МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
Інститут інформаційних технологій в економіці

Кафедра математичного моделювання та статистики

Освітньо-професійна програма	**Економічна кібернетика та Дата Сайнс**
Галузь знань	**05 соціальні та поведінкові науки**
Спеціальність	**051 економіка**

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему **«Аналіз медіа контенту методами машинного та глибокого навчання»**
(назва теми)

здобувача Завальського Андрія Анатолійовича
(ПІБ, підпис)

Науковий керівник: кандидат економічних наук,
доцент Кмитюк Т. Л.
(науковий ступінь, учене звання, ПІБ)

(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри кандидат фізико-математичних наук,
професор Великоіваненко Г.І.

(підпис)

Київ 2023

ЗМІСТ

ВСТУП

За останні десятиліття засоби інформаційних технологій практично повністю заполонили усі сфери життя людини: переважна більшість продажів здійснюється в інтернеті через замовлення на вебсторінках, люди спілкуються та створюють спільноти в соціальних мережах, реклама та маркетинг використовують банерну та контекстну рекламу на сторінках електронних видань, частка автоматизованого виробництва з залученням найновітніших сучасних технологій автоматизації з кожним роком все більше та більше зростає. Все це є неминучим наслідком стрімкого розвитку технологій в галузях будування техніки, обробки інформації, штучного інтелекту, тощо. Розвиток різноманітних технологій автоматизації та обробки інформації зрозумілий з очевидних причин, він забезпечує значну економію часу на рутинних справах людини, яка користується його прогресом, дозволяє оптимізувати та зробити більш ефективною роботу найманих працівників, дає змогу бізнесу швидше задовольняти потреби своїх споживачів, збирати, накопичувати та зручно аналізувати результати своєї діяльності.

Однією з найважливіших сфер застосування сучасних ІТ рішень є засоби масової інформації (ЗМІ), саме через статті та заголовки представників медіа індустрії люди мають змогу дізнаватися доступну, корисну, своєчасну та необхідну для специфіки їхньої діяльності інформацію. Як відомо, саме ЗМІ формулюють денну повістку та визначають коли, як та на що саме буде спрямована увага населення. На основі проведених досліджень, розслідувань, інтерв’ю представники глобальних медіа можуть легко впливати на діяльність та результати роботи найвідоміших та найпотужніших підприємств, визначати долю та розвиток кар’єри політичних діячів, підприємців, діячів культури тощо. Історія знає багато прикладів, коли випуски новин чи опубліковані в новинах дослідження, що стосувалися окремих суб’єктів економічної діяльності, викликали значні флуктуації в настроях та попиті споживачів такого підприємства, в умовах

сучасних вільних стокових ринків такі різкі стрибки особливо помітні: різкий зріст або спад ціни акцій компаній, залучення нових або навпаки скорочення вже існуючих акціонерів фірм, зростання або, навпаки, спад привабливості фірми для потенційних інвесторів, тощо, – аби викликати описані реакції достатньо лише одного випуску в масовому або локальному виданні, стосовно того чи іншого аспекту діяльності окремо взятої форми/організації.

Оскільки кількість інформації в просторі інтернету зростає з неймовірною швидкістю, звичні методи обробки такої інформації, як читання, перевірка достовірності описаних матеріалів вже не можуть демонструвати бажаного результату, досить часто на основі цих ресурсів необхідно приймати швидкі та інколи навіть доленосні рішення, тому сучасні підходи до обробки навісної інформації можуть заощадити значну кількість коштів, ресурсів та часу тим, хто ними користується.

Метою кваліфікаційної роботи є апробація методів та алгоритмів машинного та глибокого навчання для прискореного аналізу, обробки та узагальнення україномовного медіа контенту.

Для досягнення зазначеної мети необхідно вирішити такі завдання:

1) визначити сутність поняття контент-аналізу та основні методи його проведення;

2) проаналізувати стан та умови розвитку автоматичного збору даних з вебресурсів;

3) дослідити сучасні методи переведення текстової інформації в зручну для машинного навчання форму;

4) створення візуального користувацького інтерфейсу (UI додатку);

5) здійснити практичну реалізацію функціоналу UI додатку;

Об’єктом дослідження даної роботи є україномовний медіа контент.

Предметом дослідження роботи є методи машинного та глибокого навчання, які застосовуються для виявлення, аналізу та коригування інформації.

Методами дослідження даної роботи є підходи екстракції інформації з вебмедіа ресурсів (вебскрейпінг), обробка природної мови, машинне навчання та прикладне використання.

Теоретична, методична та практична значущість даної роботи розкривається в розгляді та описі новітніх, на сьогодні не широко розповсюджених в україномовному просторі підходів до обробки природної мови, алгоритмів нейронних мереж, що застосовуються як для створення нових так і класифікації існуючих слів в тексті, а також додатку з користувацьким інтерфейсом для прискореного аналізу текстових ресурсів, ідею та ресурсний код якого можна використати для імплементації або вдосконалення реальних програмних продуктів.

Інформаційною базою дослідження даної роботи є, здебільшого, новітні публікації в науково-популярних журналах та виданнях, в яких публікуються останні дослідження та досягнення в областях комп'ютерних наук, науки про дані, штучного інтелекту, тощо. Для опису теоретичних основ деяких алгоритмів та принципів роботи бібліотек, що застосовувалися для реалізації практичної частини, використовувалися сучасні західні джерела.

Структура роботи складається з вступу, двох розділів, що описують теоретичну основу та застосування описаних підходів на практиці відповідно, висновку та списку використаних джерел.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ І МЕТОДИ АНАЛІЗУ МЕДІА КОНТЕНТУ

1.1. Сутність поняття контент-аналізу

Контент-аналіз – якісно-кількісний метод вивчення документів, який характеризується об'єктивністю висновків і строгістю процедури та полягає у квантифікаційній обробці тексту з подальшою інтерпретацією результатів. Предметом контент-аналізу можуть бути як проблеми соціальної дійсності, котрі висловлюються чи навпаки приховуються у документах, так і внутрішні закономірності самого об'єкта дослідження [1].

Класично виділяється два основних типи контент-аналізу: кількісний і якісний. Кількісний аналіз – тип аналізу контенту, що передбачає систематичний і об'єктивний підрахунок і категоризацію змісту певної форми тексту. Отримані дані потім піддаються статистичному аналізу для виявлення закономірностей, тенденцій і зв’язків в тексті чи окремому абзаці. Кількісний контент-аналіз часто використовується для вивчення медіа-контенту, реклами та політичних виступів. Якісний аналіз – тип аналізу контенту, пов'язаний з інтерпретацією та розумінням сенсу та контексту контенту, включає в себе систематичний аналіз змісту для виявлення тем, шаблонів та інших відповідних особливостей, а також для інтерпретації основних значень та наслідків цих ознак. Якісний контент-аналіз часто використовується для вивчення інтерв'ю, фокус-груп та інших форм якісних даних, де дослідник зацікавлений у розумінні суб'єктивного досвіду та сприйняття учасників.

Виділяють декілька етапів проведення кількісного контент-аналізу:

1. Визначення питання або мети дослідження.

Перш ніж почати контент-аналіз, аналітик повинен чітко визначити своє дослідницьке питання або мету. Це необхідно для визначення контенту, який потрібно проаналізувати, і тип аналізу, який потрібно провести.

2. Формування вибірки.

Необхідно вибрати репрезентативний зразок вмісту, який необхідно проаналізувати. Це може включати формування випадкової вибірки або цільової вибірки, залежно від питання дослідження та доступності вмісту.

3. Розробка схеми прийняття рішень.

Пункт включає розробка схеми оцінки вибірки та поділу тексту на набір категорій вмісту, забезпечення чіткого розуміння цих категорій та взаємозалежних елементів в них. На цьому етапі можлива ручна категоризація або використання різноманітних інструментів програмного забезпечення, методів кластеризації, категоризації.

4. Аналіз даних та інтерпретація.

Після формування каркасу оцінки – аналізувати дані за допомогою відповідних статистичних або якісних методів, залежно від питання дослідження і типу даних. Інтерпретувати результати аналізу в контексті дослідження питання або мети.

5. Висновки.

Формування висновків про свої результати в чіткій та стислій формі, розкриваючи питання дослідження, методологію, результати та висновки.

Для проведення якісного контент-аналізу, в свою чергу, виділяють наступні підходи: визначення об’єкта впливу, аналіз близькості та когнітивне відображення.

1. Визначення об’єкта впливу.

Цей тип реляційного аналізу передбачає оцінку різних емоційних забарвлень, наявних в конкретному тексті. Хоча розуміння об’єкта впливу може бути безцінним, проведення його може виявитися важким залежно від тексту. Наприклад, текст може описувати емоційний стан людей в різний період часу, в різних місцях і за різних обставин, що унеможливлює формування остаточної характеристики тексту щодо певного об’єкту.

2. Аналіз близькості.

Відносно простіший аналітичний підхід, ніж визначення впливу на об’єкт, аналіз близькості оцінює спільне виникнення явних понять у тексті. Цей метод передбачає прикладний аналіз тексту і створення, так званої, матриці понять, яка являє собою групу взаємопов'язаних спільних концепцій. Концептуальні матриці допомагають оцінити і визначити загальне значення тексту або ідентифікацію вторинного повідомлення або теми. Сучасним підходом до оцінки такої спільності також токенізовані є вектори слів.

3. Когнітивне відображення.

Когнітивне відображення може використовуватися як спосіб візуалізації результатів впливу на об’єкт, або аналізу близькості. Цей метод використовує результати аналізу екстракції або близькості, щоб створити графічну карту, що ілюструє зв'язок між емоціями або концепціями.

Контент-аналіз має кілька ключових характеристик, до яких відносяться:

1. Об'єктивність.

Контент-аналіз спрямований на те, щоб бути об'єктивним методом дослідження, а це означає, що дослідник не вносить власні упередження або інтерпретації в аналіз. Це досягається за допомогою стандартизованих і систематичних процедур категоризації.

2. Системність.

Передбачає використання системного підходу для аналізу та інтерпретації змісту спілкування. Це передбачає визначення питання дослідження, вибір вибірки контенту для аналізу, розробку схеми кодування та аналіз даних.

3. Кількісні оцінки.

Аналіз тексту або контенту часто включає підрахунок і вимірювання виникнення конкретних тем, показників, організацій, осіб у змісті, що робить його кількісним методом дослідження. Це дозволяє проводити статистичний аналіз та узагальнення результатів.

4. Прив’язка до контексту.

Контент-аналіз розглядає контекст, в якому відбувається спілкування, такий як період часу, аудиторія та мета спілкування тощо.

5. Послідовність.

Контент-аналіз – це ітераційний процес, це означає, що дослідник може уточнити схему кодування та аналіз, як вони аналізують дані, щоб гарантувати, що результати є правильними та надійними.

6. Надійність та валідність.

Контент-аналіз має на меті бути надійним та обґрунтованим методом дослідження, що означає забезпечення послідовність та точність результатів. Це досягається за допомогою міжкатегоріальних тестів надійності та інших заходів для забезпечення якості даних та аналізу.

1.2. Організація вебсторінок в просторі інтернет

Однією з найважливіших речей при роботі з медіа контентом в Інтернеті є розуміння роботи браузерів та організація вебсторінок в просторі інтернет.

Всесвітня павутина (веб/web/www) – це глобальна система взаємопов'язаних гіпертекстових документів, зображень, відео тощо, доступ до яких здійснюється через інтернет. Переважна більшість вебсторінок написані на мові розмітки гіпертексту (HTML), яка визначає, як елементи на сторінці відображаються в браузері. Структуру всесвітньої павутини складають мільйони вебсерверів, що являють собою програми, розміщені на комп’ютерах, розташованих по всьому світу, та слугують для відправки та обробки HTTP/HTTPS запитів, найпопулярнішими з них є Apache, nginx, Cloudflare та Google.

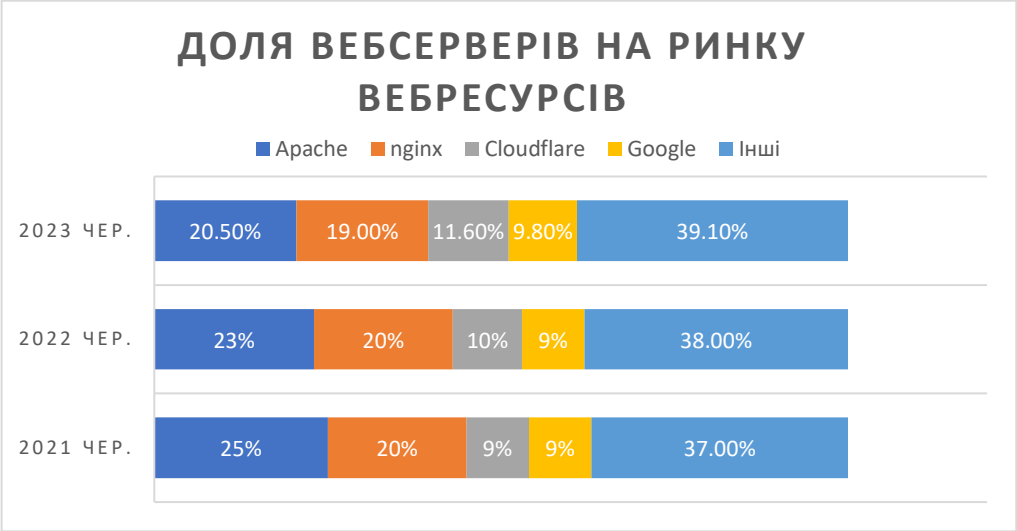

Рисунок 1.1 – Доля вебсерверів на ринку вебресурсів

Джерело: розроблено автором на основі [2].

Виходячи з інформації на графіку бачимо (рис.1.1), що сервери згаданих компаній стабільно займають близько 60% ринку, що є значною часткою, враховуючи, що створити власний серверний простір може абсолютно будь-хто, хто забажає.

Веббраузер – це прикладне програмне забезпечення, яке створене для перегляду вебсторінок і, відповідно, виступає клієнтом в інтернет-просторі. Найрозповсюдженішими браузерами станом на сьогодні є Google Chrome, частка якого на ринку пошукових сервісів становить 63%, друге місце займає Safari – 20%, далі Edge (6%), Firefox (3%), Opera (3%) та інші (рис. 1.2).

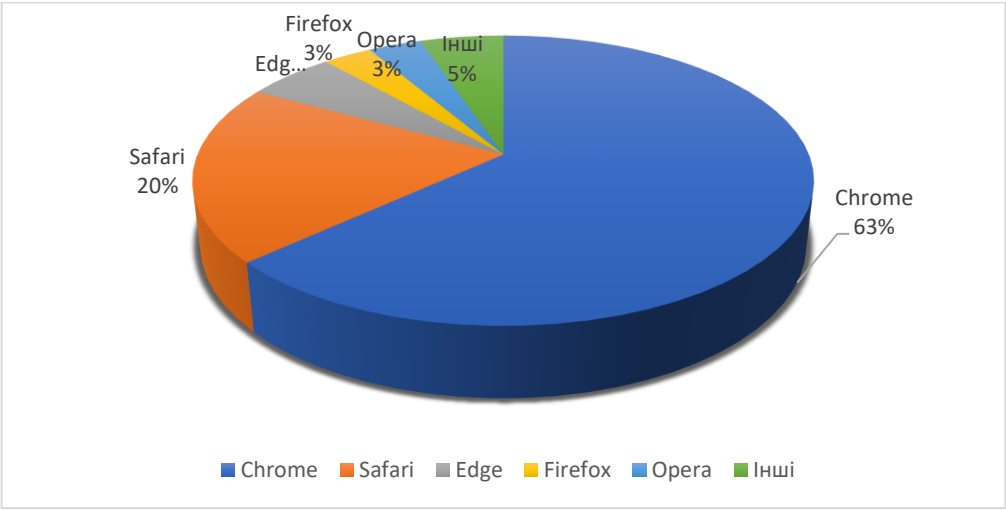

Рисунок 1.2 – Частка браузерів на ринку пошукових сервісів

Джерело: розроблено автором на основі [3] за 01.10.2023.

Як зазначалося, ресурсами в вебпросторі є HTML-файли, зображення, відео, аудіо та інші ресурси, необхідні клієнтам, оскільки переважна більшість текстового контенту відображається в браузерах як HTML, зазвичай в цьому форматі і зберігається контент, що публікується медіа виданнями, журналами тощо. HTML має деревовидну структуру, яка називається DOM (Document Object Model), це програмний інтерфейс, який дозволяє програмам і скриптам отримувати доступ до вмісту HTML, XHTML і XML-документів, а також змінювати вміст, структуру та оформлення таких документів. Веббраузери використовують DOM для відображення HTML-документів. Коли браузер завантажує HTML-документ, він створює в пам'яті представлення документа у вигляді дерева вузлів. Кожен вузол у дереві DOM представляє собою елемент HTML, атрибут, текст або інші дані, які містяться в документі (рис. 1.3).

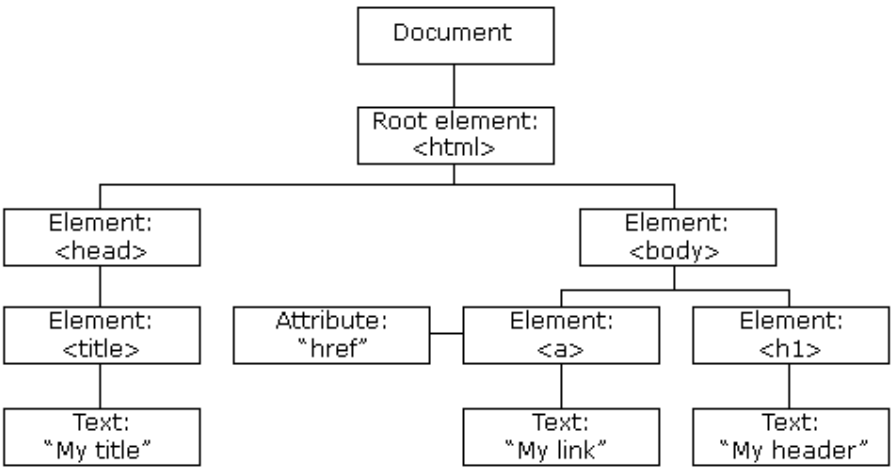

Рисунок 1.3 – Структура дерева мови гіпертексту HTML

Джерело: [4].

