centers[0][0])

centroids_container[1].append(cluster_centers[1][0])
    # контейнер, в якому зберігаються всі передбачення однієї з моделей
    # пізніше в ньому заміняться індекси тих передбачень, які були однакові
    # у результуючому наборі наборі моделей.
    # значення передбачень на замінюваних індексах будуть оптимізовані згідно кожного набору моделей
    raw_prediction = classifiers_results[0].prediction
    index_data = classifiers_results[0].index_data
    indexed_raw_prediction_data = dict()
    for i in range(0, len(index_data)):
        indexed_raw_prediction_data[index_data[i]] = raw_prediction[i]

    indexed_model_combination_prediction_dict = determine_prediction_by_centroids_average(centroids_container,
                                                                                           classifiers_results[0].topic,
                                                                                           documents_id_container)
    optimized_predictions_dict = replace_raw_prediction(indexed_raw_prediction_data, indexed_model_combination_prediction_dict)
    models_names = [classifier.type_classifier for classifier in classifiers_results]
    models_set = ModelsSetData(
        classifiers_names= models_names,
        non_intersection_document_id= documents_id_container,
        probabilities_container_dict = probabilities_container_dict,
        optimized_predictions=optimized_predictions_dict,
        original_labels=original_labels_container
    )
    return models_set


def replace_raw_prediction(indexed_raw_prediction_data, indexed_model_combination_prediction_dict):
    optimized_prediction = dict()
    for key, value in indexed_raw_prediction_data.items():
        optimized_prediction[key] = value
        for k, v in indexed_model_combination_prediction_dict.items():
            if (k == key):
                optimized_prediction[key] = v

    return optimized_prediction


def calculate_centroids_average(centroids_container):
    centroids_average = []
    for i in range(0, len(centroids_container[0])):
        sum = 0
        for centroids_list in centroids_container:
            sum += centroids_list[i]
        centroids_average.append(sum / len(centroids_container))

    return centroids_average


def distribute_topic_non_topic(centroids_average, topic):
    # створюється контейнер, в якому будуть записані відповіді topic-non-topic
    # згідно середнього значення центроїдів ймовірностей передбачень для окремої вибірки моделей
    topic_non_topic_lst = []
```

```
    for i in centroids_average:
        if (i <= 0):
            topic_non_topic_lst.append('non-' + topic)
        else:
            topic_non_topic_lst.append(topic)
    return topic_non_topic_lst


def determine_prediction_by_centroids_average(centroids_container, topic, documents_id_container):
    centroids_average = calculate_centroids_average(centroids_container)
    model_combination_prediction = distribute_topic_non_topic(centroids_average, topic)
    indexed_model_combination_prediction_dict = dict()
    for i in range(0, len(documents_id_container)):

        indexed_model_combination_prediction_dict[documents_id_container[i]] = model_combination_prediction[i]

    return indexed_model_combination_prediction_dict
```

Лістинг № 6 model_evaluator.py

```
from sklearn.metrics import accuracy_score, confusion_matrix

def evaluate_by_important_metrics(originals_labels, every_combination_models_predictions):

    every_combination_models_predictions_evaluated = dict()
    for key, value_obj in every_combination_models_predictions.items():
        tmp_predictions = []
        for v in value_obj.optimized_predictions.values():
            tmp_predictions.append(v)
        accuracy = accuracy_score(y_true=originals_labels, y_pred=tmp_predictions)
        # recall = recall_score(y_true = originals_labels, y_pred = tmp_predictions, average=None)
        confusion_m = confusion_matrix(y_true=originals_labels, y_pred= tmp_predictions)
        value_obj.accuracy = round(accuracy, 4)
        value_obj.recall = round(confusion_m[0][0] / (confusion_m[0][0] + confusion_m[1][0]), 4)
        value_obj.confusion_matrix = confusion_m

    every_combination_models_predictions_evaluated[key] = value_obj

    sorted_dict = {k:v for k,v in sorted(every_combination_models_predictions_evaluated.items(),
                                         key = lambda item: item[1].accuracy)}
    return  sorted_dict
```

Лістинг № 7 reuters_parser.py

```
import re

from html.parser import HTMLParser


class ReutersParser(HTMLParser):
    """
    ReutersParser subclasses HTMLParser and is used to open the SGML
    files associated with the Reuters-21578 categorised test collection.

    The parser is a generator and will yield a single document at a time.
    Since the data will be chunked on parsing, it is necessary to keep
```

```
    some internal state of when tags have been
"entered" and "exited".
    Hence the in_body, in_topics and in_topic_d
boolean members.
    """
    def __init__(self, encoding='latin-1'):
        """
        Initialise the superclass (HTMLParser)
and reset the parser.
        Sets the encoding of the SGML files by
default to latin-1.
        """
        HTMLParser.__init__(self)
        self._reset()
        self.encoding = encoding

    def _reset(self):
        """
        This is called only on initialisation of
the parser class
        and when a new topic-body tuple has been
generated. It
        resets all off the state so that a new
tuple can be subsequently
        generated.
        """
        self.in_body = False
        self.in_topics = False
        self.in_topic_d = False
        self.in_reuters = False
        self.body = ""
        self.topics = []
        self.topic_d = ""
        self.reuters = ""
        self.cgisplit = ""
        self.newid = ""

    def parse(self, fd):
        """
        parse accepts a file descriptor and
loads the data in chunks
        in order to minimise memory usage. It
then yields new documents
        as they are parsed.
        """
        self.docs = []
        for chunk in fd:
self.feed(chunk.decode(self.encoding))
            for doc in self.docs:
                yield doc
            self.docs = []
        self.close()

    def handle_starttag(self, tag, attrs):
        """
        This method is used to determine what to
do when the parser
        comes across a particular tag of type
"tag". In this instance
        we simply set the internal state
booleans to True if that particular
        tag has been found.
        """
        if tag == "reuters":
            #pass
            self.in_reuters = True
            for attribute in attrs:
                if attribute[0] == "cgisplit":
                    self.cgisplit               =
attribute[1].encode("utf-8").lower()
                if attribute[0] == "newid":
                    self.newid = attribute[1]
                    #break
        elif tag == "body":
            self.in_body = True
        elif tag == "topics":
            self.in_topics = True
        elif tag == "d":
            self.in_topic_d = True

    def handle_endtag(self, tag):
        """
        This method is used to determine what to
do when the parser
        finishes with a particular tag of type
"tag".

        If the tag is a  tag, then we remove all
        white-space with a regular expression
and then append the
        topic-body tuple.
```

```
        If the tag is a  or  tag then we simply set
        the internal state to False for these booleans, respectively.

