КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

КЛАСФИКАЦІЯ І КЛАСТЕРИЗІЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ

Виконав: студент 4-го курсу
Ігор ГАВРЕШ
(підпис)

Науковий керівник:
професор кафедри теоретичної кібернетики
доктор фіз.-мат. наук, професор
Юрій КРАК
_____
(підпис)

Засвідчую, що в цій роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теоретичної кібернетики
«_1_» ___Червня_____ 2022 р.,
протокол № _11_
Завідувач кафедри
доктор фіз.-мат. наук, професор
Юрій КРАК
(підпис)

КИЇВ-2022

РЕФЕРАТ

Обсяг роботи 40 сторінок, 22 ілюстрації, 1 таблиця, 24 джерел посилань.

ТЕКСТОВА АНАЛІТИКА, ОБРОБКА ПРИРОДНЬОЇ МОВИ, КЛАСИФІКАЦІЯ, КЛАСТЕРИЗАЦІЯ, АНАЛІЗ ТОНАЛЬНОСТІ, МАШИННЕ НАВЧАННЯ.

Об’єктом роботи є класифікація та кластеризація текстової інформації природньою мовою. Предметом роботи є програмні додатки кластеризації текстової інформації та оцінки тональності (тернарної класифікації).

Метою роботи є створення програмних засобів кластеризації та класифікації із використанням мови програмування Python та бібліотек машинного навчання, що можуть бути використані як в навчальних, так й в комерційних цілях.

Методи розроблення: комп’ютерна обробка текстів із використанням алгоритмів роботи з натуральною мовою (NLP), векторні та матричні математичні моделі текстів, математичні моделі ієрархічної та неієрархічної кластеризації, базові математичні моделі класифікації без використання ансамблевих моделей. Інструменти розроблення: комерційна версія інтегровано середовища розробки (IDE) PyCharm, менеджер середовища Anaconda (версія personal), мова програмування Python 3.10, бібліотеки машинного навчання Scikit learn, Pandas, обробки природньої мови SciPy, NLTK.

Результати роботи: створено загальний огляд методів та напрямків використання текстової аналітики, розглянуто базовий алгоритм текстової аналітики. Розглянуті базові моделі класифікації та кластеризації декількох типів, а також критерії (метрики) оцінки якості застосування таких моделей.

Розроблено програмні продукти для кластеризація корпусу текстів та оцінки тональності текстів. Програмні продукти можуть бути використані як в навчальних, так я в комерційних цілях. При розробці було використано виключно ресурси з відкритим кодом та вільною ліцензією.

ЗМІСТ

ВСТУП

Поточне сторіччя характеризується суттєвим збільшенням обчислюваних потужностей комп’ютерної техніки й, як один з наслідок, кількістю даних що збираються та накопичуються.

Термін “Big data” було запроваджено редактором журналу Nature Кліффордом Лінчем[1] у 2008 році у спецвипуску, присвяченому вибуховому зростанню світових обсягів інформації. На думку редактора, до великих даних слід відносити потоки даних понад 100 Гб на день.

Спочатку під цим простим терміном були приховані лише два поняття – зберігання та обробка даних. Під обробкою даних розуміємо різні методи, моделі, підходи, інструменти обробки даних. Пізніше BigData переросла в соціально-економічний феномен, пов'язаний з появою нових технологічних можливостей аналізу великого обсягу даних.

Суттєва частина big data накопичується в текстовому форматі, тому виникають потреби текстової аналітики – глибокого інтелектуального аналізу даних, що побудований на алгоритмах машинного навчання, нейромережах, тощо.

Проблема полягає в тому, що згадані моделі у переважній більшості створювалися для обробки та моделювання нумеричних наборів даних. Тому найчастіше для застосування математичних моделей до корпусу текстів застосовуються моделі, що параметризують розглядуваний корпус текстів, ставлячи у відповідність кожному його елементу деякий числовий набір.

Процес такої параметризації, як правило, складається іх препроцесінгу текстів корпусу та безпосередньо параметризації [2].

В першому розділі роботи розглядаються теоретичні основи та види текстової аналітики, її призначення, а також побудова конвеєрів обробки даних у додатках текстової аналітики. Типовий конвеєр складається з трьох частин: препроцесінгу текстів корпусу, параметризації текстового корпусу та

застосування потрібних математичних моделей до числової багатовимірної множини, отриманої в результаті параметризації.

В розділі розглядаються як найбільш застосовувані алгоритми препроцесінгу та їх комбінації, так й усі основні моделі параметризації текстового корпусу.