Завдяки структурі, при якій кожен елемент «огорнутий» в своєрідний контейнер, ми маємо змогу отримувати доступ до кожного такого елемента та отримувати інформацію з кожного тега вебсторінки.

Засобом комунікації клієнта (споживача, що ініціює підключення та робить запит до сервера) та сервера (надавача послуг, що очікує підключення, отримує запити, обробляє їх та видає відповідний результат) є мережевий протокол, що має назву HyperText Transfer Protocol (HTTP), або його більш безпечне розширення

HyperText Transfer Protocol Secure (HTTPS), що використовує додаткове шифрування при передачі інформації між портами. Для ідентифікації в інтернеті протоколом HTTP використовується Uniform Resource Identifier (URI), кожен запит клієнта до сервера не зберігає інформацію про попередні запити і є незалежнім від інших.

Оскільки протокол HTTP забезпечує комунікацію клієнт-сервер, він має вісім основних методів для реалізації обміну даними:

1. GET – використовується для отримання даних/представлення вказаного ресурсу.
2. POST – використовується для відправлення та розміщення даних, створення нового ресурсу або додавання даних до існуючого ресурсу.
3. PUT – використовується для оновлення існуючого ресурсу.
4. PATCH – використовується для часткового оновлення існуючого ресурсу.
5. DELETE – використовується для видалення існуючого ресурсу.
6. HEAD – метод є аналогом зазначеного методу GET, але в відповіді не повертається в тіло ресурсу.
7. OPTIONS – використовується для отримання інформації про можливості сервера або параметрів з'єднання для конкретного ресурсу.
8. TRACE – використовується для перевірки шляху, який проходить запит від клієнта до сервера.

Методи протоколу HTTP, зазвичай, характеризуються набором з п’яти параметрів, що допомагають зрозуміти основні можливості, функціонал, вхідні та вихідні параметри методу:

1. Наявність тіла запиту (Request has body) – визначає наявність тіла запиту при передачі клієнтом даних на сервер. Тіло запиту – один з можливих варіантів передачі даних протоколом, інколи метод може не мати тіла, в такому разі дані можуть передаватися іншим чином.

2. Наявність тіла при успішному результаті запиту (Successful response has body) – визначає наявність тіла результату запиту при передачі даних з серверу до клієнта.

3. Безпечність (Safe) – визначає, чи може виконання запиту окремим методом призвести до зміни даних на сервері.

4. Ідемпотентність (Idempotent) – вказує на властивість методу, яка говорить, що повторення одного і того ж самого запиту не призводить до змін стану сервера. Іншими словами, якщо запит є ідемпотентним, то кілька послідовних викликів цього запиту повинні мати той самий ефект, що і один виклик. Це важливо з точки зору надійності і безпеки мережевих операцій. Запити, які не є ідемпотентними, можуть призводити до непередбачуваних або небажаних наслідків, особливо в умовах помилок мережі чи несправності системи. З ідемпотентними запитами простіше працювати в умовах повторених викликів або відновлення стану операцій.

5. Можливе кешування (Cacheable) – вказує на те, чи може бути відповідь на конкретний запит збереженою (кешованою) для подальшого використання. Якщо відповідь вважається збереженою, то клієнти, проміжні проксі-сервери та інші учасники мережі можуть зберігати копію цієї відповіді та використовувати її для подальших запитів, замість того, щоб запитувати сервер, що економить ресурси сервера та пришвидшує роботу на боці клієнта.

Описані методи мають такі характеристики:

1. GET:
    1.1. Request has body: Ні, не має тіла у запиті. Тіло запиту розглядається небезпечним та ігнорується багатьма серверами.
    1.2. Successful response has body: Так, відповідь може містити тіло з даними ресурсу.
    1.3. Safe: Так, зазвичай вважається безпечним, оскільки не призводить до змін на сервері.
    1.4. Idempotent: Так, повторення запиту не змінює стану сервера.
    1.5. Cacheable: Так, може бути кешованим для підвищення ефективності.
2. POST:
    2.1. Request has body: Так, дані передаються через тіло запиту.

2.2. Successful response has body: Так, відповідь може містити тіло з даними створеного чи оновленого ресурсу.
2.3. Safe: Небезпечний, оскільки може призводити до змін на сервері.
2.4. Idempotent: Ні, повторення запиту може призводити до створення нового ресурсу.
2.5. Cacheable: Ні, зазвичай не кешується, оскільки може призводити до змін стану сервера.

3. PUT:
3.1. Request has body: Так, дані передаються через тіло запиту.
3.2. Successful response has body: Так, відповідь може містити тіло з даними оновленого чи створеного ресурсу.
3.3. Safe: Зазвичай вважається небезпечним, оскільки може призводити до змін на сервері.
3.4. Idempotent: Так, повторення запиту не змінює стану сервера.
3.5. Cacheable: Ні, зазвичай не кешується, оскільки може призводити до змін стану сервера.

4. DELETE:
4.1. Request has body: Зазвичай може бути пустим або містити ідентифікатор ресурсу.
4.2. Successful response has body: Зазвичай не має тіла у відповіді.
4.3. Safe: Зазвичай вважається небезпечним, оскільки призводить до видалення ресурсу.
4.4. Idempotent: Так, повторення запиту не змінює стану сервера.
4.5. Cacheable: Ні, зазвичай не кешується, оскільки призводить до змін стану сервера.

5. PATCH:
5.1. Request has body: Так, передає лише часткові дані для оновлення ресурсу.
5.2. Successful response has body: Так, відповідь може містити тіло з даними оновленого ресурсу.

5.3. Safe: Зазвичай вважається небезпечним, оскільки може призводити до змін на сервері.

5.4. Idempotent: Так, повторення запиту не змінює стану сервера.

5.5. Cacheable: Ні, зазвичай не кешується, оскільки призводить до змін стану сервера.

6. HEAD:

6.1. Request has body: Зазвичай не має тіла у запиті.

6.2. Successful response has body: Зазвичай не має тіла у відповіді.

6.3. Safe: Так, вважається безпечним, оскільки не призводить до змін на сервері.

6.4. Idempotent: Так, повторення запиту не змінює стану сервера.

6.5. Cacheable: Так, може бути кешованим для отримання метаданих ресурсу.

7. OPTIONS:

7.1. Request has body: Зазвичай не має тіла у запиті.

7.2. Successful response has body: Так, відповідь може містити інформацію про підтримувані методи та інші можливості сервера.

7.3. Safe: Так, вважається безпечним, оскільки не призводить до змін на сервері.

7.4. Idempotent: Так, повторення запиту не змінює стану сервера.

7.5. Cacheable: Так, може бути кешованим для отримання інформації про можливості сервера.

8. TRACE:

8.1. Request has body: Зазвичай не має тіла у запиті.

8.2. Successful response has body: Так, відповідь TRACE-запиту може містити тіло, яке містить копію тіла отриманого запиту.

8.3. Safe: Зазвичай вважається безпечним, оскільки не призводить до змін стану сервера. TRACE-запити дозволяють клієнтам відстежувати маршрут запиту і перевіряти, як він модифікувався під час проходження через різні сервери.

8.4. Idempotent: Так, повторення TRACE-запиту не повинно змінювати стан сервера.

8.5. Cacheable: Ні, зазвичай не кешується, оскільки призводить до змін стану сервера. TRACE-запити не підходять для кешування через їхній потенційний вплив на конфіденційність даних та безпеку сервера.

Ознайомившись з функціями протоколу HTTP, маємо змогу розглянути саму структуру запиту (request) та структуру результату запиту (response). Структура HTTP запиту є такою:

1. Рядок запиту (Request Line):
   1.1. Метод протоколу – вказує, яку операцію треба виконати, наприклад, GET, POST або будь-який з зазначених методів.
   1.2. URI (Uniform Resource Identifier) – це шлях/посилання/ідентифікатор ресурсу, на який клієнт надсилає запит.
   1.3. Версія протоколу – вказує версію HTTP, яка використовується.

   Приклад:

   GET /diploma/path HTTP/2
2. Заголовки (Headers):
   2.1. Заголовки надають додаткову інформацію про запит або клієнта. Деякі заголовки можуть включати:
   - Host (Доменне ім'я сервера)
   - User-Agent (Інформація про браузер або клієнт)
   - Content-Type (Тип даних в тілі запиту (якщо тіло присутнє))
   - Authorization (Інформація для авторизації на сервері)
   - та інші.

   Наприклад:

   Host: diploma.com

   User-Agent: Mozilla/5.0

   Content-Type: application/json
3. Тіло (Body):
   - За наявності даного елемента в методі може вміщувати в собі будь-які дані в залежності від формату (JSON, XML та ін.)

Наприклад:

```
{
    "Item1": "value1",
    "Item2": "value2"
}
```

Загальний вигляд HTTP-запиту може виглядати приблизно так:

GET /diploma/path HTTP/2

Host: diploma.com

User-Agent: Mozilla/5.0

Структура ж HTTP відповіді на запит має наступну структуру:

1. Рядок стану (Status Line):
   1.1. Версія протоколу: Відображає версію HTTP, яку використовує сервер.
   1.2. Код стану (Status Code): Цифровий код, що вказує на результат виконання запиту.
   1.3.Пояснення стану (Reason Phrase): Коротке пояснення коду стану.

   Наприклад:

   HTTP/2 200 OK
2. Заголовки (Headers):
   - Server – інформація про вебсервер, який відправляє відповідь.
   - Date – дата та час генерації відповіді.
   - Content-Type – вказує тип медіа-даних у тілі відповіді.
   - Content-Length – вказує довжину тіла відповіді у байтах.
   - Location – вказує нове місце для ресурсу в разі редиректу.
   - Set-Cookie – встановлює куку (локальні дані) на боці клієнта.
   - Cache-Control – вказує налаштування кешу для відповіді.
   - та інші.

   Наприклад:

   HTTP/2 200 OK

   Server: Apache

Date: Mon, 01 Nov 2023 12:00:00 GMT

Content-Type: text/html

Content-Length: 1234

3. Тіло (Body): Тіло містить фактичні дані, які сервер відправляє клієнту. В залежності від типу значень, що сервер відправляє, це можу бути HTML, JSON, XML тощо.

Наприклад (при відправці даних типу HTML):

```
<!DOCTYPE html>
<html>
    <head>
        <title>Diploma</title>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

Отже, загальний вигляд відповіді протоколу HTTP матиме такий вигляд:

HTTP/2 200 OK

Server: Apache

Date: Mon, 22 Nov 2023 12:00:00 GMT

Content-Type: text/html

Content-Length: 1234

```
<!DOCTYPE html>
<html>
    <head>
        <title>Example Page</title>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
```

```
</html>
```

Оскільки у відповіді на запит завжди має місце тризначний код (XXX) стану відповіді, кожному коду запиту присвоєне власне ім’я, що пояснює результат роботи/виконання конкретного методу. Усі коди поділено на декілька груп, кожна з яких визначається по першій цифрі статус-коду:

- 1xx (Інформаційні) – означає що результат має інформаційний характер та можна продовжувати роботу (рідкий у використанні).
- 2xx (Успішні) – повідомляє, що запит був успішно виконаний.
- 3xx (Перенаправлення) – повідомляє, що клієнту необхідно виконати додаткові дії для завершення запиту.
- 4xx (Помилки клієнта) – повідомляє, що запит не був виконаний успішно через те, що вчинені клієнтом дії не відповідають умовам сайту, введені дані не пройшли верифікацію.
- 5xx (Помилки сервера) – повідомляє, що запит не був виконаний успішно через внутрішню помилку на боці сервера.

1.3. Вебскрейпінг медіа контенту

Оскільки метою даної роботи є аналіз медіа контенту в глобальній мережі інтернет, інформацію про будову вебсторінок та принципи роботи запитів можна використати для спрощеного доступу, отримання та організації інформації. Технологія, при якій в збір інформації з вебсайтів здійснюється за посередництва додаткового програмного забезпечення або методів автоматичних запитів до сторінок в інтернеті називається вебскрейпінгом [5]. Дана техніка може застосовуватися в різних сферах, зазвичай її часто використовують для автоматичного аналізу вебсайтів для збору інформації про ціни на товари, рейтинги продуктів, новини або будь-яку іншу доступну інформацію. Вебскрейпінг може бути корисним для збору даних для досліджень, аналізу конкурентів, відстеження змін у вмісті вебсайтів та інших задач в режимі реального часу та не тільки, конкретніші приклади використання можна описати такими розділами:

1. Бізнес-аналітика.

Вебскрейпінг дозволяє підприємствам отримувати велику кількість даних з вебсайтів конкурентів, аналізувати ціни на товари, слідкувати за рекламними стратегіями та здійснювати стратегічний аналіз ринку.

2. Маркетинг.

У сучасному бізнес-середовищі важливо отримувати актуальну інформацію про стан та кон’юнктуру ринків, які час від часу стають більш флуктуативними. Вебскрейпінг допомагає збирати дані про попит, тенденції та поведінку споживачів.

3. Аналіз соціальних мереж (SMM).

Вебскрейпінг може бути використаний для відстеження соціальних мереж, аналізу публікацій, взаємодії користувачів та реакції на продукти чи послуги.

4. Прогнозування та наукові дослідження.

Дослідження великих обсягів даних з використанням вебскрейпінгу може допомогти в прогнозуванні тенденцій, вивченні патернів та здійсненні наукових досліджень.

Існує шість основних випадків використання для забору вебресурсів: контент скрейпінг, проведення досліджень, збір контактів, порівняння цін, моніторинг даних про погоду та виявлення змін на вебсайтах.

Збирання контенту – може бути проявом піратства, вилучення оригінального вмісту з законного вебсайту та публікація його на іншому вебсайті без знання чи дозволу власника оригінального контенту. Збирання контенту може мати форму узагальнення інформації, використання інформації з декількох джерел для створення нового відображення інформації, відомого також як інтеграція вебданих. Наприклад, користувачі можуть створювати нові агрегатори подій чи централізовані портали для пошуку, наприклад, роботи, використовуючи дані з інших вебсайтів.

Оглядаючи всі варіанти використання вебскрейпінгу бачимо (рис. 1.4), що «збирання контенту» (контент скрейпінг) є найпопулярнішою метою використанням забору вебресурсів серед користувачів, на нього припадає 38% компаній, наступною популярною метою є «проведення досліджень» з часткою 26% фірм та «збір контактів» та «порівняння цін» з частками 19% та 16% відповідно, загалом бачимо чітку сформовану мету отримання вебданих для реалізації цілей фірм.

Рисунок 1.4 – Доля використання вебскрейпінгу в різних сферах

Джерело: розроблено автором на основі [6].

Тенденції у скрейпінгу вебресурсів показують (рис.1.4), що найбільшим споживачем вебданих є індустрія електронної комерції із загальною ринковою часткою приблизно 50%. Рекрутингові фірми також є значними учасниками в індустрії забору вебресурсів, вони дістають дані з порталів з оголошеннями вакансій по всьому світу щоб отримувати інформацію щодо тисяч вакансій, які щодня публікуються, а також відомостей про кандидатів на роботу. Індустрії нерухомості та туризму є наступними важливими учасниками, оскільки вони спостерігають велику кількість активності ботів для збору інформації для порівнянь цін. Кілька інших галузей, таких як наукова сфера та навіть технологічна індустрія, використовують забір вебресурсів для різних цілей і тим самим сприяють зростаючій популярності скрейпінгу вебресурсів як інструмента. Тепер розглянемо випадки використання індустрії забору вебресурсів для цих провідних гравців (рис. 1.5).

Рисунок 1.5 – Сфери-споживачі послуг скрейпінгу

Джерело: розроблено автором на основі [7].

Існують інструменти, що реалізують методи скрейпінгу на комерційній основі. Ринок програмного забезпечення та засобів, що використовуються для вебскрейпінгу неухильно зростає та стрімко розвивається, за результатами центру з проведення статистичних досліджень Research Nester: ринок програмного

скрейпінгу вебресурсів очікує досягти розміру 16 мільярдів доларів США до кінця 2035 року, зростаючи за 16% річним темпом у період прогнозу, тобто з 2023 по 2035 рік. У 2022 році розмір розробки програмного забезпечення для скрейпінгу вебресурсів становив приблизно 4 мільярди доларів США. Цей ріст здебільшого спричинений зростанням електронної комерції. Прогнозується, що до 2023 року кількість цифрових покупців у світі становитиме близько 3 мільярдів. Це приблизно 32% населення світу. Таким чином, попит на програмне забезпечення з скрейпінгу також збільшиться. вебскрейпінг – це метод, який передбачається використовувати для регулярного збору даних про продукти з різних сайтів електронної комерції, таких як Amazon, eBay, Google Shopping та інші [8].

Незважаючи на існуючі засоби для реалізації контент-скрейпінгу, всі вони використовують однакові мови програмування для власної реалізації, за кількістю створених репозиторіїв на платформі GitHub найпопулярнішими мовами, що використовуються для реалізації отримання вебданих є (рис. 1.6):

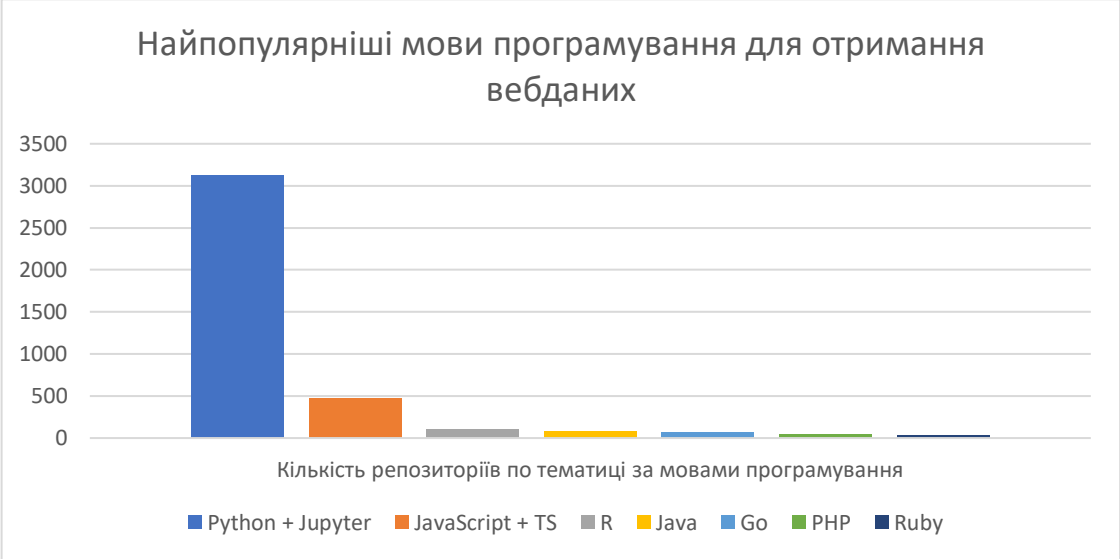

Рисунок 1.6 - Мови програмування для реалізації скрейпінгу

Джерело:розроблено автором на основі [9].

Насьогодні найпопулярнішою мовою, зі значним відривом, є саме Python, для контент-скрейпінгу за допомогою цієї мови використовуються:

1. Beautiful Soup та Requests – Beautiful Soup є потужною бібліотекою для парсингу HTML і XML-документів. Вона дозволяє легко отримувати дані з вебсторінок та реалізовувати навігацію по їхній структурі. Requests – це проста та ефективна

бібліотека для виконання HTTP-запитів. Вона дозволяє отримувати вміст вебсторінок для подальшого аналізу.

2. Scrapy – потужний фреймворк для вебскрейпінгу, який дозволяє створювати складні та ефективні хуки (spiders) для автоматизованого збору даних.
3. Selenium – фреймворк, що використовується для автоматизації веббраузера і може бути використаний для взаємодії з вебсторінками, якщо інші методи не ефективні.
4. LXML – бібліотека для обробки XML і HTML. Вона швидка та ефективна і має схожий інтерфейс з XPath для навігації та витягування даних.
5. Urllib – модуль, що входить до стандартної бібліотеки Python і дозволяє виконувати різні операції з URL, включаючи витягування вмісту вебсторінок.
6. PyQuery – бібліотека, яка надає можливість використовувати синтаксис jQuery для витягування і обробки HTML-документів.
7. MechanicalSoup – бібліотека, яка надає зручний інтерфейс для взаємодії з вебформами і отримання вмісту сторінок.

Найбільш вживаними та універсальними є бібліотеки Requests та BeautifulSoup, розглянемо їхній основний функціонал.

Основне завдання бібліотеки Requests – автоматизувати відправку запитів на сервер, що аналогічно ручній реалізації запитів клієнтом в браузері, для цього можна використовувати наступний функціонал CRUD-операцій (Create, Read, Update, Delete) (рис. 1.7).