        If the tag is a  tag (found within a tag), then we
        append the particular topic to the "topics" list and
        finally reset it.
        """
        if tag == "reuters":
            self.body  =  re.sub(r'\s+',  r'  ', self.body)
            self.in_reuters = False
            self.docs.append(      (self.topics, self.body, self.cgisplit, self.newid) )
            self._reset()
        elif tag == "body":
            self.in_body = False
        elif tag == "topics":
            self.in_topics = False
        elif tag == "d":
            self.in_topic_d = False
            self.topics.append(self.topic_d)
            self.topic_d = ""


    def handle_data(self, data):
        """
        The  data  is  simply  appended  to  the appropriate member state
        for  that  particular  tag,  up  until  the end closing tag appears.
        """
        if self.in_body:
            #self.body += data.encode("utf-8")
           self.body += data
        elif self.in_topic_d:
         #     self.topic_d  +=  data.encode("utf-8")
         #                    self.topic_d     += data.encode(encoding="utf-8")
           self.topic_d += data
```

Лістинг№8stemmed_tfidfvectorizer.py

```
import nltk.stem
from    sklearn.feature_extraction.text    import TfidfVectorizer

class StemmedTfidfVectorizer(TfidfVectorizer):
    def build_analyzer(self):
        english_stemmer                          = nltk.stem.SnowballStemmer('english')
        analyzer                                 = super(TfidfVectorizer,self).build_analyzer()
        return            lambda            doc: (english_stemmer.stem(w) for w in analyzer(doc))
```

Лістинг№9 universal_permutator.py

```
from copy import copy

def  permutate_universally(classifiers_results  , kmeans_process_func):
    # стоворимо словник для запису відповідей для кожної комбінації моделей
    every_combination_models_predictions        = dict()
    #  створення  всіх  можливих  універсальних комбінацій результатів класифікаторів
    for i in range(0, len(classifiers_results)):
        tmp_lst = []
        tmp_lst.append(classifiers_results[i])
        for     j     in     range(i     +     1, len(classifiers_results)):

tmp_lst.append(classifiers_results[j])
            for       k       in       range(j, len(classifiers_results)):
                copy_tmp = copy(tmp_lst)
                copy_tmp[j      -      i]      = classifiers_results[k]
                #  назви  моделей  на  поточній комбінації
                combination_set                 = [model.type_classifier for model in copy_tmp]
                combination_set_str             = '|'.join(combination_set)
```

```
every_combination_models_predictions[combination
_set_str] = kmeans_process_func(copy_tmp)



    return every_combination_models_predictions
```

Лістинг№10 utils.py

```
import os
import shutil
import pickle
from pathlib import Path



def dump_teached_data_load(path_load):
    input_f = open(path_load, 'rb')
    list_cls = pickle.load(input_f)
    input_f.close()
    return list_cls

# запис результатів навчання в dump
def dump_teached_data_save(list_cls, path_save):
    output = open(path_save, 'wb')
    pickle.dump(list_cls, output)
    output.close()

# видалення директорій з файлами в них
def del_directiry(outdir_in):
    currentdir = os.getcwd()
    outdir = 'datesaved'
    #outdir_in = 'tmp'
    if os.path.isdir(outdir) == True:
        if   os.path.isdir(outdir   +   '/'   +
outdir_in) == True:
            shutil.rmtree(outdir   +   '/'   +
outdir_in, ignore_errors=True)

# запис результатів тестування в файли по
категоріям
def save_indexed_data_to_file(cls, outdir_in):
    indent = 20
    print("Saved to file...")
    first_mode_dir = os.getcwd()
    currentdir = os.getcwd()
    outdir = 'datesaved'
    # outdir_in = 'test'
    parentdir = Path(currentdir).stem
    if parentdir != outdir_in:
        if os.path.isdir(outdir) == False:
            os.mkdir(outdir)
        if   os.path.isdir(outdir   +   '/'   +
outdir_in) == False:
            os.mkdir(outdir + '/' + outdir_in)
        os.chdir(currentdir + '/' + outdir + '/'
+ outdir_in)
    dir = cls.type_classifier
    if os.path.isdir(dir) == False:
        os.mkdir(dir)
    file_name = dir + '/' + cls.topic + '.txt'
    outfile      =      open(file_name,      'w',
encoding='utf8')
    outfile.write(cls.type_classifier + '\n')
    outfile.write(cls.topic + '\n')
    outfile.write(str(cls.index_data) + '\n')
    outfile.write(str(cls.prediction_proba)    +
'\n')
    outfile.write(str(cls.result_classified)   +
'\n')
    outfile.close()
    os.chdir(first_mode_dir)
```

Лістинг№11 classifier_data.py

```
import numpy as np
from   sklearn.metrics   import   accuracy_score,
confusion_matrix

class Classifier():
    def  __init__(self,  type_classifier,  topic,
index_data, prediction,
             result_classified,
count_pred_right, count_pred_right_topic,
             count_pred_right_non_topic,
prediction_proba, labels):
        self.type_classifier = type_classifier
        self.topic = topic
        self.index_data = index_data
        self.prediction = prediction
```

```
        self.result_classified                  =
result_classified
        self.count_pred_right = count_pred_right
        self.count_pred_right_topic             =
count_pred_right_topic
        self.count_pred_right_non_topic         =
count_pred_right_non_topic
        self.prediction_proba = prediction_proba
        self.labels = labels
        self.accuracy                           =
round(accuracy_score(y_true=labels,       y_pred=
prediction), 4)
        self.confusion_matrix                   =
np.array(confusion_matrix(y_true=labels,  y_pred=
prediction)) + 0.01
        self.recall                             =
round(self.confusion_matrix[0][0]               /
(self.confusion_matrix[0][0]                    +
self.confusion_matrix[1][0]), 4)
```

Лістинг№12 models.py

```
from sklearn.naive_bayes import MultinomialNB
from           sklearn.neighbors           import
KNeighborsClassifier
from sklearn.svm import SVC
from            sklearn.ensemble            import
RandomForestClassifier, AdaBoostClassifier
from          sklearn.linear_model          import
LogisticRegression

classifiers = \
    {
        "svm":      SVC(C=1.0,      gamma='auto',
kernel='linear',         class_weight='balanced',
probability=True),
        "naive-bayes":
MultinomialNB(alpha=0.01),
        "k-neighbors":
KNeighborsClassifier(n_neighbors=30,
weights='uniform'),
        "ada-boost": AdaBoostClassifier(),
        "random-forest":
RandomForestClassifier(n_estimators=200,
max_depth=3, random_state=0),
        "logistic-regression":
LogisticRegression()
    }
```

Лістинг№13 models_combination_data.py

```
class ModelsSetData():
  def      __init__(self,      classifiers_names,
non_intersection_document_id,probabilities_conta
iner_dict,
                optimized_predictions,
original_labels):

        self.classifiers_names                  =
classifiers_names
        self.itersection_document_id            =
non_intersection_document_id
        self.probabilities_container_dict       =
probabilities_container_dict
        self.optimized_predictions              =
optimized_predictions
        self.original_labels = original_labels
        self.accuracy = 0
        self.recall = 0
        self.confusion_matrix = 0
        probabilities                           =
probabilities_container_dict.values()
        sum = 0
        count = 0
        for i in probabilities:
              for j in i:
                  sum+=j
                  count+=1
        self.average_prob  =  round(sum  /  count,
3)
```

Лістинг№14 base_models_accuracy_recall_graph.py