```
import requests
url = 'https://api.diploma.com'
responseDataGet = requests.get(url)
responseDataPost = requests.post(url)
responseDataPut = requests.put(url)
responseDataDelete = requests.delete(url)
```

Рисунок 1.7 – Виконання стандартних CRUD операцій методами протоколу HTTP/HTTPS

Джерело: розроблено автором.

В даному випадку виконуються стандарті операції отримання, розміщення, оновлення, видалення даних за посиланням-ідентифікатором, результати запиту будуть розміщені у відповідних змінних responseDataGet, responseDataPost, responseDataPut, responseDataDelete.

Додатково до здійснюваних запитів засобами бібліотеки можливо додавати наступні параметри (рис. 1.8):

1. Params – словник або рядок, що містить параметри запиту.
2. Data – словник, рядок або файл, що містить дані, які необхідно передати у тілі запиту (якщо воно наявне в методі).
3. Json – об'єкт, який автоматично перетвориться в JSON і буде відправлений у тілі запиту (якщо воно наявне в методі).
4. Headers – словник із заголовками HTTP.
5. Auth – кортеж із даними аутентифікації (наприклад, auth=('username', 'password')).

```
import requests
url = 'https://api.diploma.com/data'
queryParams =   {
                'param1': 'value1',
                'param2': 'value2'
                }
queryHeaders = {'User-Agent': 'my-app/2.0'}

responseDataGet = requests.get(url, params = queryParams, headers = queryHeaders)
responseDataPost = requests.post(url, data=None, json=None)
responseDataPut = requests.put(url, data=None)
responseDataDelete = requests.delete(url)
```

Рисунок 1.8 – Приклад виконання CRUD операцій з наявними параметрами в запиті

Джерело: розроблено автором.

Результати виконання запиту також можна представити у необхідній користувачу формі шляхом виклику результатів відповідних методів:

response.text – текстове представлення відповіді.

response.json() – повернення JSON-представлення відповіді (якщо відповідь містить JSON).

response.status_code – код стану HTTP-відповіді.

BeautifulSoup – це бібліотека, що призначена для зручного парсингу HTML і XML даних у Python, вона дозволяє здійснювати навігацію по дереву розмітки та взаємодіяти з різними елементами документа. Функціонал бібліотеки BeautifulSoup в основному використовується для вебскрейпінгу та аналізу HTML-коду, щоб отримати дані вебсторінки додатково необхідно використовувати бібліотеку Requests (рис. 1.9).

```
import requests
from bs4 import BeautifulSoup

# отримання даних HTML сторінки через запит
url = 'https://api.diploma.com'
html_content = requests.get(url)

# Парсинг HTML-коду
soup = BeautifulSoup(html_content, 'html.parser')

# Знаходження всіх елементів тегу <a>
links = soup.find_all('a')

# Знаходження всіх елементів тегу <a> з атрибутом class="specific_class"
specificLinks = soup.find_all('a', class_='specific_class')

# Знаходження першого елемента тегу <p>
paragraph = soup.find('p')

# Отримання текстового вмісту елемента
text = paragraph.text
```

Рисунок 1.9 – Основні методи(функції) для роботи з HTML документом за допомогою BeautifulSoup

Джерело: розроблено автором.

Додатково є можливість комбінувати та організовувати теги в колекції як з усього документу, так і з виокремлених тегів нижчого рангу (дочірніх), для цього можна використовувати методи *.parent* та *.children* (рис.1.10).

```
# Знаходження першого елемента тегу <p>
paragraph = soup.find('p')

# Знаходження батьківського елемента
parent = paragraph.parent

# Знаходження всіх дочірніх елементів
children = list(paragraph.children)
```

Рисунок 1.10 – Парсинг даних в межах окремих тегів через навігацію по рангам

Джерело: розроблено автором.

Підбиваючи підсумки, можна стверджувати, що технологія вебскрейпінгу неухильно та всебічно розвивається і за досить тривалий час свого розвитку досягла значної легкості у використанні. Можливості отримувати дані в автоматичному режимі стає все більш знайомою та доступною у загальному розумінні, технологія розвивається у вигляді створення нових бібліотек та фреймворків на базі вже існуючих мов програмування. Слід зазначити, що окрім описаних вище бібліотек існують і інші фреймворки та бібліотек, що дозволяють різними способами організовувати комунікацію з браузерами та серверами та мають додатковий зручний функціонал для скрейпінгу.

1.4. Моделі глибокого навчання для роботи з природною мовою

Окрім звичного аналізу даних в ручному режиму, дата-скрейпінгу та аналізу інформації в напівавтоматичному режимі дуже популярними є методи з застосуванням машинного та глибокого навчання для роботи з текстами. Раніше, до появи великих обсягів даних, розуміння роботи з великими даними (big data) та моделями для навчання на великих обсягах, для обробки текстів вдавалися до використання так званих «регулярних виразів» (regular expression). Дана техніка допомагає створювати та імплементувати строкові моделі для їхнього пошуку в реальних текстах, така можливість є в переважній більшості мов програмування, які працюють зі строковими значеннями (C++, C#, Java, JavaScript, PHP, Python та інші). В мові програмування Python реалізація роботи з регулярними виразами реалізується шляхом використання бібліотеки *re* (regular expression або regex).

Перш ніж розпочати роботу з виразами необхідно вирішити, що саме необхідно знайти в тексті. Потрібно побудувати шаблон (pattern), що задовольнятиме елементу пошуку, для цього використовується окрема мова для regular expression. Нижче наведені деякі основні елементи, які можна використовувати для створення шаблону:

1. Прості символи – в шаблоні відповідають самим собі у тексті.
   Наприклад, шаблон abc буде відповідати стрічці "abc" в тексті.
2. Спеціальні символи – символи, що мають спеціальне значення і використовуються для визначення певних типів символів.
   Наприклад:

. – відповідає будь-якому одиночному символу, крім символу нового рядка.

^ – початок рядка.

$ – кінець рядка.

3. Класи символів – для визначення класів символів використовуються квадратні дужки.
   Наприклад, [аеіоиуї] відповідає будь-якому голосному символу.
4. Кількісні мітки – вказують на кількість повторень попереднього елемента, для реалізації використовуються вирази типу {n}, {n,}, {n, m}.
   Наприклад, \d{5} відповідає п’яти цифрам.
5. Альтернативи – є можливість вказати альтернативні варіанти за допомогою вертикальної риски (|).
   Наприклад, книга|щоденник відповідає або "книга", або "щоденник ".
6. Спеціальні послідовності: використовуються для визначення певних класів символів, таких як \d (цифри), \w (букви та цифри), \s (пробіли), тощо.

Приклади найпопулярніших патернів регулярних виразів мають наступний вигляд:

1. /^(?:19|20)\d\d-(?:0[1-9]|1[0-2])-(?:0[1-9]|[12][0-9]|3[01])$/ – дата у форматі YYYY-MM-DD.
2. /^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$/ – формат електронної пошти користувача.
3. /^((https?|ftp|file):\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/ – формат посилання URL

Даний метод залишається дуже часто вживаним і може знадобитися в нагоді не тільки для аналізу текстів, а і для валлідації даних, введених користувачем, перевірка введений полів на відповідність і не тільки. Однак даний метод не здатен

проаналізувати елементи тексту, які можуть мати лексичні, граматичні або смислові особливості, якщо слово/речення не має особливої структури, з якою можна поєднати, наприклад спеціальні символи, то патерн регулярних виразів не зможе їх виділити та проаналізувати, внаслідок чого виникає необхідність іншого інструментарію для роботи з текстами та їхнього аналізу.

Оскільки робота з текстами в ручному форматі більше не є ефективною в умовах всебічної автоматизації та цифровізації, є потреба у розгляді нових інструментів для роботи з текстовою інформацію. За останні роки дослідження в області машинного навчання та штучного інтелекту здійснили неймовірний прошив, створено велику кількість архітектури нейронних мереж, які заміщають людину в області обробки відео, фото, аудіо і, звичайно, тексту. Для того, що використовувати різноманітні техніки машинного навчання та глибокого навчання, необхідно привести вхідні дані до відповідного та зручного формату для навчання. Розглянемо методи обробки природної мови, за допомогою яких здійснюється підготовка тексту для навчання:

1. «Bag of words» («мішок слів») – метод обробки природної мови, який використовується для перетворення тексту на вектор, який далі можна використовувати в алгоритмах машинного навчання. Основна ідея полягає в тому, щоб розглядати текст як набір окремих слів, ігноруючи порядок та граматичні правила, а замість цього зосереджуючись лише на наявності слів у тексті. Для кожного тексту створюється вектор, який вказує, скільки разів слово зустрілося в тексті. Наприклад, речення «Це розділ для диплому» в алгоритмі «Bag of words» може бути представлено як вектор [1, 1, 1, 1], де кожна компонента вектора відповідає одному слову («Це», «розділ», «для», «диплому»), а значення вказує на кількість повторень кожного слова у текст. Цей метод простий, але ефективний для деяких задач обробки тексту, таких як класифікація документів чи виявлення тем. Однак він не враховує семантичні зв'язки між словами та може втратити інформацію про порядок слів у тексті.

2. Частота термінів – Інверсна частота документів (TF-IDF, Term Frequency–Inverse Document Frequency) – метод оцінки ваги термінів/слів, який

використовується для оцінки важливості слова в контексті корпусу текстів. Цей метод дозволяє виділити ключові слова в конкретних документах, підсилюючи терміни, які характерні для певного документа, але за рідкістю зустрічаються в інших. Компоненти TF-IDF мають наступний вигляд:

2.1. Частота термінів (TF, Term Frequency) – визначає, наскільки часто термін/слово зустрічається в конкретному документі. Частота термінів обчислюється за допомогою формули 1.1:

$$TF(t,d) = \frac{\text{К-ть появ слова } t \text{ в документі } d}{\text{Загальна к-ть слів в документі } d}, \quad (1.1)$$

2.2. Інверсна частота документів (IDF, Inverse Document Frequency) – вимірює, наскільки інформативним є термін на всій вибірці тексту. IDF обчислюється за допомогою формули 1.2, а саме логарифму відношення загальної кількості документів N до кількості документів, в яких зустрічається термін/слово t, важливо також додавати 1 до кількості документів, аби уникнути ділення на 0:

$$IDF(t,D) = \log\left(\frac{N}{\text{К-ть документів в збірці, що містять термін } t + 1}\right), \quad (1.2)$$

2.3. Частота термінів – Інверсна частота документів (TF-IDF, Term Frequency–Inverse Document Frequency) – значення перемноження TF на IDF для отримання оцінки важливості терміна/слова в конкретному документі в межах всього документу, розраховується за формулою 1.3:

$$TF\text{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D), \quad (1.3)$$

Отже, величина TF-IDF для кожного терміна/слова в кожному документі визначає, наскільки важливий термін для цього документа і як він відрізняється від інших документів у збірці документів. Цей метод широко використовується в

задачах інформаційного пошуку, кластеризації текстів та інших завдань обробки природної мови [10].

3. Doc2Vec (Paragraph Vector) – розширена модель Word2Vec, яка генерує векторні представлення слів в межах тексту, натомість Doc2Vec генерує векторні представлення цілих абзаців або документів, вона має кілька ключових компонентів, а її робота базується на певних принципах машинного навчання: PV-DM (Distributed Memory) і PV-DBOW (Distributed Bag of Words).

3.1. PV-DM – модель, що намагається передбачити наступне слово у контексті документа, використовуючи його навколишні контекстні слова з додаванням ідентифікатора абзацу (рис. 1.11).

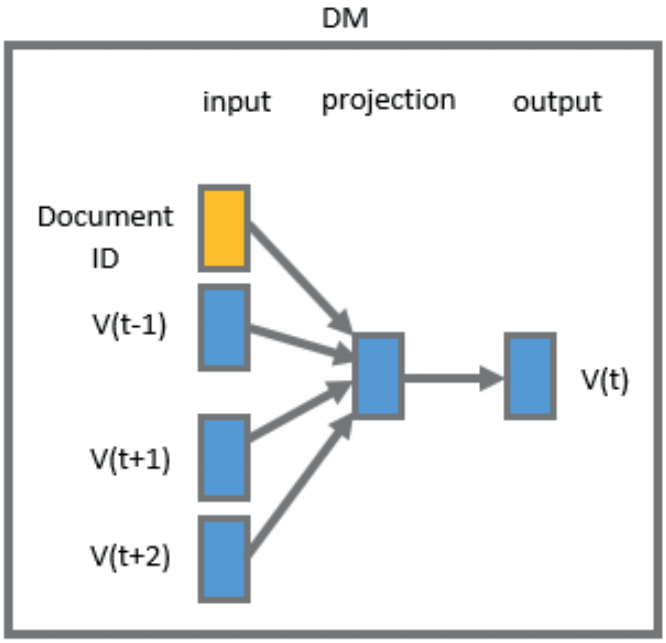

Рисунок 1.11 – Інтерпретація PV-DM моделі

Джерело: [11].

3.2. PV-DBOW – модель, що намагається передбачити випадково вибрані слова з документа, використовуючи тільки вектор документа, приймає ідентифікатор документа як вхідний (рис.1.12).

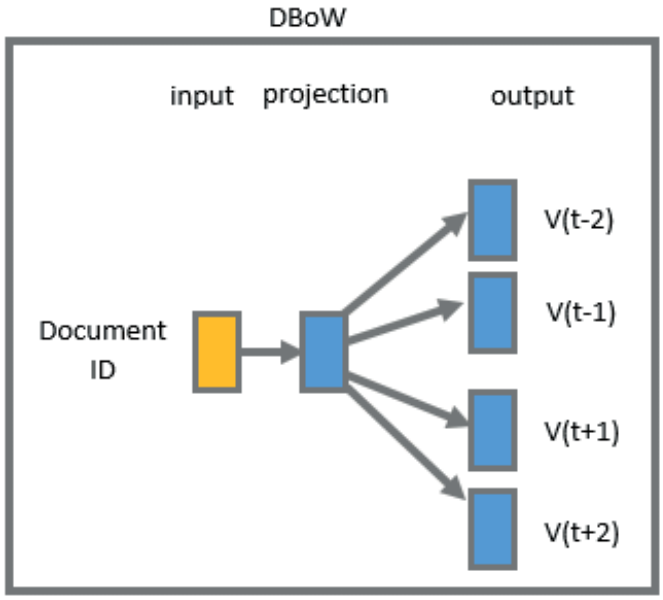

Рисунок 1.12 – Інтерпретація PV-DBOW моделі

Джерело: [11].

3.3. Архітектура нейронної мережі:

Для навчання підмоделей використовується нейронна мережа, яка оптимізується за допомогою градієнтного спуску. Модель включає шари для векторів слів та вектора документа, а також може містити додаткові шари для внутрішнього представлення. Doc2Vec намагається мінімізувати втрати (losses) між фактичним та передбачуваним словом у контексті документа. Процес навчання охоплює багато ітерацій через всю збірку текстів. Має гіперпараметри, такі як розмір вектора, швидкість навчання, кількість шарів, кількість ітерацій та інші, які можна оптимізувати для кращої продуктивності. Таким чином, Doc2Vec можна використовувати, щоб знайти аналогічні документи, це може допомогти знайти позитивні і негативні відгуки, тому що, в цілому, документ з позитивними відгуками буде мати відповідний вектор, а документ з негативними відгуками буде мати інший відповідний вектор [11].

4. N-gram (N-гами) – використовуються для задач моделювання мови, де метою є передбачення ймовірності слова з огляду на його попередній контекст. У моделі unigram (1-гама) кожне слово розглядається незалежно, тоді як моделі bigram (2-гами) розглядають пари послідовних слів. Моделі Trigram (3-гами) і

Higer-Order (N-гам) захоплюють довші залежності. Незважаючи на їх простоту, N-грамові моделі пропонують базову лінію для моделювання мови і корисні в сценаріях, де більш складні моделі є затратними з точки зору реалізації ресурсу [12].

Досить зручною практикою є класифікація текстів, або окремих елементів в тексті, існує багато методів, якими можна реалізувати дану можливість, оскільки, кожне речення/абзац/документ можливо представити у вигляді вектора чисел та визначати залежності між ними. Алгоритми класифікації, як і будь-які інші алгоритми, можуть відрізнятися своєю складністю, тобто затребуваними технічними потужностями, які необхідні для виконання всіх ітерацій алгоритму і як результат навчання моделі та її використання, проте на великій навчальній вибірці можуть давати результати з незначною різницею похибок, розглянемо два зарекомендованих алгоритми для класифікації текстів/елементів тексту різної складності:

1. Згорткові нейронні мережі (Convolutional Neural Networks, CNN).

1.1. Задачі, які можуть виконуватися моделлю:

- Класифікація зображень
- Визначення об'єктів
- Класифікація текстових матеріалів
- Аналіз тональності

1.2. Реалізація моделі:

CNN умовно поділені на 3-и шари: шар-згортка, шар-пулінг, шар-з'єднання. Шар згортки є основним будівельним блоком CNN та займає основну частину обчислювального потужності нейронної мережі. Цей шар виконує скалярний добуток між двома матрицями, де одна матриця – це набір вивчених параметрів, інакше відомих як ядро/фільтр/матриця вагів, яка корегується в процесі мінімізації функції втрат, а інша матриця – обмежена частина рецептивного поля (рис. 1.13).

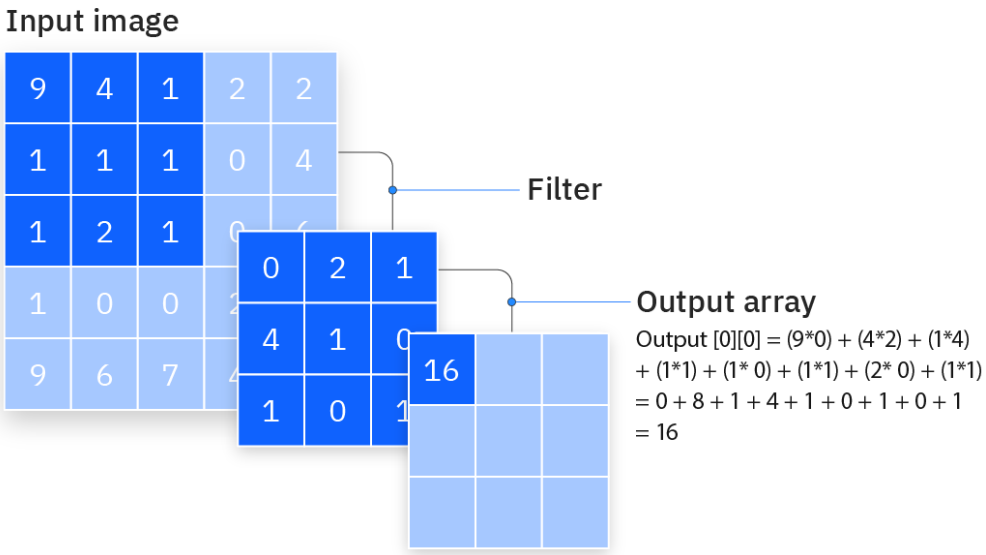

Рисунок 1.13 – Реалізація процесу згортки CNN

Джерело: [13].

Ядро має просторові розміри, менші, ніж зображення, але більші за глибину. Це означає, що, якщо зображення складається з трьох (RGB) шарів, висота і ширина ядра буде просторово невеликою, але глибина простягається до всіх трьох шарів. під час прямого проходження ядро переміщується вздовж висоти та ширини зображення, створюючи зображення репрезентації тієї рецептивної області. Це породжує двовимірне представлення зображення, відоме як карта активації, яка показує відповідь ядра на кожній просторовій позиції зображення. Розмір переміщення ядра називається кроком. Якщо у нас є вхід розміром $W \times W \times D$ та D_{out} кількість ядер із просторовим розміром F, з кроком S та кількістю додаткового заповнення P, то розмір вихідного об'єму можна визначити за наступною формулою 1.4.

$$W_{\text{out}} = \frac{W - F + 2P}{S} + 1, \tag{1.4}$$

Тривіальні шари нейронних мереж використовують множення матриць на матрицю параметрів, яка описує взаємодію між вхідною та вихідною одиницею. Це означає, що кожна вихідна одиниця взаємодіє з кожною вхідною одиницею. Однак

згорткові нейронні мережі мають розріджену взаємодію. Це досягається тим, що розмір ядра менший за вхід, наприклад, зображення може мати мільйони або тисячі пікселів, але оброблюючи його за допомогою ядра, ми можемо виявити значущу інформацію, яка становить десятки чи сотні пікселів. Це означає, що нам потрібно зберігати менше параметрів, що не лише зменшує потребу в виділенні додаткової пам'яті та навантаженні на процесор, але також покращує статистичну ефективність моделі.

Якщо обчислення однієї ознаки в точці простору (x_1, y_1) є корисним з точки зору корегування ваг, то воно також повинно бути корисним у іншій точці простору, скажімо (x_2, y_2). Це означає, що для одного двовимірного зрізу, тобто для створення однієї карти активації, нейрони зобов'язані використовувати той самий набір ваг. У традиційній нейронній мережі кожен елемент матриці ваг використовується один раз і потім більше не використовується, тоді як у згортковій мережі є спільні параметри, тобто ваги, застосовані до одного входу, такі ж, як ваги, застосовані в іншому місці. Через спільне використання параметрів шари згорткової нейронної мережі матимуть властивість еквіваріантності до зміщення. Це означає, що якщо ми змінимо вхід таким чином, вихід також зміниться так само.

Шар-пулінг замінює вивід мережі в певних місцях, отримуючи статистику об'єднаного вигляду навколишніх виводів. Це допомагає зменшити просторовий розмір представлення, що зменшує обсяг обчислень та ваг. Операція пулінгу обробляється окремо для кожного зрізу представлення. Існує кілька функцій пулінгу, таких як середнє значення прямокутного оточення, L_2-норма прямокутного оточення та зважене середнє значення на основі відстані від центрального пікселя. Однак найпопулярнішим процесом є максимальний пулінг, який повідомляє максимальний вивід з оточення.