```
import matplotlib.pyplot as plt

def
build_accuracy_recall_graph_base(classifiers_res
ults):
```

```
    sorted_cls = sorted(classifiers_results , key=lambda x: x.accuracy, reverse=False)
    accuracies = []
    recalls = []
    models_names = []
    accuracy_recall_averages = []
    for classifier in sorted_cls:
        accuracies.append(classifier.accuracy)
        recalls.append(classifier.recall)

models_names.append(classifier.type_classifier)

accuracy_recall_averages.append((classifier.recall + classifier.accuracy) / 2)
    font = {'family': 'serif',
            'color': '#233842',
            'weight': 'normal',
            'size': 12,
            }

    fig = plt.figure(figsize=(5, 3), dpi=100)
    ax = fig.add_subplot(111)
    ax.plot(models_names, accuracies, label='Точність')
    ax.plot(models_names, recalls, label='Повнота')
    ax.plot(models_names, accuracy_recall_averages, label='Сумарний показник')
    ax.tick_params(axis='x', rotation=90, labelsize=6)
    ax.set_xlabel('Комбінації моделей', fontdict=font)
    ax.set_ylabel('Точність та повнота', fontdict=font)
    ax.set_title('Точність та повнота базових моделей', fontdict=font)
    ax.text(3, 0.95,
            f'Найкраща модель:\n{models_names[accuracy_recall_averages.index(max(accuracy_recall_averages))]}'

f'\nТочність:{accuracies[accuracy_recall_averages.index(max(accuracy_recall_averages))]}, Повнота: {recalls[accuracy_recall_averages.index(max(accuracy_recall_averages))]}'
            , style='italic', fontsize=8,
            bbox={'facecolor': 'red', 'alpha': 0.5, 'pad': 5})

    ax.legend()
    ax.grid()
```

Лістинг№15

combination_dependence_graph.py

```
import matplotlib.pyplot as plt

def y_function(x):
    y = 0
    for i in range(0, x):
        for j in range(i + 1, x):
            for k in range(j, x):
                y += 1
    return y

def build_combination_dependence_graph():
    font = {'family': 'serif',
            'color': '#233842',
            'weight': 'normal',
            'size': 12,
            }
    models_count_lst = range(2, 20)
    combinations_count_lst = [y_function(count) for count in models_count_lst]
    plt.style.use('ggplot')
    fig = plt.figure(figsize=(5, 3), dpi=100)
    ax = fig.add_subplot(111)
    ax.plot(models_count_lst, combinations_count_lst, 'bo')
    ax.set_xlabel('Кількість моделей')
    ax.set_ylabel('Кількість унікальних комбінацій')
    ax.set_title('Залежність кількості комбінацій від кількості моделей', fontdict=font)
    ax.plot(models_count_lst, combinations_count_lst, 'bo')
    ax.set_xticks(models_count_lst)


def main():
```

```
    build_combination_dependence_graph()

if __name__ == "__main__":
    main()
```

Лістинг№16 correlation_btw_model_and_origin_labels_graph.py

```
import random
import matplotlib.pyplot as plt
import tkinter as tk
from scipy.stats import pearsonr
from pandas import DataFrame
from tkinter import *
from   matplotlib.backends.backend_tkagg   import
FigureCanvasTkAgg
from matplotlib.patches import Rectangle



def
replace_original_labels(original_labels_dict):
    numeric_original_labels = dict()
    sub_str = "non"
    for    models_names,    original_labels    in
original_labels_dict.items():
        numeric_original_labels_tmp = []
        for label in original_labels:
            n = round(random.uniform(0, 0.1), 3)
            if sub_str in label:

numeric_original_labels_tmp.append(-1 - n)
            else:

numeric_original_labels_tmp.append(1 + n)
        numeric_original_labels[models_names]   =
numeric_original_labels_tmp
    return numeric_original_labels


def
calculate_correlation(every_combination_models_p
redictions_evaluated):
    original_labels                                =
{models_name:models_set_obj.original_labels
                          for          models_name,
models_set_obj                                    in
every_combination_models_predictions_evaluated.i
tems()}
    numeric_original_labels                    =
replace_original_labels(original_labels)
    probabilities_of_each_model_in_set         =
[models_set_obj.probabilities_container_dict
for models_set_obj in

every_combination_models_predictions_evaluated.v
alues()]

correlation_coef_btw_each_model_in_set_and_origi
nal_labels = dict()
    index = 0
    for    models_names,    numeric_labels    in
numeric_original_labels.items():
        correlation_coef_dict_tmp = dict()
        if(len(numeric_labels) < 2):
            index += 1
            continue
        for model_name , model_probabilities in
probabilities_of_each_model_in_set[index].items(
):
            coef    =    pearsonr(numeric_labels,
model_probabilities)[0]

correlation_coef_dict_tmp[model_name]          =
round(coef, 3)

correlation_coef_btw_each_model_in_set_and_origi
nal_labels[models_names]                       =
correlation_coef_dict_tmp
        index += 1
    return
correlation_coef_btw_each_model_in_set_and_origi
nal_labels

def
build_correlation_graphs2d(every_combination_mod
els_predictions_evaluated):

correlation_coef_btw_each_model_in_set_and_origi
nal_labels                                     =
calculate_correlation(every_combination_models_p
redictions_evaluated)
```

```
    font = {'family': 'serif',
            'color': '#233842',
            'weight': 'bold',
            'size': 8,
            }
    colors = ['green', 'blue', 'purple', 'brown', 'teal', 'orange']
    root = tk.Tk()
    root.title('Кореляція моделей з істинними значеннями')
    main_frame = Frame(root)
    main_frame.pack(fill=BOTH, expand=1)
    my_canvas = Canvas(main_frame)
    my_canvas.pack(side=LEFT, fill=BOTH, expand=1)
    my_scrollbar = Scrollbar(main_frame, orient=VERTICAL, command=my_canvas.yview)
    my_scrollbar.pack(side=RIGHT, fill=Y)

    my_canvas.configure(yscrollcommand=my_scrollbar.set)
    my_canvas.bind('<Configure>', lambda e: my_canvas.configure(scrollregion=my_canvas.bbox("all")))
    second_frame = Frame(my_canvas)
    my_canvas.create_window((0, 0), window=second_frame, anchor='nw')
    def _on_mouse_wheel(event):
        my_canvas.yview_scroll(-1 * int((event.delta / 120)), "units")
    my_canvas.bind_all("<MouseWheel>", _on_mouse_wheel)


    for models_names, correlation in correlation_coef_btw_each_model_in_set_and_original_labels.items():
        names_and_corr_dict = {'models': list(correlation.keys()), 'corr': list(correlation.values())}
        df = DataFrame(names_and_corr_dict, columns=['models', 'corr'])

        figure = plt.figure(figsize=(5, 3), dpi=100)
        ax = figure.add_subplot(111)
        ax.set_title(models_names, fontdict=font)
        ax.set_xlabel('Моделі', fontdict=font, fontsize=12)
        ax.set_ylabel('Коеф. кореляції між кожною моделлю в наборі і оригіналом', fontdict=font)