Якщо у нас є карта активацій розміром $W \times W \times D$, ядро пулінгу із просторовим розміром F та кроком S, то розмір вихідного об'єму можна визначити за формулою 1.5.

$$W_{\text{out}} = \frac{W - F}{S} + 1, \tag{1.5}$$

Це призведе до вихідного об'єму розміром $W_{out} \times W_{out} \times D$. У всіх випадках пулінг надає певну інваріантність до зсуву, що означає, що об'єкт буде впізнаваний незалежно від того, де він з'являється на кадрі. В шарі повного з'єднання є повна взаємодія з усіма нейронами в попередньому і наступному шарах, як це бачимо в звичайних повністю з'єднаних нейронних мережах. Тому його можна обчислити, як завжди, за допомогою множення матриць, за яким слідує ефект зсуву (баєс). Шар-з'єднання допомагає відображати представлення між входом та виходом [14].

У випадку завдань з обробки природної мови (NLP), коли застосовується до тексту замість зображень, ми маємо одновимірний масив, який представляє текст. Тут архітектура згорткових мереж змінюється на операції згорткового і пулінгу у одному вимірі. Одним з типових завдань у NLP, де використовуються згорткові мережі, є класифікація речення, тобто визначення класу для речення заздалегідь визначених категорій, враховуючи N-грами, тобто слова або послідовності слів, або також символи чи послідовності символів.

2. Наївний Баєсівський класифікатор (Naive Bayes).

2.1.Задачі, які можуть виконуватися моделлю:

- Фільтрація спаму
- Автоматичне маркування категорій
- Аналіз тональності текстів

2.2.Реалізація моделі:

Наївний метод Баєса – це алгоритм керованого навчання, заснований на застосуванні теореми Баєса з «наївним» припущенням про умовну незалежність між кожною парою ознак, враховуючи значення змінної класу. Теорема Баєса стверджує наступне відношення (1.6), задане змінною класу y і залежним вектором ознак x_1 через x_n.

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}, \tag{1.6}$$

Використовуючи наївне припущення про умовну незалежність описане виразом 1.7, для всіх i цей зв'язок спрощується до виразу 1.8.

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y), \tag{1.7}$$

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}, \tag{1.8}$$

Оскільки $P(x_1, \dots, x_n)$ є постійною з урахуванням вхідних даних, ми можемо використовувати правило класифікації 1.9.

$$\begin{gathered} P(y \mid x_1, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y), \\ \Downarrow \\ \hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i \mid y) \end{gathered} \tag{1.9}$$

Використовуючи оцінку апостеріорного максимуму (MAP, Maximum a Posteriori) для оцінки $P(y)$ та $P(x_i|y)$ відносну частоту класу y в навчальному наборі можемо описати як вираз 1.10.

$$\theta_{\text{MAP}} = \arg\max_\theta P(\theta|D) = \arg\max_\theta \frac{P(D|\theta) \cdot P(\theta)}{P(D)}, \tag{1.10}$$

Наївні класифікатори Баєса можуть бути надзвичайно швидкими порівняно з більш складними методами. Відокремлення розподілу умовних ознак класу означає, що кожен розподіл може бути незалежно оцінений як одновимірний

розподіл. Це допомагає обійти часту проблему з великою розмірністю та полегшити навантаження на виробничі технічні потужності [15].

На сьогодні існують нейронні мережі, які можуть не тільки класифікувати елементи в тексті та залежності між словами, але й генерувати тести та передбачувати наступні елементи речень, навчаючись на послідовностях слів у реченнях. Найпопулярнішими алгоритмами реалізації таких процесів є:

1. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN):

1.1.Задачі, які реалізовує алгоритм:

- Машинний переклад.
- Генерація тексту на основі вхідного контексту.
- Аналіз послідовностей

1.2.Реалізація моделі.

Рекурентні нейронні мережі використовують однакові ваги для кожного елемента послідовності, зменшуючи кількість параметрів і дозволяючи моделі узагальнювати послідовності різної довжини. RNN узагальнюються до структурованих даних, крім послідовних, таких як географічні або графічні дані, завдяки своєму дизайну. У стандартних нейронних мережах всі вхідні та вихідні дані незалежні одне від одного. Однак у деяких випадках, наприклад, при прогнозуванні наступного слова у реченні, необхідно враховувати попередні слова, і тому потрібно, щоб попередні слова також запам’ятовувалися. В результаті була створена рекурентна нейронна мережа (RNN), яка використовує прихований шар для подолання цієї проблеми. Найважливішою частиною RNN є прихований стан, який запам'ятовує конкретну інформацію про послідовність [16].

Рекурентні нейронні мережі, зазвичай, мають архітектуру, при якій на кожен часовий крок t , активація $a^{<t>}$ і результат $y^{<t>}$ виражаються формулою 1.11.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{і} \quad y^{<t>} = g_2\left(W_{ya}a^{<t>} + b_y\right), \qquad (1.11)$$

де W_{ax} , W_{aa} , W_{ya} , b_a , b_y – це коефіцієнти, які виділені тимчасово, і g_1 , g_2 – функції активації.

Графічне представлення архітектури мережі зображено на рис. 1.14.

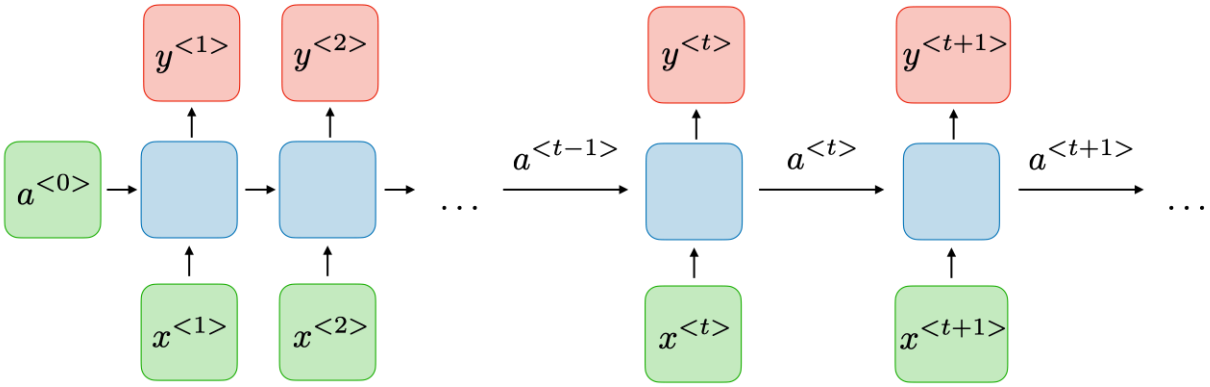

Рисунок 1.14 – Архітектура рекурентної нейронної мережі

Джерело: [16].

Найбільш популярними для даної мережі функції активації такі:

- Сигмоїда (Sigmoid): $g(z) = \frac{1}{1+e^{-z}}$
- Гіперболічний тангенс (Tanh): $g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- Випрямлений лінійний вузол (RELU): $g(z) = \max(0, z)$

Графічне зображення функцій активації зображено на рис. 1.15.

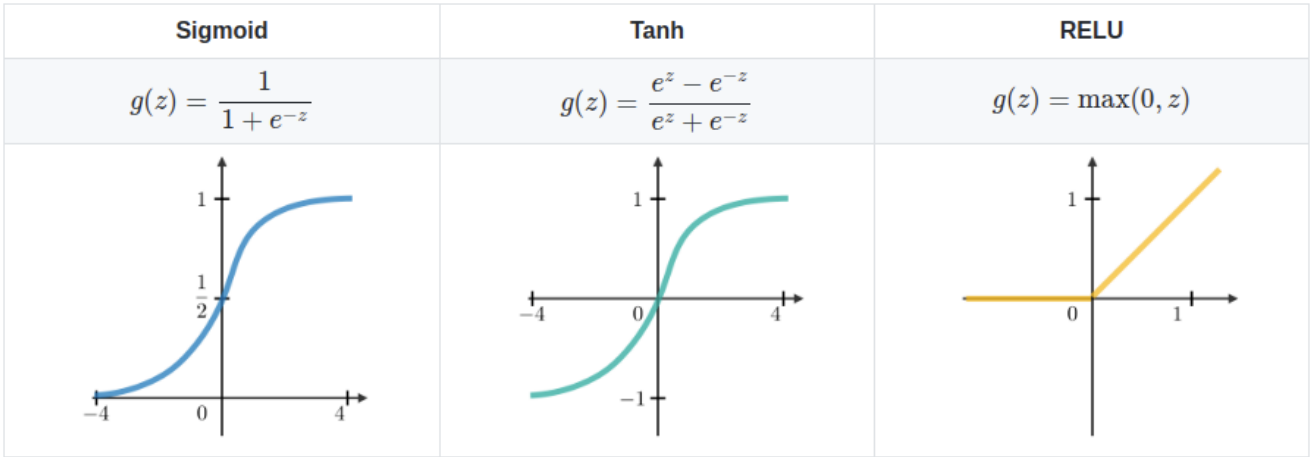

Рисунок 1.15 – Найпопулярніші функції активації нейронів

Джерело: [17].

Функція активації нейрона визначає, чи слід вмикати чи вимикати його. Нелінійні функції зазвичай перетворюють вихідний результат нейрона на число від 0 до 1 або -1 до 1.

В залежності від архітектурної структури, рекурентні нейронні мережі можуть бути таких типів (рис. 1.16):

- Один до Одного (One to One)
- Один до Багатьох (One to Many)
- Багато до Одного (Many to One)
- Багато до Багатьох (Many to Many)

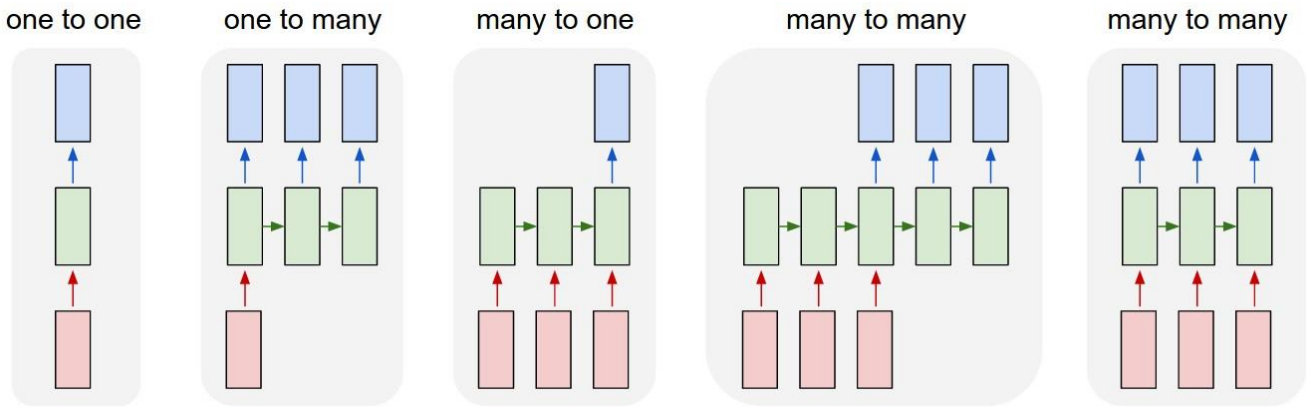

Рисунок 1.16 – Типи рекурентних нейронних мереж

Джерело: [18].

1. Один до Одного (One to One) ($T_x = T_y = 1$) – це найбільш базовий та традиційний тип архітектурної структури нейронної мережі, що дає один вихід для одного входу (рис. 1.16), використовується для вирішення звичайних задач машинного навчання.
2. Один до Багатьох (One to Many) ($T_x = 1,\ T_y > 1$) – це вид архітектури RNN, яка застосовується в ситуаціях, де для одного входу отримується кілька виходів. Основний приклад застосування – генерація музики. В моделях генерації музики RNN використовуються для створення музичного твору (кілька виходів) з однієї музичної ноти (один вхід).
3. Багато до Одного (Many to One) ($T_x > 1,\ T_y = 1$) – зазвичай застосовується в моделях аналізу настрою, цей тип моделі використовується, коли для отримання одного виходу потрібні кілька входів. Наприклад, у моделі текстовий ввід (слова як декілька входів) вказує його фіксований настрій (один вихід). Іншим прикладом може бути модель оцінки фільмів, яка використовує текстові відгуки як вхід, щоб надати рейтинг фільму від 1 до 5.

4. Багато до Багатьох (Many to Many) ($T_x > 1,\ T_y > 1$) – приймає декілька входів і надає декілька виходів. Проте багато-до-багатьох моделі можуть бути двох видів:
   4.1. $T_x = T_y$ – це випадок, коли розміри вхідного та вихідного шарів співпадають. Ця форма RNN використовується в розпізнаванні іменованих сутностей.
   4.2. $T_x \neq T_y$ – може бути представлена в моделях, де розміри вхідного та вихідного шарів відрізняються. Найбільш поширене застосування такої архітектури RNN багато-до-багатьох зустрічається в машинному перекладі. Наприклад, «I Love you», перекладаються лише двома словами на іспанську – «te amo». Таким чином, моделі машинного перекладу можуть повертати слова більше або менше, ніж вхідний рядок через нерівну архітектуру RNN багато-до-багатьох, яка працює в фоновому режимі [18].
2. Довгострокова короткострокова пам'ять (Long short-term memory, LSTM):
   2.1. Задачі, які реалізовує модель:
   - Розпізнавання часових послідовностей.
   - Генерація тексту.

   2.2. Реалізація моделі.

LSTM є моделлю, що побудована на базі RNN, однак є доповненою та виключає недолік, який називається «втратою короткострокової пам’яті». Основне обмеження RNN полягає в тому, що дані моделі не можуть запам'ятати дуже довгі послідовності і потрапити в проблему «поступового зникаючого градієнта». Градієнти функції втрат в нейронних мережах наближаються до нуля, коли додається більше шарів з певними функціями активації, що ускладнює тренування мережі. Мережі LSTM приходять на допомогу для вирішення проблеми «зниклого градієнту». Вони роблять це, ігноруючи/забуваючи непотрібні дані/інформацію в мережі: LSTM буде забувати дані, якщо від інших вхідних даних (слів у попередньому реченні) не надходить корисної інформації. Коли приходить нова

інформація, мережа визначає, яку інформацію проігнорувати, а яку залишити в пам'яті. Розглядати модель зручно, порівнюючи її з RNN (рис. 1.17):

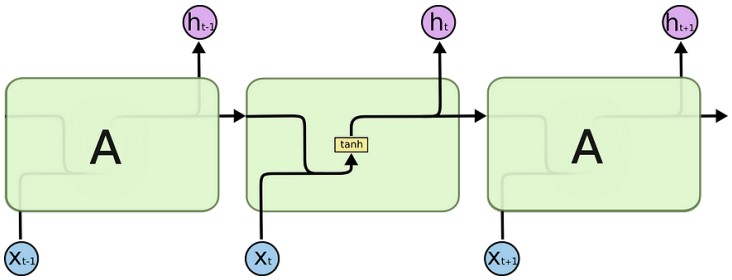

Рисунок 1.17 – Графічна інтерпретація логіки моделі RNN з гіперболічним тангенсом, як функція активації

Джерело: [19].

У мережах LSTM, замість простої мережі з однією функцією активації, ми маємо декілька компонентів, які надають мережі можливість забувати та запам'ятовувати інформацію (1.18).

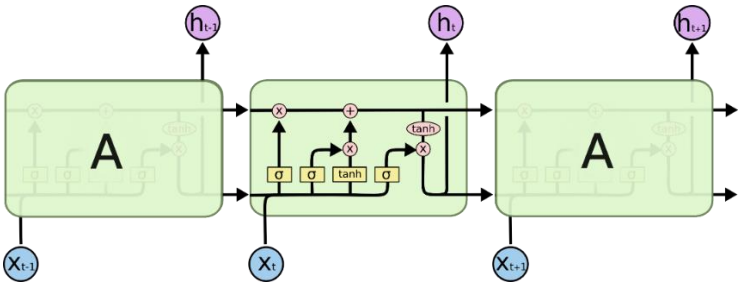

Рисунок 1.18 – Графічна інтерпретація логіки моделі LSTM з гіперболічним тангенсом, як функція активації

Джерело: [19].

Де,  *– шар нейронної мережі,*

 *– точкова операція,*

 *– трансфер-вектор.*

LSTM мають 4 різних компоненти, розглянемо їх докладніше.

Стан пам’яті (Memory cell) – відповідає за запам'ятовування та забування. На основі контексту введення, тобто що деяку попередню інформацію слід запам'ятовувати, а деяку забувати, і частину нової інформації слід додати до пам'яті. Перша операція (X) - це поелементна операція, яка є множенням стану

пам’яті на масив $[-1, 0, 1]$. Інформація, помножена на 0, буде забута LSTM. Інша операція (+) відповідає за додавання нової інформації до стану пам’яті (рис. 1.19).

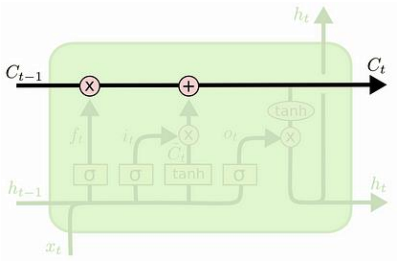

Рисунок 1.19 – LSTM Стан пам’яті (Memory cell)

Джерело: [19].

«Забування» (Forget gate) – вирішує, яку інформацію слід «забути», видалити як корегування параметрів моделі, для прийняття цього рішення використовується шар з сигмоїдою, як функцією активації (1.12), цей шар сигмоїди називається «відсіюванням забуття» (forget gate layer). Він виконує скалярний добуток $h_{(t-1)}$ і $x_{(t)}$ та за допомогою шару сигмоїди виводить число від 0 до 1 для кожного числа в стані пам’яті $C_{(t-1)}$. Якщо вивід «1» – ми зберігаємо цю інформацію, якщо «0» – гарантоване повне забуття інформації. Графічне представлення зображено на рис. 1.20.

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big), \tag{1.12}$$

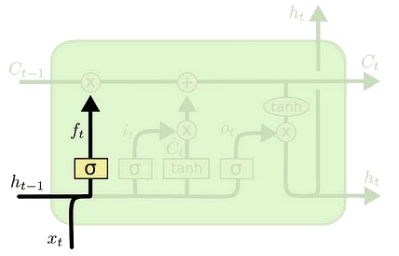

Рисунок 1.20 – LSTM забування (Forget Gate)

Джерело: [19].

1. Вхід (Input gate)

Надає нову інформацію LSTM і вирішує, чи слід цю нову інформацію зберігати в стані пам’яті. Компонент складається з 3-х частин:

1.1. Шар сигмоїди (1.13) визначає значення для оновлення. Цей шар називається «шар входу».

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{1.13}$$

1.2. Шар гіперболічного тангенса (1.14), як функції активації, створює вектор нових кандидатських значень $\widetilde{C_t}$, які можна додати до стану.

$$\widetilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{1.14}$$

Графічна інтерпретація кроків 1.1. та 1.2. зображена на рис. 1.21.

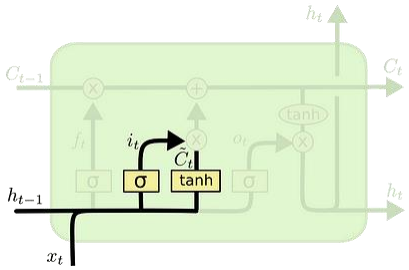

Рисунок 1.21 – LSTM Input Gate

Джерело: [19]

1.3. В результаті ми комбінуємо два виводи як скалярний добуток $i_t \times C_t$ і оновлюємо новий стан пам'яті C_t , який отримується додаванням виводів від «відсіювання забуття» та «входу» (1.15).

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}, \tag{1.15}$$

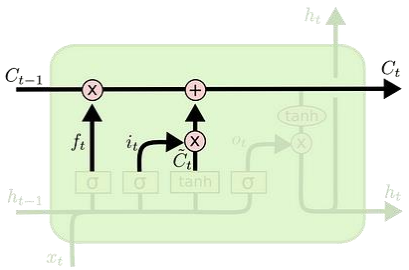

Рисунок 1.22 – LSTM новий стан пам'яті (Memory cell)

Джерело: [19].

2. Вихід (Output gate)

Вихід значення LSTM залежить від нового стану пам’яті. Першим кроком шар функції активації сигмоїди (1.16) вирішує, які частини стану пам’яті ми будемо виводити,

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \tag{1.16}$$

другим кроком шар з функцією активації з гіперболічним тангенсом (1.17) використовується на стані пам’яті для «стискання» значень від -1 до 1, які нарешті множаться на вивід сигмоїдного шару.

$$h_t = o_t * \tanh(C_t), \tag{1.17}$$

Графічна репрезентація output gate зображена на рис. 1.23.

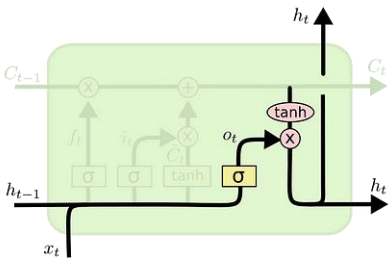

Рисунок 1.23 – LSTM Output gate

Джерело: [19].

Розглянемо бібліотеку Python, яка реалізує описані методи, алгоритми та практики для роботи з природною мовою та є однією з найпопулярніших на даний час – spaCy. SpaCy імплементує готові навчені моделі для роботи з різними мовами. Тегування, синтаксичний аналіз, категоризація тексту та багато інших компонентів spaCy працюють на статистичних моделях. Кожне «рішення», яке приймають ці компоненти, наприклад, «яку мітку частини мови призначити» або «чи слово є іменованою сутністю», – це прогноз на основі поточних значень ваг моделі. Навчання nlp моделі – це ітеративний процес, в якому прогнози моделі порівнюються з анотаціями-зразками для оцінки градієнта втрат. Градієнт втрат використовується для обчислення градієнта ваг за допомогою методу «зворотного розповсюдження». Градієнти показують, як слід змінити значення ваг, щоб

прогнози моделі стали більш схожими на позначки-зразки. Загальна модель навчання моделей мови spaCy зображена на рис. 1.24.

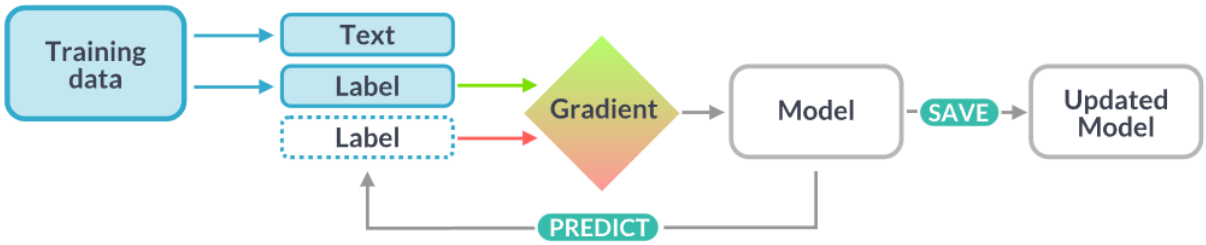

Рисунок 1.24 – Модель навчання nlp-моделі spaCy

Джерело: [20].

Модель включає в себе декілька кроків обробки текстів, які реалізують її функціонал, ці кроки називаються «конвеєром обробки» або «пайплайном» (рис. 1.25).

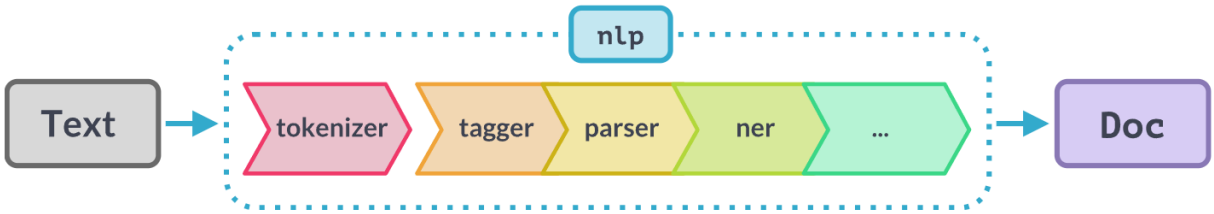

Рисунок 1.25 – Конвеєр обробки вхідного тексту nlp-моделі spaCy

Джерело: [20].

Даний код (рис. 1.26) є реалізацією зображеного конвеєру обробки мови.

```
import spacy

# Завантаження навченої моделі мови, в даному разу української
nlp = spacy.load("uk_core_news_sm")

# Використання мовних патернів української мови для аналіу тексту
doc = nlp("Цей текст є прикладом для використання широкого функціоналу nlp-моделі spaCy")
```

Рисунок 1.26 Реалізація застосування моделі spaCy на тексті

Джерело: розроблено автором.

В даному випадку, за допомогою короткого запису модель української мови, яка вже навчена та великій кількості україномовних текстів, завантажується та присвоюється змінній «nlp» та використовуються для вхідного тексту, в подальшому для реалізації функціоналу бібліотеки використовуватиметься змінна «doc», можливості можуть бути наступні:

1. Лінгвістичні анотації (Linguistic annotations) – spaCy надає різноманітні лінгвістичні анотації, щоб зрозуміти граматичну структуру тексту/окремих речень. Це включає типи слів, такі як частини мови, та їх взаємозв'язки.

2. Токенізація (Tokenization) – під час обробки spaCy спочатку токенізує текст, тобто розділяє його на слова, пунктуацію і так далі. Це робиться застосуванням правил, специфічних для кожної мови. Наприклад, пунктуацію в кінці речення слід розділяти, тоді як «U.K.» повинно залишитися одним токеном. Кожен документ складається з окремих токенів, і ми можемо їх ітерувати. Для реалізації задачі використовується модель CNN.

3. Визначення частин мови (Part-of-speech tags and dependencies) – після токенізації spaCy може розбирати та мітити заданий документ. Саме тут використовується навчений конвеєр та його статистичні моделі, які дозволяють spaCy робити прогнози щодо того, яка мітка або позначка ймовірно найбільше підходить в даному контексті. Навчений компонент включає бінарні дані, які формуються за допомогою достатньої кількості прикладів для того, щоб система могла робити узагальнені прогнози по всій мові – наприклад, слово, яке слідує після артикля в англійській мові, ймовірно, є іменником.

4. Найменування сутностей (Named Entities) – це визначення слів в тексті, як «об'єктів реального світу», яким призначено ім'я, наприклад, особа, країна, продукт чи назва книги. spaCy може визначати різні типи іменованих сутностей у документі. Оскільки моделі є статистичними та сильно залежать від прикладів, на яких вони були навчені, це не завжди працює ідеально і може потребувати налаштування в самому пайплайні користувачем, в залежності від використання.

Визначення схожості (Word vectors and similarity) – схожість визначається порівнянням векторів слів або «вбудовувань слів» – багатовимірних представлень значень слова. Вектори слів можуть бути створені за допомогою алгоритму, наприклад, Word2Vec.

РОЗДІЛ 2

АВТОМАТИЗАЦІЯ ПРИСКОРЕНОЇ ОБРОБКИ МЕДІА КОНТЕНТУ МЕТОДАМИ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ

2.1. Побудова тестового завдання та архітектури додатку для обробки текстового контенту

Наразі існує велика кількість просторів, на яких видання можуть поширюватись та просувати власний медіа контент: вебсайти , соціальні мережі, відео платформи, додатки для новин, месенджери, агрегатори новин та інюю Попри різноманітність джерел отримання інформації, саме вебдодатки займають перше місце як ресурси, з яких люди черпають новини та інформацію. Принаймні такий тренд зберігається, оцінюючи дослідження американського центру досліджень «Pew Research Center», який проводить щорічні дослідження сприйняття новинного медіа контенту в США [21]. Розглядаючи розподіл споживання контенту за джерелами інформації бачимо (рис. 2.1), що майже половина всіх опитаних взагалі не споживають новинний контент з будь-яких джерел (42%), найбільше серед тих, хто споживає, припадає на такі джерела, як: сайти чи додатки (25%), пошукові мережі (15%), соціальні мережі (12%), подкасти (6%).

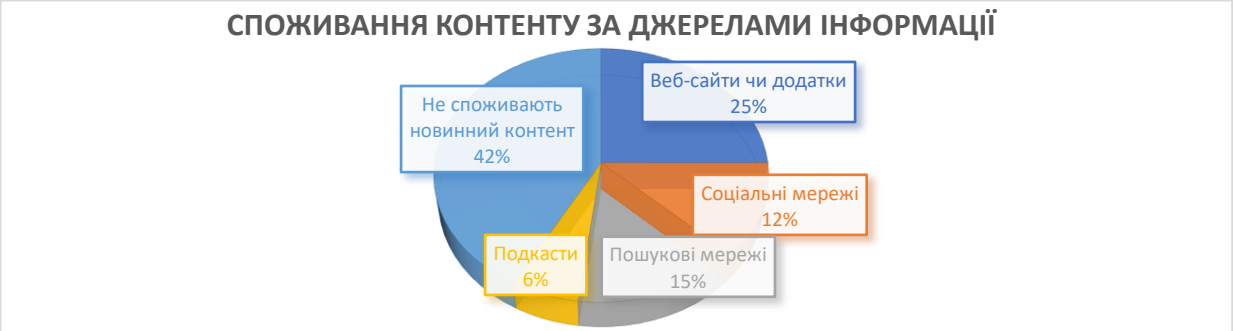

Рисунок 2.1 – Споживання контенту за джерелами інформації

Джерело: розроблено автором на основі [20].

Зважаючи на таку важливість контенту, що публікується та споживається саме з використанням вебпорталів виникає необхідність в ресурсах, що здатні в більш швидкій та доступній формі обробляти дані та виводити статистику з певною результуючою інформацією. В умовах щоденних публікацій великої кількості текстів, найманим працівникам медіа сфери: журналістам-аналітикам, репортерам, редакторам та іншим, доводиться обробляти величезні масиви інформації на регулярній основі, аби знайти актуальну для розділу їхньої аналітики тему. Це потребує значних витрат часу та людського ресурсу, який може бути використаний на більш затребувані та актуальні задачі. Під час моніторингу інформаційного простору журналістам часто доводиться відвідувати сайт кожного видання окремо, аби переглядати та обробляти статті, що публікує кожне видання, що є не зручним та затратним процесом. Аби забезпечити ефективну та швидку роботу журналістів, було б доречно створити єдиний простір для моніторингу та огляду основної інформації різноманітних опублікованих статей.

Для розробки додатку необхідно визначитися з основним функціоналом додатку. Загальне призначення додатку аналізу медіа контенту має містити чіткі параметри входу та параметри виходу після проведення певних внутрішніх операцій (рис. 2.2.), в якості вхідних параметрів додаток прийматиме текст.

Рисунок 2.2 – Призначення додатку аналізу медіа контенту

Джерело: розроблено автором.

Після сформованого розуміння призначення та функціоналу додатку, можна приступити до побудови більш детальної блок-схеми взаємодії користувача та ПЗ, через представлення основного алгоритму роботи у зручній формі, а саме з

використанням графічних фігур в залежності від етапу виконання алгоритму. Для побудови блок-схеми відповідно використовуватимуться окремі елементи [22].

– Процес. Символ відображає виконання процесу, операції вводу/виводу.

– Термінатор. Символ визначає вхід в зовнішнє середовище та вихід з нього, тобто початок та кінець програми.

– Рішення. Символ визначає функцію/рішення, має один вхід та декілька виходів, в залежності від результату рішення.

– Процес (Процедура). Символ означає виконання процесу (процедури) обробки інформації будь-якого вигляду, декількох процесів обробки інформації.

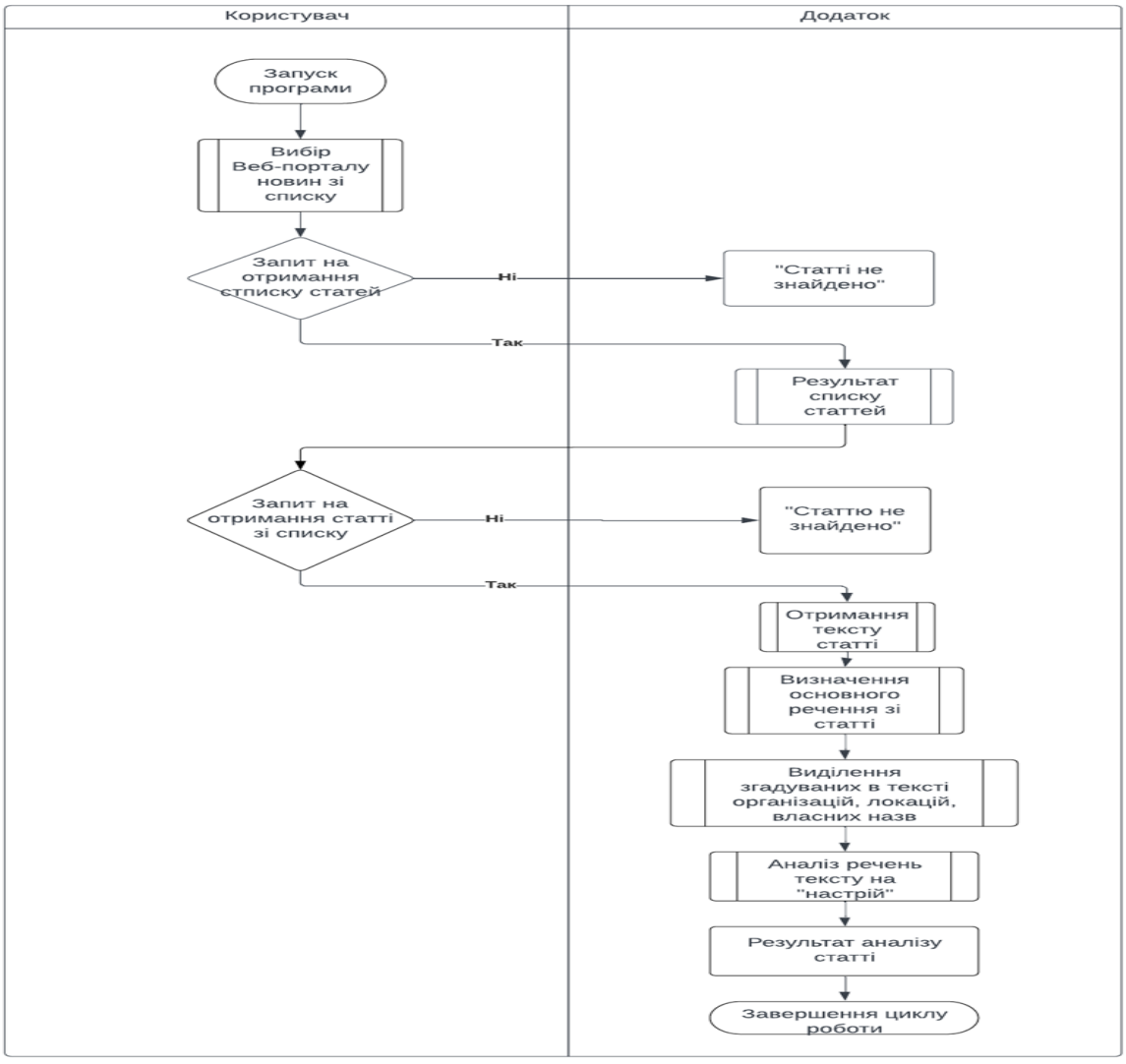

Рисунок 2.3 – Блок-схема додатку для аналізу медіа контенту

Джерело: розроблено автором

Отже, головна мета додатку – отримувати списки статей з вебсторінок в Інтернеті, обирати бажану статтю зі списку та отримувати короткий та змістовний огляд по обраній статті з узагальненням, списками наявних в статті організацій, географічних назв, осіб та аналіз на «настрій» статті. Після розробки графічної репрезентації алгоритму (рис. 2.3.) та визначення кінцевої мети можна приступити до визначення методів, засобів та інструментів, за допомогою яких даний алгоритм буде реалізований практично.

Для реалізації додатку використовуватиметься мова програмування Python. Python – це динамічна, об’єктно орієнтовна універсальна мова програмування, вона використовується для роботи з базами даних і оброблення текстів, ігрові індустрії, програмування графічних інтерфейсів, вебдодатків, серверних клієнтів тощо. Зручність мови Python основана на тому, що вона є мовою високого рівня, має набір конструкцій структурного програмування та підтримує модульність. Гнучкість та універсальність мови Python забезпечує її широке розповсюдження [23].

Увесь додаток буде умовно поділений на такі частини (рис. 2.4):

1. User Interface (UI) – візуальне представлення робочого додатку з комбінуванням функціоналу
2. ML Model (MODEL) – навчені моделі машинного навчання, необхідні для виклику класифікації
3. Functions (FUNC) – компліментарні функції, функції виклику та навчання моделі.

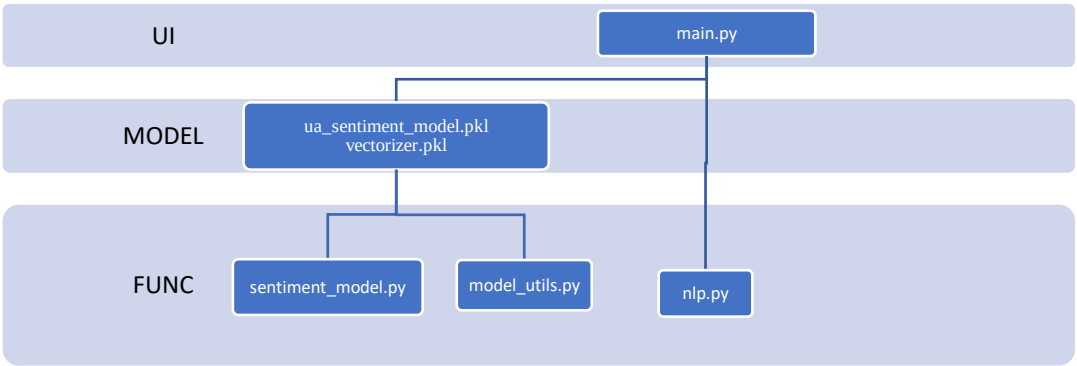

Рисунок 2.4 – Структура ttkbootstrap додатку

Джерело: розроблено автором

Для представлення візуальної (user interface) частини використовувалася новітня бібліотека Ttkbootstrap. Ttkbootstrap – це надбудова для tkinter (більш стара бібліотека для побудови інтерфейсів), яка дозволяє створювати сучасні інтерфейси користувача, з вбудованими стилями Bootstrap, вона включає в себе велику кількість вбудованих графічних тем та можливість створювати власні. Основна ідея ttkbootstrap полягає в тому, щоб надати можливість створювати інтерактивні вікна, кнопки, текстові поля та інші елементи інтерфейсу для взаємодії з користувачем.

Основна ідея бібліотек Ttkbootstrap та Tkinter – вибудовування ієрархічного дерева у формі контейнерів, що входять до складу головного контейнера (root/mainWindow), та розміщуються в його межах, відповідно. В подальшому може створюватися та комбінуватися не обмежена кількість контейнерів, які також можуть вміщувати контейнери (рис. 2.5).

```python
import ttkbootstrap as ttk

# Створення головного вікна (root)
root = ttk.Tk()

# Створення контейнеру з кнопкою
button = ttk.Button(root, text="Click me!")
button.config(style="success")

# Додавання контейнеру з кнопкою до головного вікна
button.pack()

# Цикл для безперевійної роботи програми
root.mainloop()
```

Рисунок 2.5 – Структура ttkbootstrap додатку

Джерело: розроблено автором

Загальний інтерфейс додатку складатиметься з 2-х вікон, що забезпечуватимуть зручне розташування логічних елементів:

1. Вікно з усіма статтями
    1.1. Вікно для запиту
    1.2. Вікно з результатом запиту
2. Вікно з аналітикою по окремій статті

2.2. Розробка інтерфейсу користувача (UI) додатку

Розробка додатку та кожного логічного елементу додатку буде реалізована в послідовній та плановій структурі відповідно до логічного виконання дій користувача. Кожен елемент буде оброблений відповідно до плану:

1. Реалізація елементу у вигляді коду на Python.
2. Опис реалізації.
3. Результат реалізації.

1. Вікно для запиту статей:

1.1.Розглянемо реалізацію вікна для запиту статей (рис. 2.6).

```
# Головне вікно
mainWindow = tb.Window(themename= "darkly")

# mainWindow.geometry("1200x600")
mainWindow.title("Оглядач новин")
mainWindow.iconbitmap('kneu.ico')

# Поділ на розділ з усіма статтями та аналозом статті
notebook = tb.Notebook(mainWindow, bootstyle ="dark")
notebook.pack(fill=tb.BOTH, expand=True)

articleAll = tb.Frame(notebook)
articleAnalisis = tb.Frame(notebook)

# Поділ розділу з усіма статтями на частину для запиту статей (articleAllRquest) та результат запиту (articleAllResponse)
articleAllRquest = tb.Frame(articleAll)
articleAllRquest.pack(side=tb.LEFT, fill=tb.BOTH, pady=10, padx=10)

articleAllResponse = tb.Frame(articleAll)
articleAllResponse.pack(side=tb.RIGHT, expand=True, fill=tb.BOTH, pady= 5, padx=5)

# Створення випливаючого вікна з доступними виданням новин для запиту на статті
issueType = ["Економічна правда", "УНІАН Економіка"]
issueTypeChosen = tb.Combobox(articleAllRquest, bootstyle = "success", values = issueType)
issueTypeChosen.set("Оберіть видання")
issueTypeChosen.pack(pady = 20)

# Створення кнопки для виклику запиту на статті
requestButton = tb.Button(articleAllRquest, text = 'Отримати статті', command = RequestArticles)
requestButton.pack(pady = 5)
```

Рисунок 2.6 – Реалізація вікна з можливістю надсилання запиту

1.2. Опис реалізації:

1.2.1. Створення контейнеру «Window» головного вікна програми mainWindow, додана назва програми, рамка для виводу.

1.2.2. Створення контейнеру «Notebook» початкового розділу програмних вікон, 2- а контейнери «Frame» на розділ з усіма статтями (articleAll) та

розділ для аналізу окремої статті (articleAnalisis), в рамках контейнеру Notebook.

1.2.3. Створення 2-х контейнерів «Frame» для поділу вікна articleAll на вікно для запиту всіх статей (articleAllRequest) та результату запиту (articleAllResponse)

1.2.4. Створення випливаючого вікна з назвами доступних видань, в нашому випадку це українські загальнодоступні видання «Економічна правда» та розділ з новинами економіки видання «УНІАН».

1.2.5. Створення кнопки, при натиску на яку викликатиметься подія «RequestArticles»

1.3. Результат зображено на рис. 2.7.

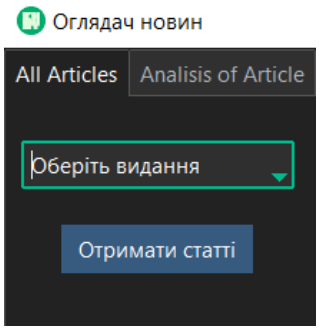

Рисунок 2.7 – Загальний вигляд вікна запиту новин в додатку

Джерело: розроблено автором

2. Вікно з результатом запиту статей:

2.1. Розглянемо реалізацію вікна для результату запиту статей (рис. 2.8).

```
# Створення вікна з отриманим результатом, списком останніх статей обраного видання
tb.Label(articleAllResponse, bootstyle = "success", text = "Останні статті видання").pack(pady=5, padx=10)

# Побудова таблиці з результатми отриманих статей
tableColumns = ["Видання", "Тема", "Посилання"]
resposeArticlesTable = tb.Treeview(articleAllResponse,  bootstyle = "success", columns = tableColumns, show = 'headings')
resposeArticlesTable.pack(fill=tb.BOTH, expand=True)
resposeArticlesTable.heading('Видання', text = 'Видання')
resposeArticlesTable.heading('Тема', text = 'Тема')
resposeArticlesTable.heading('Посилання', text = 'Посилання')
resposeArticlesTable.bind('<Double-Button-1>', CallArticleSummarise)
```

Рисунок 2.8 – Реалізація вікна з отриманням результату запиту

Джерело: розроблено автором

2.2. Опис реалізації:

2.2.1. Створення контейнеру «Label» з назвою розділу вікна «Останні статті видання».

2.2.2. Створення контейнеру «Treeview», тобто таблиці для даних запитуваних статей, в таблиці виділені стовпці «Видання», «Тема», «Посилання».

2.2.3. Створення події при подвійному натисканні на елемент події, що викликатиме функцію аналізу статті «CallArticleSummarise».

2.3.Результат об’єднання 2-х вікон зображений на рисунку 2.9.

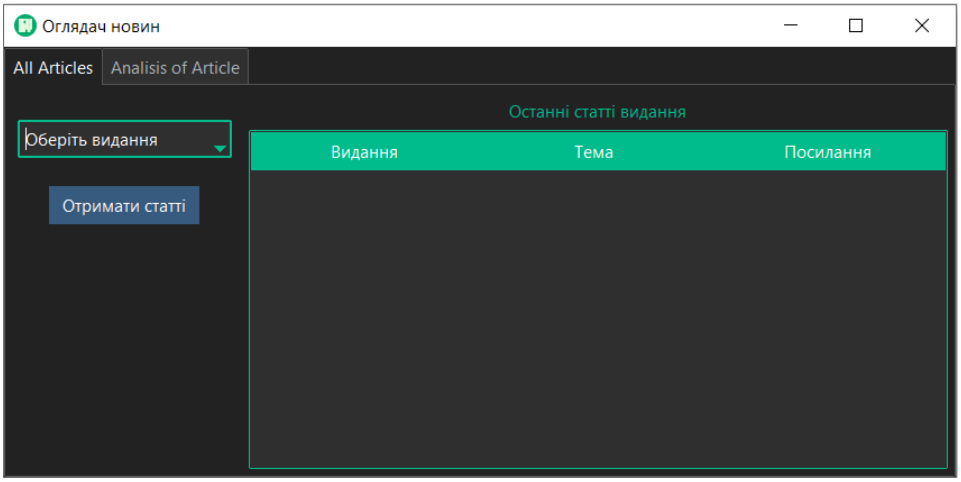

Рисунок 2.9 – Результат створення вікна з усіма статтями

Джерело: розроблено автором

3. Вікно з аналітикою по окремій статті:

3.1. Розглянемо реалізацію вікна з аналітикою по окремій статті (рис. 2.10).