        bar = FigureCanvasTkAgg(figure, second_frame, )
        bar.get_tk_widget().pack()

        df = df[['models', 'corr']].groupby('models').sum()
        df = df.sort_values(by=['corr'], ascending=True)
        df.plot(kind='bar', ax=ax, grid=True, legend=False, rot=0, fontsize=8, width=0.4)

        index = 0
        for ch in ax.get_children():
            if isinstance(ch, Rectangle) and ch.xy != (0, 0):
                ch.set_color(colors[index])
                index += 1


    root.mainloop()

def build_correlation_graphs3d(every_combination_models_predictions_evaluated):

    correlation_coef_btw_each_model_in_set_and_original_labels = calculate_correlation(

        every_combination_models_predictions_evaluated)
    font = {'family': 'serif',
            'color': '#233842',
            'weight': 'bold',
            'size': 12,
            }
    colors = ['green', 'blue', 'purple', 'brown', 'teal', 'orange', 'red', 'yellow', 'pink']
    colors3d = []
    fig = plt.figure()
```

```
    ax = fig.add_subplot(111, projection = '3d')
    ax.figure.set_size_inches(7, 6)
    ax.set_xlabel('Комбінація', fontdict=font)
    ax.set_zlabel('Коеф.кореляції', fontdict=font)
    ax.set_title('Коеф. кореляції між кожною моделлю в наборі і оригіналом', fontdict=font)
    x = []
    y = []
    y_labels = []
    z = []
    dx = []
    dy = []
    dz = []

    y_pos = 0
    color_index = 0
    for models_names, correlation in correlation_coef_btw_each_model_in_set_and_original_labels.items():
        y_labels.append(models_names)
        x_pos = 0
        if(color_index == len(colors)):
            color_index = 0
        for corr in correlation.values():
            colors3d.append(colors[color_index])
            x.append(x_pos)
            y.append(y_pos)
            z.append(0)
            dx.append(0.4)
            dy.append(0.8)
            dz.append(corr)
            x_pos+=1
        y_pos+=5
        color_index+=1

    ax.set_yticks(range(0, len(y_labels) * 5, 5))
    ax.set_yticklabels(y_labels, fontsize=5, weight = 'bold', rotation = 90, color = '#233842')
    ax.bar3d(x, y, z, dx, dy, dz, color = colors3d)
```

Лістинг№17

models_set_accuracy_recall_graph.py

```
import matplotlib.pyplot as plt

def build_accuracy_recall_graph(every_combination_models_predictions_evaluated):
    accuracies = []
    recalls = []
    models_set_names = []
    accuracy_recall_averages = []
    for set_obj in every_combination_models_predictions_evaluated.values():
        accuracies.append(set_obj.accuracy)
        recalls.append(set_obj.recall)

        models_set_names.append(str(set_obj.classifiers_names))

        accuracy_recall_averages.append((set_obj.accuracy + set_obj.recall) / 2)
    font = {'family': 'serif',
            'color': '#233842',
            'weight': 'normal',
            'size': 12,
            }

    fig = plt.figure(figsize=(5, 3), dpi=100)
    ax = fig.add_subplot(111)
    ax.plot(models_set_names, accuracies, label = 'Точність')
    ax.plot(models_set_names, recalls, label = 'Повнота')
    ax.plot(models_set_names, accuracy_recall_averages, label='Сумарний показник')
    ax.tick_params(axis='x', rotation=90, labelsize = 6)
    ax.set_xlabel('Комбінації моделей', fontdict=font)
    ax.set_ylabel('Точність та повнота', fontdict=font)
    ax.set_title('Точність та повнота кожного набору моделей', fontdict=font)
    ax.text(19, 0.98, f'Найкращий набір:\n{models_set_names[accuracy_recall_averages.index(max(accuracy_recall_averages))]}'
```