```
# Створення заголовків для полів виводу результатів аналізу статті
sumarryTitle = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Основне з статті')
sumarryTitle.config(state = 'normal')
sumarryTitle.grid(row = 0, column = 0, pady = 10, padx = 2, columnspan = 3)

locationTitle = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Територіальні назви')
locationTitle.config(state = 'normal')
locationTitle.grid(row = 2, column=0)

organizationsTitle = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Назви організацій')
organizationsTitle.config(state = 'normal')
organizationsTitle.grid(row = 2, column=1)

propersTitle = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Фігуруючі імена')
propersTitle.config(state = 'normal')
propersTitle.grid(row = 2, column=2)

sentimentTitle = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Тональний графік речень статті')
sentimentTitle.config(state = 'normal')
sentimentTitle.grid(row = 0, column=3)

# Створення полів виводу результатів аналізу статті
sumarryText = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Оберіть статтю в розділі "All Articles"', wraplength = 1000,  padding=10)
sumarryText.grid(row = 1, column=0, pady=10, padx=10, columnspan= 3 )

locationsText = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Оберіть статтю в розділі "All Articles', wraplength = 333,  padding=10)
locationsText.grid(row = 3, column=0, pady=10, padx=10)

organizationsText = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Оберіть статтю в розділі "All Articles', wraplength = 333,  padding=10)
organizationsText.grid(row = 3, column=1, pady=10, padx=10)

propersText = tb.Label(articleAnalisis,  bootstyle = "inverse dark", text = 'Оберіть статтю в розділі "All Articles', wraplength = 333,  padding=10)
propersText.config(state = 'normal')
propersText.grid(row = 3, column=2, pady=10, padx=10)

notebook.add(articleAll, text = "All Articles")
notebook.add(articleAnalisis, text = "Analisis of Article")

mainWindow.mainloop()
```

Рисунок 2.10 – Реалізація вікна з аналізом статті

Джерело: розроблено автором

3.2. Опис реалізації:

3.2.1. Створено 5 заголовки полів для результатів аналізу типу «Label».

3.2.2. Створено 5 полів для результатів аналізу статті відповідно до заголовків.

3.3.Результат реалізації вікна з аналітикою по окремій статті зображено на рис. 2.11.

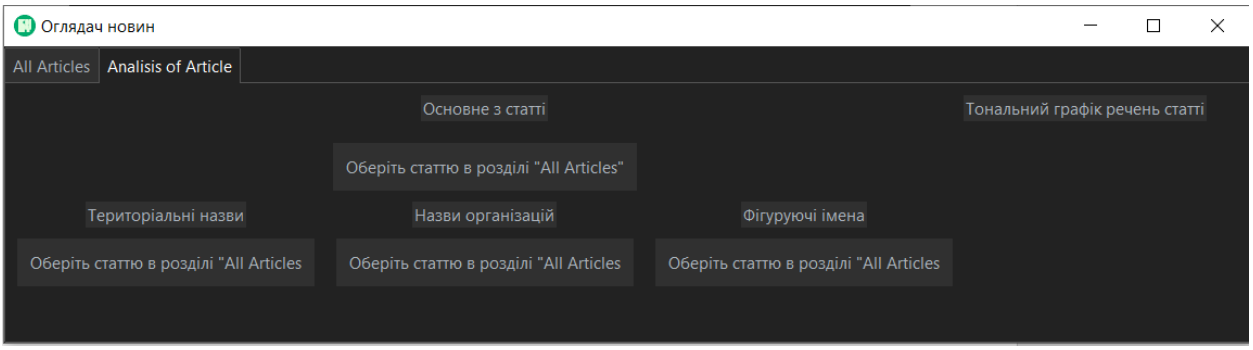

Рисунок 2.11 – Результат реалізації вікна з аналітикою для статті

Джерело: розроблено автором

2.3. Реалізація функціоналу UI додатку

Розгляд функціоналу додатку здійснюватиметься поділом функціоналу на основні функції, що його реалізують, та їхні основні допоміжні підфункції.

Ми виділили дві основні функції додатку:

1. Запит на отримання статей видання
2. Отримання тексту статті по виданню
3. Формування аналітики по окремій статті

1. Запит на отримання статей

1.1. Реалізація пункту отримання статей конкретного видання зображена на рис. 2.12.

```python
news_sources = {
    'Економічна правда': 'https://www.epravda.com.ua/news/',
    'УНІАН Економіка': 'https://www.unian.ua/detail/main_news/economics/'
}


def RequestArticles():
    # Видалення попереднього результату
    for item in resposeArticlesTable.get_children():
        resposeArticlesTable.delete(item)

    # Отримання типу видання
    issueName = issueNameInput.get()

    html_text = rq.get(news_sources[issueName]).text
    soup = bs(html_text, 'lxml')

    if issueNameInput.get() == 'Економічна правда':
        articlesList = CallIssue_EconomicTruth(soup)
    elif issueNameInput.get() == 'УНІАН Економіка':
        articlesList = CallIssue_UNIAN(soup)

    for article in articlesList:
        resposeArticlesTable.insert('', END, values = article)
```

Рисунок 2.12 – Реалізація функції RequestArticles для отримання статей

Джерело: розроблено автором

1.2. Опис реалізації:

1.2.1. Видалення вмісту таблиці з запитуваними статтями (якщо вони в ній наявні).

1.2.2. Присвоєння в змінну issueName значення, яке обране в випливаючому вікні назв запитуваних видань.

1.2.3. Запит GET по посиланню з словника для отримання актуальної HTML сторінки з останніми заголовками новин.

1.2.4. Парсинг HTML сторінки за допомогою бібліотеки BeautifulSoup.

1.2.5. Виклик функції парсингу кожного окремого видання, необхідність створювати особливі вимоги для парсингу кожного окремого видання виникає в том, що кожен сайт має власну особливу організацію сторінки та назви елементів на ній, через що умови для отримання даних необхідно прописувати для кожного видання окремо, в нашому випадку це видання «Економічна правда» та «УНІАН».

2. Отримання тексту статті по виданню.

2.1. Реалізація функції отримання списку останніх статей для видавництва «Економічна правда» зображена на рис. 2.13.

```
# Виклик парсингу та повернення списку статей видання "Економічна правда"
def CallIssue_EconomicTruth(soup):
    articlesList = []
    domen = "https://www.epravda.com.ua"
    articlesWithTags = soup.find_all('div', class_ = 'article__title')
    for article in articlesWithTags:
        hrefLink =  article.find('a').get('href')
        link = domen + hrefLink
        text = article.text
        articlesList.append((issueNameInput.get(), text, link))
    return articlesList
```

Рисунок 2.13 – Реалізація парсингу тексту статті видання "Економічна правда"

Джерело: розроблено автором

2.1.1. Пошук по HTML документу всіх тегів з назвою класу «article__title».

2.1.2. Формування списку, що складається з полів таблиці відповіді, тобто «Видання», «Тема», «Посилання».

2.1.3. Повернення списку в функцію запиту

2.2. Реалізація функції отримання списку останніх статей для видавництва «УНІАН Економіка» зображена на рис. 2.14.

```
# Виклик парсингу та повернення списку статей видання "УНІАН Економіка"
def CallIssue_UNIAN(soup):
    articlesList = []
    articlesWithTags = soup.find_all('div', class_ = 'list-thumbs__info')

    for article in articlesWithTags:
        link =  article.find('a').get('href')
        articleTopic = article.find('h3')
        text = articleTopic.text
        cleaned_text = text.replace('\n', '')
        articlesList.append((issueNameInput.get(), cleaned_text, link))
    return articlesList
```

Рисунок 2.14 – Реалізація парсингу тексту статті видання "УНІАН Економіка"

Джерело: розроблено автором

2.2.1. Пошук по HTML документу всіх тегів з назвою класу «list-thumbs__info».

2.2.2. Формування списку, що складається з полів таблиці відповіді, тобто «Видання», «Тема», «Посилання».

2.2.3. Повернення списку в функцію запиту

2.2.4. Заповнення таблиці результатів списком, в залежності від результату запитуваного видання.

2.3. Результат реалізації, при запиті на статті видання «Економічна правда» зображений на рис. 2.15.

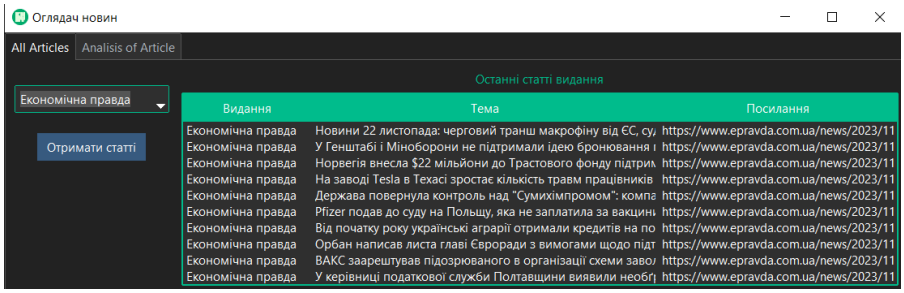

Рисунок 2.15 – Результат отримання всіх статей

Джерело: розроблено автором

Як бачимо, результат запиту повертає значення останніх розміщених виданням статей та розміщає дані відповідно у таблиці.

3. Функція отримання аналітики по окремій статті

3.1. Розглянемо реалізацію отримання аналітики по окремій статті (рис. 2.16).

Як вже зазначалося раніше, виклик даної функції викликається при подвійному кліку по окремій статті з загального списку, реалізація має наступний вигляд та вміщує в себе декілька різних функцій з різних файлів, для полегшення сприйняття коду.

```
def CallArticleSummarise(event):
    selected_item = resposeArticlesTable.selection()
    if selected_item:
        values = resposeArticlesTable.item(selected_item, 'values')
    link = values[2]
    html_text = rq.get(link).text
    soup = bs(html_text, 'lxml')

    if values[0] == 'Економічна правда':
        setencesAsList = CallSummarise_EconomicTruth(soup)
    elif values[0] == 'УНІАН Економіка':
        setencesAsList = CallSummarise_UNIAN(soup)

    full_text = ' '.join(setencesAsList)
    sumarryText.config(text = get_article_summary(full_text))
    locationsText.config(text = extract_location_names(full_text))
    organizationsText.config(text = extract_organization_names(full_text))
    propersText.config(text = extract_proper_names(full_text))

    scoresOfSentences =[]
    for sentence in setencesAsList:
        score = analyze_sentiment(sentence)
        scoresOfSentences.append(score)
    data = {"sentences": setencesAsList,
            "scores": scoresOfSentences}
    df = pd.DataFrame(data)

    fig, ax = plt.subplots(figsize=(5, 4))
    df['scores'].value_counts().sort_index().plot(kind='bar', color=['red', 'green'], ax=ax)
    ax.set_title('Гістограма тональності речень')
    ax.set_xlabel('Тональність')
    ax.set_ylabel('Кількість речень')

    canvas = FigureCanvasTkAgg(fig, master=articleAnalisis)
    canvas.draw()
    canvas.get_tk_widget().grid(row = 1, column=3, rowspan= 3)
    notebook.select(1)
```

Рисунок 2.16 – Реалізація функції виклику аналізу статті

Джерело: розроблено автором

3.2. Опис реалізації:

3.2.1. Отримуємо інформацію про обраний користувачем об’єкт (статтю), ініційований подвійним кліком.

3.2.2. Якщо обраний об’єкт існує – присвоюємо інформацію з нього в змінну values.

3.2.3. Отримуємо посилання на статтю з масиву value по індексу [2].

3.2.4. Робимо GET запит на отримання інформації про статтю по отриманому посиланню.

3.2.5. Оскільки для кожної статті своє власне планування HTML сторінки, диференціюємо вибір на окремі функції парсингу для кожного видання.

3.3. Реалізація отримання тексту статті видання «Економічна правда» зображена на рис. 2.17.

```
def CallSummarise_EconomicTruth(soup):
    articleText = soup.find('div', class_ = 'post__text').find_all('p')
    setencesAsList = []
    for article in articleText:
        article =  article.text
        setencesAsList.append(article)
    return setencesAsList
```

Рисунок 2.17 – Реалізація отримання тексту статті з «Економічна правда»

Джерело: розроблено автором

3.3.1. Пошук по HTML документу всіх тегів з назвою класу «post__text».

3.3.2. Формування списку речень статті, при отриманні тексту з отриманих тегів.

3.3.3. Повернення списку отриманих речень.

3.4. Реалізація отримання тексту статті видання «УНІАН Економіка» зображена на рис. 2.18.

```
def CallSummarise_UNIAN(soup):
    articlesWithTags = soup.find('div', class_ = 'article-text').find_all('p')
    setencesAsList = []
    for article in articlesWithTags:
        article =  article.text
        setencesAsList.append(article)
    return setencesAsList
```

Рисунок 2.18 – Реалізація отримання тексту статті з «УНІАН Економіка»

Джерело: розроблено автором

3.4.1. Пошук по HTML документу всіх тегів з назвою класу «article-text».

3.4.2. Формування списку речень статті, при отриманні тексту з отриманих тегів.

3.4.3. Повернення списку отриманих речень.

3.4.4. Формування повного тексту шляхом об’єднання всіх речень в одну строкову змінну.

3.5. Функції для аналізу природної мови (nlp):

3.5.1. Реалізація функції узагальнення тексту зображена на рис. 2.19.

```python
import spacy
from collections import Counter

nlp = spacy.load("uk_core_news_sm")

def get_article_summary(article_text, num_sentences = 1):
    doc = nlp(article_text)
    sentences = [sent.text for sent in doc.sents]

    # Підрахунок частоти слів для оцінки речень
    word_freq = Counter()
    for word in doc:
        word_freq[word.text.lower()] += 1

    # Оцінка речень за частотою слів
    sentence_scores = {sent: sum([word_freq[word.text.lower()] for word in nlp(sent)]) for sent in sentences}

    # Отримання топ N реень за частотою вживання слів
    summary_sentences = sorted(sentence_scores, key=sentence_scores.get, reverse = True)[:num_sentences]

    # Форматування в змінну str
    summary = ' '.join(summary_sentences)
    return summary
```

Рисунок 2.19 – Реалізація узагальнення статті

Джерело: розроблено автором

3.5.2. Імпортуємо необхідні бібліотеки.

3.5.3. Імплементуємо алгоритм роботи з природною мовою nlp.

3.5.4. Передаємо в функцію для узагальнення текст статті, яка потребує узагальнення та кількість речень, яким вона має бути узагальнена, в даному випадку 1.

3.5.5. За допомогою функції Counter() підраховуємо частоту появи окремих слів в речень, щоб зрозуміти їхню кількісну значущість.

3.5.6. Створюємо словник з ключем у вигляді речень, а значенням у вигляді їхньої значущості по частоті слів.

3.5.7. Формуємо остаточний цілісний текст (на випадок, якщо узагальнення описується декількома, а не одним реченням).

3.5.8. Повертаємо текст з функції.

3.6. Функції виводу згадуваних в тексті організацій, географічних назв, осіб, підхід до визначення назв однаковий для всіх функцій зображені на рис. 2.20.

```
def extract_location_names(text):
    doc = nlp(text)
    locationNames = []
    for ent in doc.ents:
        if ent.label_ == "LOC":
            locationNames.append(ent.text)
    return count_elements(locationNames)

def extract_organization_names(text):
    doc = nlp(text)
    organizationNames = []
    for ent in doc.ents:
        if ent.label_ == "ORG":
            organizationNames.append(ent.text)
    return count_elements(organizationNames)

def extract_proper_names(text):
    doc = nlp(text)
    properNames = []
    for ent in doc.ents:
        if ent.label_ == "PER":
            properNames.append(ent.text)
    return count_elements(properNames)
```

Рисунок 2.20 – Функції організацій, географічних назв, осіб

Джерело: розроблено автором

3.6.1. Проводимо навчання на тексті, з якого необхідно дістати слова, що відповідають ознакам організацій, для кожного слова навчений текст визначає entities, для географічних назв – «LOC», для організацій та власних назв (осіб) – «ORG» та «PER» відповідно.

3.6.2. Формуємо список з таких слів для кожної функції.

3.6.3. Для того щоб слова не повторювалися, додаємо функцію, яка групуватиме однакові слова та повертатиме кількість повторень кожного окремого слова (рис. 2.21).