```
f'\nТочність:{accuracies[accuracy_recall_average
s.index(max(accuracy_recall_averages))]},
Повнота:
{recalls[accuracy_recall_averages.index(max(accu
racy_recall_averages))]}'
            , style='italic',fontsize = 8,
            bbox={'facecolor':  'red',  'alpha':
0.5, 'pad': 5})
    ax.legend()
    ax.grid()
```

Кваліфікаційна робота бакалавра

Виконавець: студент 4 курсу, група К-17-1 М.М. Стебелецький

Керівник: к.т.н, доцент кафедри КНІТ Е.А. Манзюк

Тема

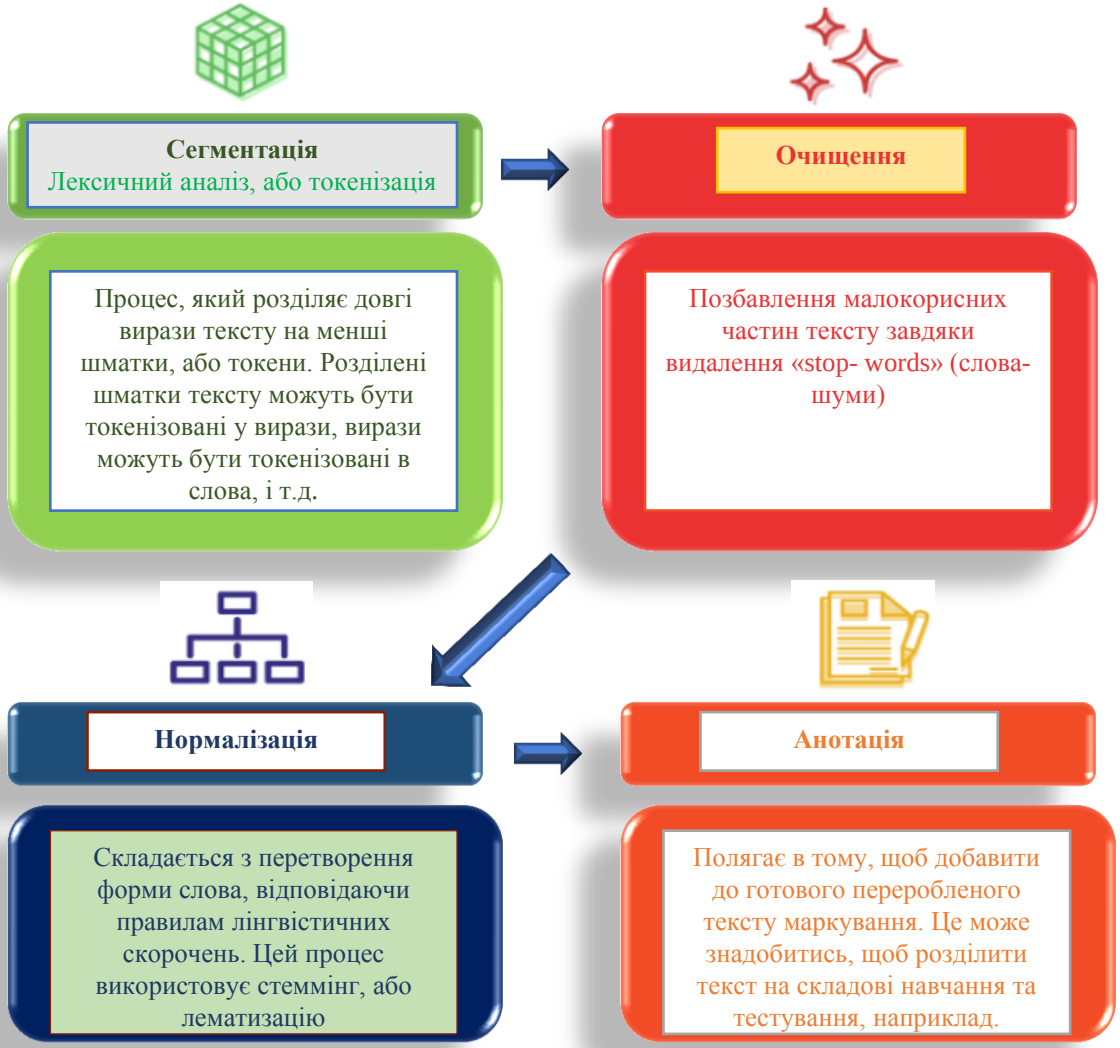
Застосування агрегативних підходів для класифікації на базі ансамблевих моделей

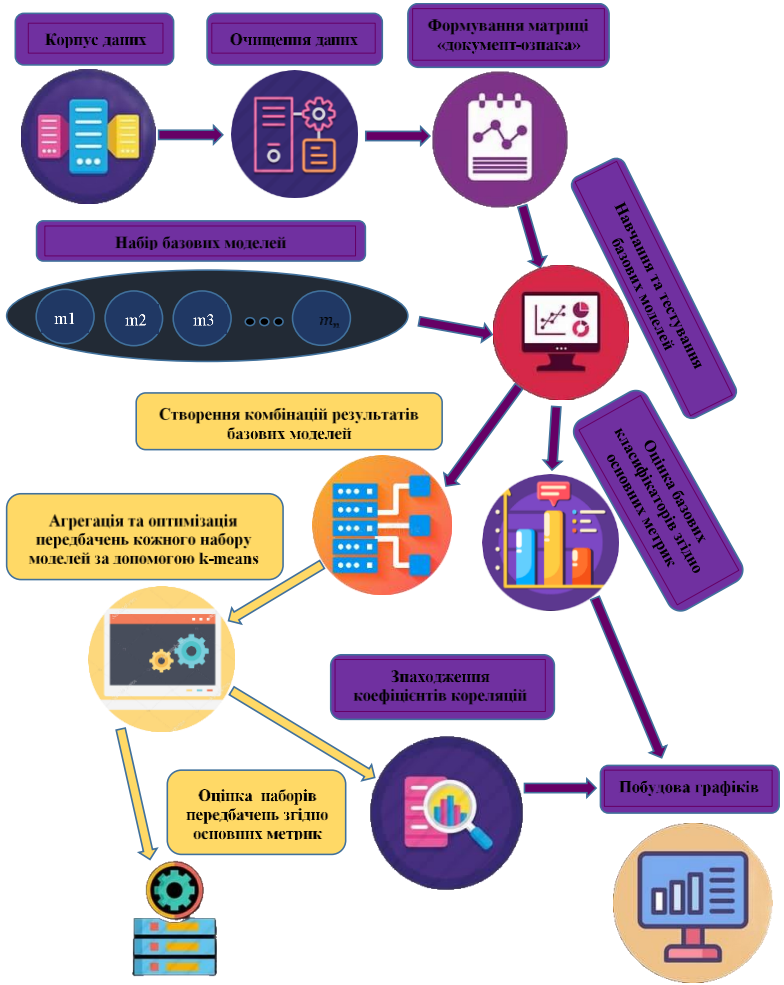
Актуальність

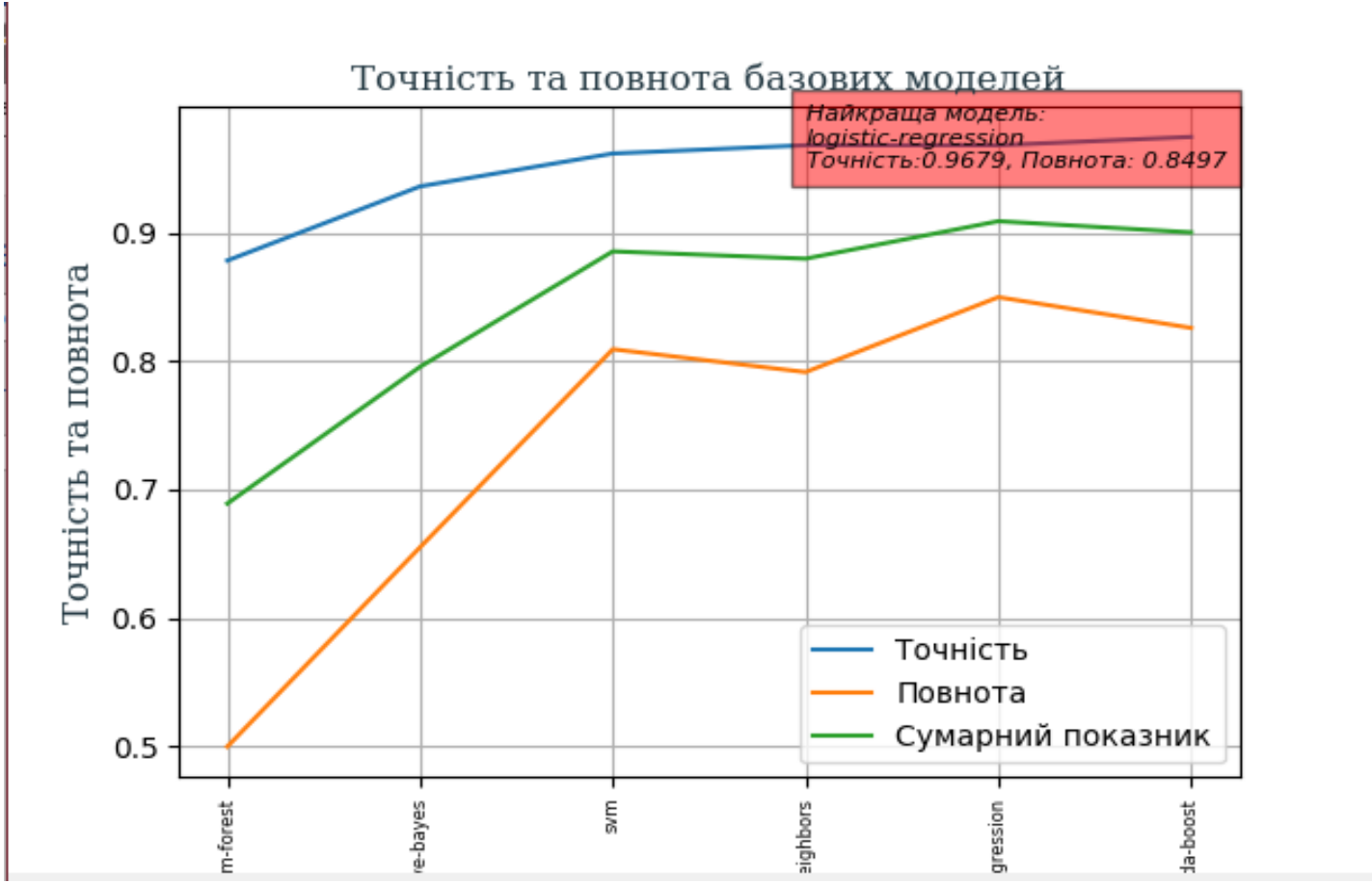
Мета будь-якого завдання машинного навчання - знайти єдину модель, яка найкращим чином пророкує бажаний результат. Замість того, щоб створювати одну модель і сподіватися, що ця модель буде кращим / найточнішим провісником, який ми можемо зробити, методи ансамблю беруть до уваги безліч моделей і усереднюють їх для отримання однієї остаточної мета-моделі.

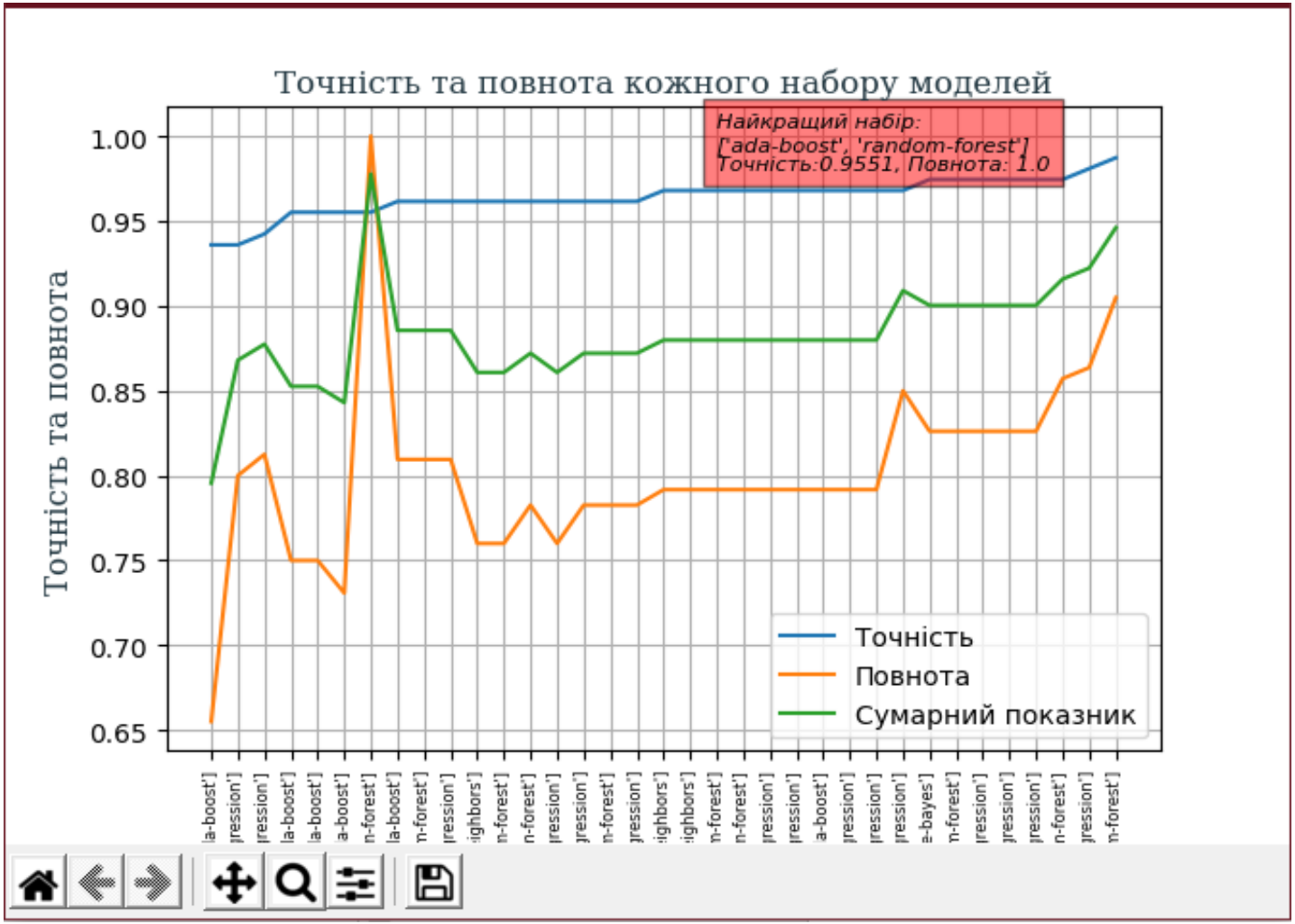
Дослідження **застосування агрегативних підходів** має наступні цілі:

1. Покращити **результативність** класифікаційної системи, зокрема, покращенню підлягають такі метрики системи, як **точність та повнота**.
2. Розробити підхід **агрегатування** базових алгоритмів класифікації та подальшого **усереднення** їхніх результатів.
3. Реалізувати інформативну **систему візуалізації** даних за допомогою **двох та трьох-вимірного** простору. За допомогою візуалізації проглядаються всі **необхідні закономірності**.
4. **Провести аналіз**:
   - сучасних підходів на базі ансамблів;
   - порівняння ефективності застосуваннявідомих методів щодо предметної області;
   - ефективності запропонованих рішень шляхом експериментальних досліджень на відомих корпусах даних.

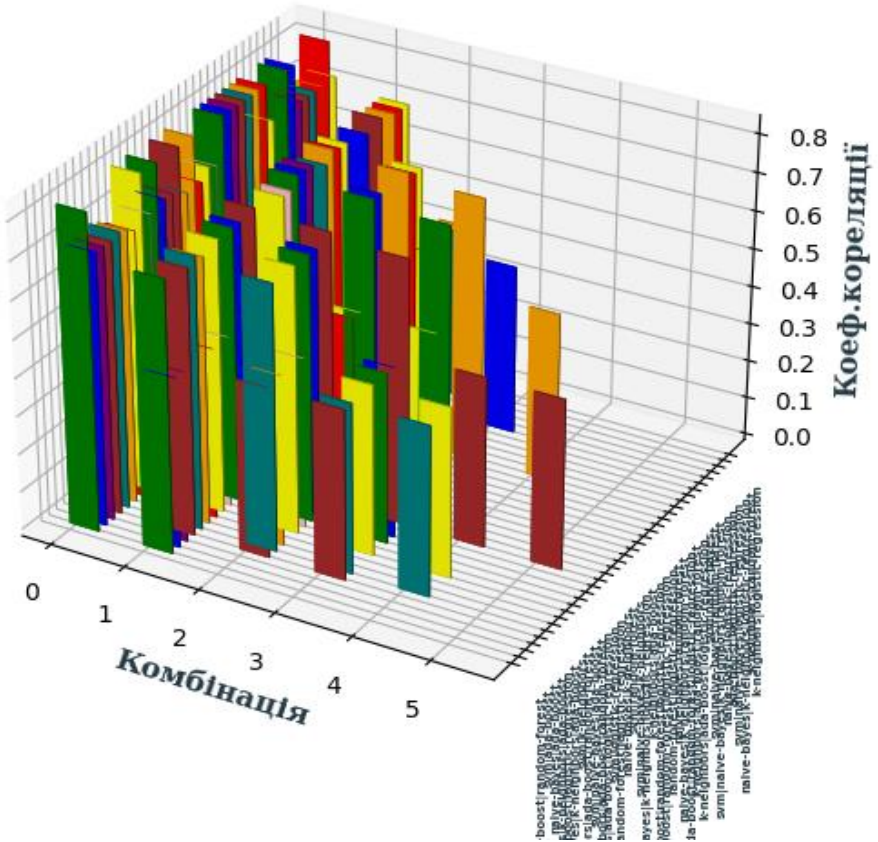

Сегментація
Лексичний аналіз, або токенізація
Процес, який розділяє довгі вирази тексту на менші шматки, або токени. Розділені шматки тексту можуть бути токенізовані у вирази, вирази можуть бути токенізовані в слова, і т.д.
Очищення
Позбавлення малокорисних частин тексту завдяки видалення «stop- words» (слова-шуми)
Нормалізація
Складається з перетворення форми слова, відповідаючи правилам лінгвістичних скорочень. Цей процес використовує стеммінг, або лематизацію
Анотація
Полягає в тому, щоб добавити до готового переробленого тексту маркування. Це може знадобитись, щоб розділити текст на складові навчання та тестування, наприклад.


Корпус даних
Очищення даних
Формування матриці «документ-ознака»
Набір базових моделей
m1
m2
m3
m_n
Навчання та тестування базових моделей
Створення комбінацій результатів базових моделей
Оцінка базових класифікаторів згідно основних метрик
Агрегація та оптимізація передбачень кожного набору моделей за допомогою k-means
Знаходження коефіцієнтів кореляцій
Оцінка наборів передбачень згідно основних метрик
Побудова графіків

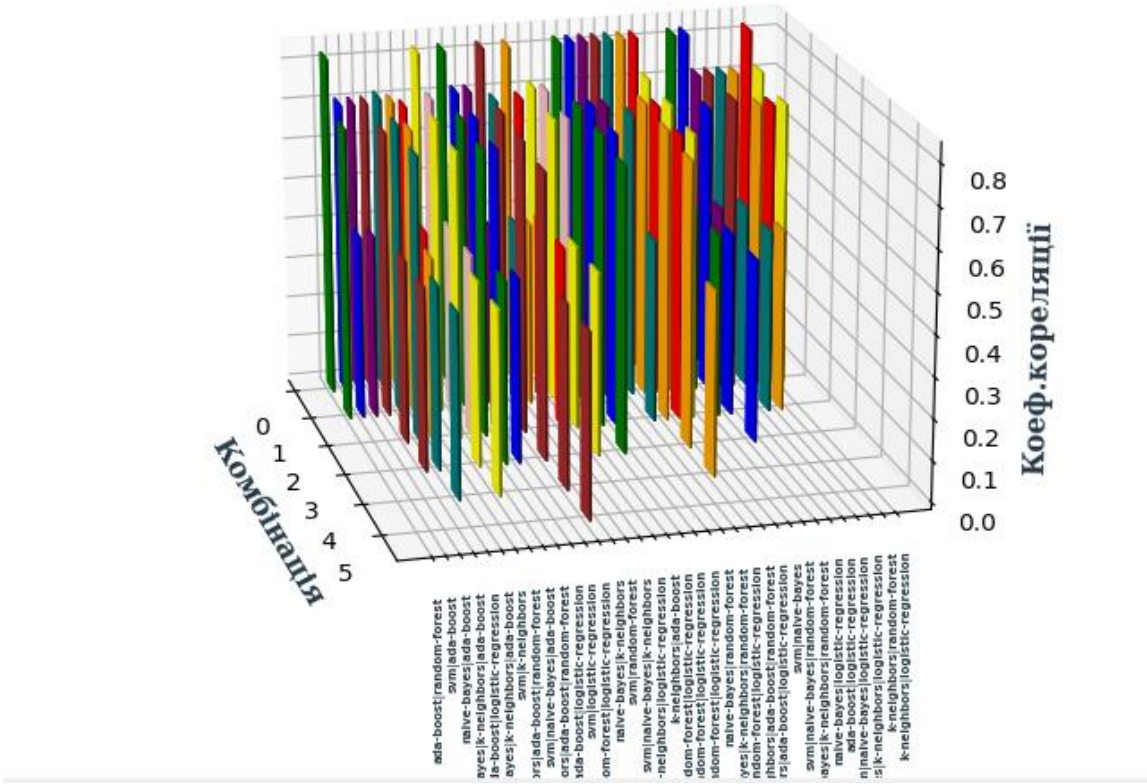

Точність та повнота базових моделей
Найкраща модель:
logistic-regression
Точність:0.9679, Повнота: 0.8497
Точність та повнота
0.9
0.8
0.7
0.6
0.5
Точність
Повнота
Сумарний показник
svm
gression


Точність та повнота кожного набору моделей
Найкращий набір:
['ada-boost', 'random-forest']
Точність:0.9551, Повнота: 1.0
Точність та повнота
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Точність
Повнота
Сумарний показник

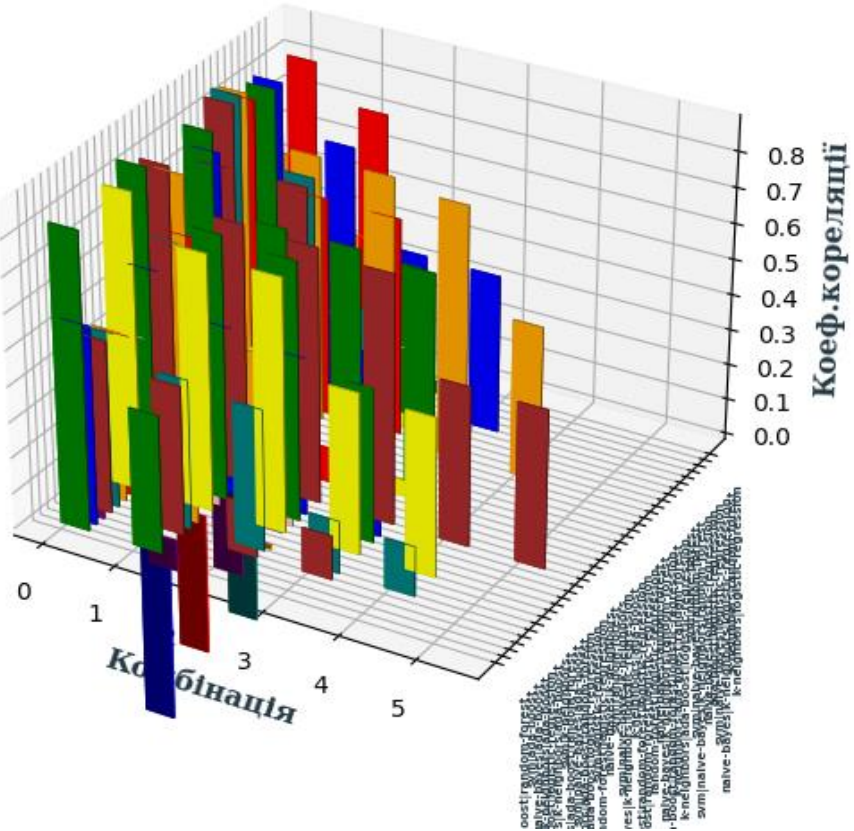
Коеф. кореляції між кожною моделлю в наборі і оригіналом



Коеф. кореляції між кожною моделлю в наборі і оригіналом



Коеф. кореляції між кожною моделлю в наборі і оригіналом



Шляхи вдосконалення

Одним із шляхів вдосконалення реалізованої системи класифікації є підбір базових класифікаторів із множини на основі вирахуваних коефіцієнтів кореляцій між передбаченнями класифікаторів і правильними відповідями. Такий підхід допоможе в подальшому ще більше оптимізувати ансамблеву модель, оскільки ансамбль буде складатись із тих моделей, які показують найвищі метрики точності.

Дякую за увагу

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

Направляється студент Стебелецький М. М. на захист дипломного проекту (роботи)
(прізвище, ініціали)

за спеціальністю 122 - Комп'ютерні науки

На тему: Застосування агрегативних підходів для класифікації на базі ансамблевих моделей

Дипломний проект (робота), рецензія і довідка про перевірку на плагіат додаються.

Декан факультету _____ САВЕНКО О.С.
(підпис) (прізвище та ініціали)

ДОВІДКА УСПІШНОСТІ

Стебелецький М. М. за період навчання на факультеті програмування та компютерних і телекомунікаційних систем з 2017 по 2021 роки. повністю виконав навчальний план спеціальності з такими розподілом оцінок за:
національною шкалою: відмінно 93,75 %, добре 6,25 %, задовільно 0,00 %.
шкалою ЄКТС: A 90,91 %, B 3,64 %, C 3,64 %, D 0,00 %, E 1,82 %.

Методист факультету _____ _____
(підпис) (прізвище та ініціали)

ВИСНОВОК КЕРІВНИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ) ТА ОБГРУНТУВАННЯ ОЦІНКИ

Студент Стебелецький М.М. успішно завершив розробку та впровадження програмної системи класифікаційної роботи. Продемонстрував при цьому належні уміння та вміння з предметної області. Робота є завершеною в повній мірі

Оцінка дипломного проекту (роботи) відмінно

Керівник дипломного прокту (роботи) _____ Мандзюк Є.А
(підпис) (прізвище та ініціали)

"8" 06 2021 р.

ВИСНОВОК КАФЕДРИ ПРО ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Дипломний проект (роботу) розглянуто. Студент Стебелецький М. М. допускається до захисту цього

Завідувач кафедри КНІТ
(назва)

_____ "8" 06 2021 р.
(підпис, прізвище, ініціали)

Tue Jun 08 13:28:06 EEST 2021, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 24.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 10%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 92707 Название: Застосування агрегативних підходів для класифікації на базі ансамблевих моделей Добавлено в БД: 2021-06-08 Авторы: М.М.Стебелецький Руководители: Е.А. Манзюк Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	67932	638	19478 (29%)	196 (31%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
90163	Название: ЗВІТ з професійної практики Добавлено в БД: 2021-05-11 Авторы: Стебелецький М.М Руководители: Скрипник Т.К. Консультанты: Опоненты:	16070 (24.0%)	157 (25.0%)

UNICHECK

Ім'я користувача:
Кафедра КН

ID перевірки:
1008229327

Дата перевірки:
08.06.2021 14:21:23 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2021 14:22:41 EEST

ID користувача:
100005671

Назва документа: **На плагіат Кваліфікаційна робота Стебелецький**

Кількість сторінок: **69** Кількість слів: **10738** Кількість символів: **80076** Розмір файлу: **3.01 MB** ID файлу: **1008302844**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

13.2%
Схожість

Найбільша схожість: **2.37%** з Інтернет-джерелом (https://ppt-online.org/538966)

12% Джерела з Інтернету	293	Сторінка 71
4.21% Джерела з Бібліотеки	61	Сторінка 75

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	14