```python
def count_elements(lst):
    element_counts = Counter(lst)
    result = list(element_counts.items())
    result_str = ", ".join(f"{element}: {count}" for element, count in result)
    return result_str
```

Рисунок 2.21 – Функція групування однакових слів

Джерело: розроблено автором

3.6.4. Використовуємо функцію Counter() для підрахунку значень списку, що в нього передається.

3.6.5. Формуємо однин рядок з виводом значень та їхньої кількості.

3.6.6. Повернення рядка з підрахунками з функції.

3.7. Реалізація «настрою» речень статті.

Для аналізу «настрою» речень використовуватимемо навчений класифікатор Наївного Баєса, розглянемо його реалізацію та навчання на рис. 2.22.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
import joblib

df = pd.read_csv('sentiment_ua.csv',  sep=';')
# Навч та тест вибірки
X_train, X_test, y_train, y_test = train_test_split(df['word'], df['pos_neg'], test_size=0.2, random_state=42)

# Bag of Words
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# НБ класифікатор
ua_sentiment_model = MultinomialNB()
ua_sentiment_model.fit(X_train_counts, y_train)

# Аналіз на тестах
y_pred = ua_sentiment_model.predict(X_test_counts)

# Оцінки
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)

joblib.dump(vectorizer, 'vectorizer.pkl')
joblib.dump(ua_sentiment_model, 'ua_sentiment_model.pkl')
```

Рисунок 2.22 – Побудова Наївного Баєсового класифікатора

Джерело: розроблено автором

3.7.1. Для навчання класифікатора Наївного баєса імпортується датасет для класифікації, датасет складається з українських слів і відповідними до них значеннями «1» – для позитивних слів, «-1» – для негативних слів [24].

3.7.2. Здійснюється поділ на навчальну та тестову вибірку у відповідності 80% даних на 20%.

3.7.3. Використовуємо алгоритм «Bag of words», необхідний для числового представлення текстів для тренувального та тестового наборів відповідно, яке використовуватиметься для навчання та оцінки моделей машинного навчання.

3.7.4. Здійснюємо навчання моделі Наївного Баєса.

3.7.5. Здійснюємо оцінку навченої моделі, та виводимо результати, як наслідок маємо результат оцінки Accuracy рівний 67% (рис.2.23), що є досить задовільним.

```
Accuracy: 0.668695652173913
Classification Report:
              precision    recall  f1-score   support

          -1       0.67      1.00      0.80       760
           1       0.80      0.03      0.06       390

    accuracy                           0.67      1150
   macro avg       0.73      0.51      0.43      1150
weighted avg       0.71      0.67      0.55      1150
```

Рисунок 2.23 – Метрики ефективності моделі

Джерело: розроблено автором

3.7.6. Після навчання моделі маємо змогу експортувати навчену модель та векторайзер, для подальшого використання на нових даних. Після того, як навчена модель та векторайзер експортовані готові до використання, ми маємо змогу імпортувати їх та використовувати для класифікації речень (рис. 2.24).

```python
import joblib

ua_sentiment_model = joblib.load('ua_sentiment_model.pkl')
vectorizer = joblib.load('vectorizer.pkl')

def analyze_sentiment(text):
    text_counts = vectorizer.transform([text])
    sentiment_score = ua_sentiment_model.predict(text_counts)[0]

    return sentiment_score
```

Рисунок 2.24 – Імпорт навчених моделей для визначення "настрою" речення

Джерело: розроблено автором

Після того, як для статті були виконані всі функції і результат оцінки «настрою» для кожного речення, можемо використати бібліотек matplotlib для графічної інтерпретації результату та здійснити автоматичне перенесення користувача на вікно за аналітикою, тобто notebook.select[1].

Результат аналізу статті, при виборі випадкової статті із загального списку запитуваних статей:

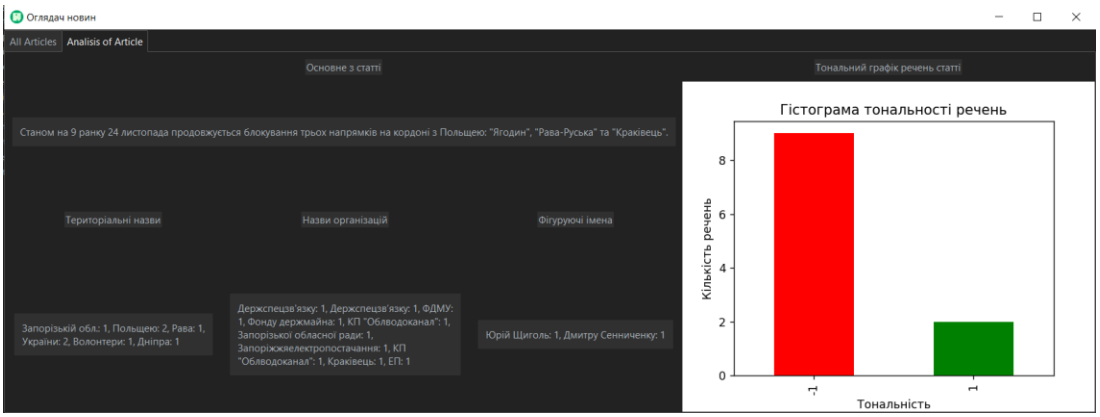

Рисунок 2.25 – Результат аналізу випадкової статті зі списку всіх статей

Джерело: розроблено автором

Після вибору випадкової статті зі списку маємо наступний результат (рис. 2.25), як бачимо, результатом є поле з гістограмою, що описує кількість «позитивних» та «негативних» речень у тексті, а також заповнені 4-и текстових поля: короткий опис статті за результатом функції з розрахунку «частоти» слів в тексті, назви згадуваних власних назв, що описують території, організації, осіб.

ВИСНОВКИ

В процесі виконання даної роботи детально вивчені та апробовані методи та алгоритми машинного та глибокого навчання для прискореного аналізу, обробки та узагальнення україномовного медіа контенту.

Визначено сутність поняття контент-аналіз, який представляє якісно-кількісний метод вивчення документів, що характеризується об'єктивністю висновків і строгістю процедури та полягає у кількісному вираженні обробленого тексту з подальшою інтерпретацією результатів, проаналізовано стан та умови розвитку автоматичного збирання даних з вебресурсів. Проведений аналіз демонструє стрімкий розвиток методу вебскрейпінгу в сучасних сферах, таких як електронна комерція, нерухомість, набір персоналу, тощо. Дана техніка використовується для автоматичного аналізу вебсайтів для збору інформації про ціни на товари, рейтинги продуктів, новини або будь-яку іншу доступну інформацію. Він може бути корисним для збору даних для досліджень, аналізу конкурентів, відстеження змін у вмісті вебсайтів та інших задач в режимі реального часу, а також для бізнес-аналітики, маркетингу, аналізу соціальних мереж та в наукових дослідженнях.

Досліджено сучасні методи переведення текстової інформації в зручну для машинного та глибокого навчання форму та методи, якими це навчання може бути реалізовано, зокрема: «Bag of words», «Term Frequency–Inverse Document Frequency», «Doc2Vec», «N-gram».

Описана бібліотека для роботи з природною мовою spaCy, яка використовує такі методи як векторизацію, класифікацію з CNN та Наївний Баєсівський класифікатор.

Використовуючи новітні бібліотеки продемонстровано створення візуального користувацького інтерфейсу та поєднання в ньому найрізноманітнішого функціоналу. Розробка додатку та кожного логічного

елементу додатку була реалізована в послідовній та плановій структурі відповідно до логічного виконання дій користувача, включаючи три основні кроки:

1.Реалізація елементу у вигляді коду на Python.

2. Опис реалізації.

3. Результат реалізації.

Результатом додатка є графічна інтерпретація результату, а саме вікно аналізу випадкової статті зі списку всіх статей, що відображає аналітику статті та її так званий «настрій».

Розвиток в сфері автоматичної обробки інформації може значно, якщо не повністю, позбавити рутинної роботи працівників сфери журналістики чи ручної обробки текстової інформації та виділити час на задачі, які дійсно потребують важливих рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Костенко Н.В., Іванов В.Ф. Досвід контент-аналізу: Моделі та практики: Монографія. 2003. URL: https://www.aup.com.ua/wp-content/uploads/2016/03/Dosvid-kontent-analizu.-Modeli-ta-praktiki.pdf (дата звернення 14.11.23)
2. Statcounter Global Stats. StatCounter. URL: https://gs.statcounter.com/ (accessed: 16.11.23)
3. Dovetail Editorial. Team What is content analysis? March 20, 2023 URL: https://dovetail.com/research/content-analysis/ (accessed: 16.11.23)
4. W3Schools. What is the HTML DOM? URL: https://www.w3schools.com/whatis/whatis_htmldom.asp (accessed: 17.11.23)
5. Andrew Crider. Scrape Web Articles With Python. URL: https://medium.com/geekculture/automating-web-scraping-articles-with-python-49dce6714517 (accessed: 17.11.23)
6. Eleanor Ajala. What is Web Scraping? Aug 21, 2018 URL: https://www.imperva.com/blog/archive/the-economics-of-web-scraping-report/ (accessed: 18.11.23)
7. Srishti Saha. The Economy of the Web Scraping Industry. Feb 12, 2021 URL: https://www.blog.datahut.co/post/the-economy-of-the-web-scraping-industry (accessed: 17.11.23)
8. Research Nester. Web Scrapping Software Market. URL: https://www.researchnester.com/reports/web-scraping-software-market/5041 (accessed: 17.11.23)
9. https://blog.apify.com . Natasha Lekh, Petr Pátek. What's the future of web scraping in 2023? Jan 20, 2023 URL: https://blog.apify.com/future-of-web-scraping-in-2023/ (accessed: 18.11.23)

10. Fatih Karabiber. TF-IDF — Term Frequency-Inverse Document Frequency. URL: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/ (accessed: 18.11.23)
11. Manish Nayak. An Intuitive Introduction to Document Vector(Doc2Vec). 24 Jun, 2019. URL: https://pub.towardsai.net/an-intuitive-introduction-of-document-vector-doc2vec-42c6205ca5a2 (accessed: 22.11.23)
12. Everton Gomede. Exploring N-gram Models in Natural Language Processing. Aug 19 2023. URL: https://medium.com/@evertongomede/exploring-n-gram-models-in-natural-language-processing-bf5852b32050 (accessed: 20.11.23)
13. Think IBM. What are convolutional neural networks? URL: https://www.ibm.com/topics/convolutional-neural-networks (accessed: 21.11.23)
14. Mayank Mishra. Convolutional Neural Networks, Explained URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 (accessed: 23.11.23)
15. Harry Zhang. The Optimality of Naive Bayes 2004. URL: https://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf (accessed: 23.11.23)
16. Debasish Kalita. A Brief Overview of Recurrent Neural Networks (RNN) Nov 7th, 2023. URL: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/ (accessed: 23.11.23)
17. Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. May 21, 2015. URL: https://datascience.stackexchange.com/questions/33423/types-of-recurrent-neural-networks (accessed: 23.11.23)
18. Vijay Choubey. Understanding Recurrent Neural Network (RNN) and Long Short Term Memory(LSTM). Jul 23, 2020. URL: https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d (accessed: 23.11.23)
19. Archit Saxena. Introduction to Long Short-Term Memory (LSTM) 17 Jan 2023. URL: https://medium.com/analytics-vidhya/introduction-to-long-short-term-memory-lstm-a8052cd0d4cd (accessed: 24.11.23)

20. spaCy. spaCy 101: Everything you need to know. URL: https://spacy.io/usage/spacy-101#serialization (accessed: 24.11.23)
21. Pew Research Center. News Platform Fact Sheet. Nov 15 2023. URL: https://www.pewresearch.org/journalism/fact-sheet/news-platform-fact-sheet/ (accessed: 18.11.23)
22. О. В. Тарасов, В. М. Федорченко, М. Ю. Лосєв. Основи програмування та алгоритмічні мови. Харків. 2010. 92 с.
23. А. В. Яковенко. Основи програмування. Python. Частина 1. Київ. 2018. 195 с. URL: https://ela.kpi.ua/bitstream/123456789/25111/1/Python.pdf (accessed: 18.11.23)
24. Serhii Kupriienko. Ukrainian-Sentiment-Analysis. Sep 5, 2021 URL: https://github.com/skupriienko/Ukrainian-Sentiment-Analysis/blob/main/sentiment_ua.csv (accessed: 11.11.23)
25. О. В. Швець, О. М. Флюр. Контент-аналіз. 2004. 760с.
26. Костенко Н. В. Масова комунікація. К.: Українська енциклопедія, 1999. — С.153-181.
27. Muhammad Hassan. Content Analysis – Methods, Types and Examples. Nov 2, 2023. URL: https://researchmethod.net/content-analysis/ (accessed: 20.11.23)
28. Kajal Yadav. Scraping 1000’s of News Articles using 10 simple steps. URL: https://towardsdatascience.com/scraping-1000s-of-news-articles-using-10-simple-steps-d57636a49755 (accessed: 19.11.23)
29. Tsaone Swaabow ThapeloEmail, Molaletsa Namoshe, Oduetse Matsebe, Tshiamo Motshegwa, Mary-Jane Morongwa Bopape. SASSCAL WebSAPI: A Web Scraping Application Programming Interface to Support Access to SASSCAL’s Weather Data. URL: https://datascience.codata.org/articles/10.5334/dsj-2021-024 (accessed: 19.11.23)
30. Cem Dilmegani. Top 5 Web Scraping Analytics Use Cases with Examples in 2023. Oct 10, 2023. URL: https://research.aimultiple.com/web-scraping-analytics/ (accessed: 20.11.23)

31. Sandro Shubladze. Forbes. Web Scraping: What It Is And How Companies Can Leverage It. Jan 3, 2023. URL: https://oxfordre.com/economics/display/10.1093/acrefore/9780190625979.001.0001/acrefore-9780190625979-e-652 (accessed: 20.11.23)
32. Bao Tram Duong, Natural Language Processing (NLP) with Deep Learning Models (RNN & CNN). URL: https://medium.com/mlearning-ai/natural-language-processing-nlp-with-deep-learning-models-rnn-cnn-coleridge-initiative-928f8d003b6d (accessed: 21.11.23)
33. Johannes Ernesti, Peter Kaiser. The Comprehensive Guide to Hands-On Python Programming. URL: https://openbook.rheinwerk-verlag.de/python/39_002.html (accessed: 11.11.23)
34. Souman Roy. Math, Stats and NLP for Machine Learning: As Fast As Possible. Feb 9, 2018. URL: https://medium.com/meta-design-ideas/math-stats-and-nlp-for-machine-learning-as-fast-as-possible-915ef47ced5f (accessed: 09.11.23)
35. Yannis Haralambous. Text Mining Methods Applied to Mathematical Texts. 30 Aug 2018 URL: https://hal.science/hal-01864536/document (accessed: 11.11.23)
36. Deborah Ferreira. Mathematical language processing: deep learning representations and inference over mathematical text. University of manchester. 2022. URL: https://pure.manchester.ac.uk/ws/portalfiles/portal/224501052/FULL_TEXT.PDF (accessed: 12.11.23)
37. Bakman Y.. Robust understanding of word problems with extraneous information. Accessed 31 Mar 2008. URL: http://aps.arxiv.org/abs/math.GM/070139 (accessed: 11.11.23)
38. Delip Rao. Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning 1st Edition 2018.
39. Megan Binder. How to Use Neuro-linguistic Programming (NLP) Techniques May 29, 2023 URL: https://www.thinkific.com/blog/neuro-linguistic-programming-nlp/ (accessed: 10.11.23)

40. Introduction to NLP Library: Spacy in Python. Uzair Adamjee Jun 24, 2020. URL: https://medium.com/analytics-vidhya/introduction-to-nlp-library-spacy-in-python-a98cf344eb6d (accessed: 09.11.23)
41. Arunachalam B. NLP using spaCy – How to Get Started with Natural Language Processing. JUNE 26, 2023. URL: https://www.freecodecamp.org/news/getting-started-with-nlp-using-spacy/ (accessed: 12.11.23)
42. Kenneth Benoit and Akitaka Matsuo. A Guide to Using spacyy. Mar 04 2020. URL: https://cran.r-project.org/web/packages/spacyr/vignettes/using_spacyr.html (accessed: 12.11.23)
43. Lester James Miranda. Multi hash embeddings in spaCy. 19 Dec 2022. URL: https://arxiv.org/pdf/2212.09255.pdf (accessed: 26.11.23)
44. Paco Nathan. Natural Language Processing in Python using spaCy: An Introduction. SEP 10 2019. URL: https://domino.ai/blog/natural-language-in-python-using-spacy (accessed: 27.11.23)
45. Swaathi Kakarla. Natural Language Processing: NLTK vs spaCy. October 17, 2019. URL: https://www.activestate.com/blog/natural-language-processing-nltk-vs-spacy/ (accessed: 27.11.23)
46. Yuli Vasiliev. Natural Language Processing with Python and spaCy. Apr 2020. 216 pages.
47. Alan D. Moore. Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter. May 15, 2018. 452 pages.
48. Hilman Ramadhan. Web Scraping with Javascript and Nodejs (2023 Guide). Oct 25, 2023. URL: https://serpapi.com/blog/web-scraping-in-javascript-complete-tutorial-for-beginner/ (accessed: 07.11.23)
49. John Grayson. Python and Tkinter Programming. January 1, 2000. 688 pages.
50. Kevin Sahin. Web Scraping With JavaScript And NodeJS. Aug 02 2022. URL: https://www.scrapingbee.com/blog/web-scraping-javascript/ (accessed: 07.11.23)
51. Ryan MitchellRyan Mitchell. Web Scraping with Python: Collecting More Data from the Modern Web 2nd Edition. May 8, 2018. 306 pages

52. Michael Heydt. Python Web Scraping Cookbook: Over 90 proven recipes to get you scraping with Python, micro services, Docker and AWS. February 9, 2018. 364 pages