Підозріле форматування	19 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *Застосування агрегативних підходів для класифікації на базі ансамблевих моделей*

Автор: *студент групи КН-17-1 Стебелецький Мирослав Миронович*

Спеціальність: *122 – Комп'ютерні науки*

Освітня програма: *освітньо-професійна*

Науковий керівник: *доцент кафедри КНІТ Манзюк Е.А.*

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	*відповідає*
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;

2) усі запозичення фрагментарні;

3) до запозичень входять фрагменти програмного коду, що на мають авторства і містять поширені конструкції;

4) серед запозичень знаходяться загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 13,2 % і адресується до першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Е.А.Манзюк

Гарант ОП _____ О. В. Мазурець

Завідувач кафедри КНІТ _____ О. В. Бармак

Хмельницький національний університет

Кафедра комп'ютерних наук та інформаційних технологій

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра

студента *групи КН-17-1 Стебелецького Мирослава Мироновича*

за темою *«Застосування агрегативних підходів для класифікації на базі ансамблевих моделей»*

1. Актуальність і значення теми: *дослідження застосування агрегативних підходів допоможе покращити результативність класифікаційної системи, зокрема, покращенню підлягають такі метрики системи, як точність та повнота.*

2. Оцінка запропонованих моделей, підходів, алгоритмів, інформаційної складової та засобів розробки: *підхід агрегатування базових алгоритмів класифікації та подальшого усереднення їхніх результатів виявився доцільним, оскільки в результаті дослідження було доведено, що ансамбль базових моделей класифікації показує вищі результати згідно основних метрик. Запропонована система візуалізації даних інформативна, за допомогою двох та трьох-вимірного простору проглядаються всі необхідні закономірності.*

3. Оцінка розробленої інформаційної системи, її практична цінність та економічна доцільність: *розроблена інформаційна система, а також дослідження, проведене під час проектування системи покликане вирішити актуальні проблеми оптимізації моделей агрегатування.*

4. Загальний висновок: *вимоги поставленої задачі виконані в повному обсязі, система класифікації на базі ансамблевих моделей працює вірно.*

Робота заслуговує на оцінку «відмінно»

Рецензент к. ф.-м. н., доц. Ярецька Н.О. 08.06.2024