Щодо математичних моделей машинного навчання та оцінки їх ефективності, розглянуто найбільш часто застосовувані в текстовій аналітиці базові моделі. На практиці, як правило, використовуються більш складні моделі, а ще частіше – багаторівневі ансамблеві моделі, але це виходить за рамки розгляду в роботі.

Другий розділ присвячений постановці задачі та проектуванню додатку на базі теоретичного розгляду, зробленого в розділі 1. Третій розділ є описом інструментів розробки та поясненням програмної реалізації спроектованих додатків.

Результатом роботи є спроектовані та розроблені додатки, що використовуються для кластеризації текстів та класифікації текстів за емоційним забарвленням.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.

1.1. Огляд текстової аналітики.

З приходом ери великих даних, традиційні методи аналізу текстів вручну з метою виявлення основних тем та тенденцій у даних виявилися неефективними. Перш за все, зниження ефективності нижче критичного рівня зв’язано з кількістю даних, що потрібно обробити, для отримання якісних аналітичних результатів: вручну проаналізувати тисячі записів в розумні терміни та з незахмарною собівартістю просто неможливо. Програмні інструменти текстової аналітики дозволяють автоматизувати цей процес та підвищити його ефективність.

Текстова аналітика, яку часто називають глибоким чи інтелектуальним аналізом тексту - це автоматизований процес вилучення важливої інформації з неструктурованих текстових даних, під час якого застосовуються методи з різних галузей знання, включаючи комп'ютерну лінгвістику (NLP), інформаційний пошук і статистику[3]. Текстова аналітика застосовується як засіб інтелектуальної обробки економічних даних, так й для наукових досліджень. Фахівці з аналізу даних використовують інструменти текстової аналітики для обробки результатів опитувань клієнтів, даних торгових автоматів, записів call-центрів, медичних книжок пацієнтів, результатів промислових наукових досліджень, юридичної документації, активності у соціальних мережах, тощо.

Використовуючи алгоритми обробки природної мови та статистичні інструменти, текстова аналітика дозволяє вирішувати такі завдання, як класифікація текстів, аналіз тональності, розпізнавання іменованих сутностей та визначення відносин. У ході виконання цих завдань суттєва інформація витягується зі складних неструктурованих текстів великого обсягу, які таким чином перетворюються на структуровані дані. Це дозволяє компаніям резюмувати відгуки про свою продукцію та послуги, пов'язувати конкретні симптоми з ефективністю обраної тактики лікування захворювання, та навіть

використовувати алгоритми машинного навчання для виявлення нових тенденцій у різних галузях виробництва та результатах маркетингових компаній.

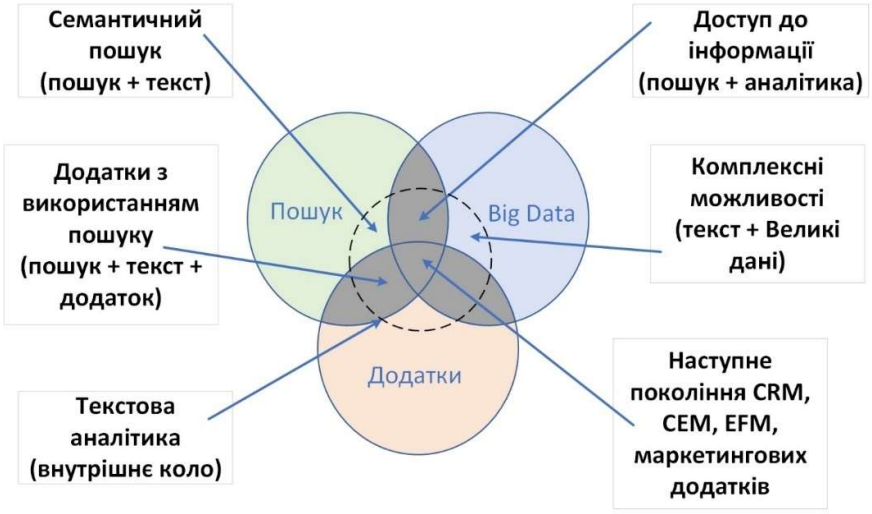


Рисунок 1.1 – Структурне зображення текстової аналітики.

Структурування подібних даних дозволяє аналітикам швидко резюмувати та візуалізувати тенденції в даних, що в свою чергу веде до кращого розуміння самих даних, прийняття інформованих бізнес-рішень та нових наукових відкриттів.

Серед багатьох застосувань текстової аналітики можна виділити такі:

– визначення, виявлення фактів;
– кодування документів
– кластеризація документів
– класифікація документів
– аналіз тональності
– анотування текстів (створення резюме документів)

В наступному підрозділі дамо короткі визначення кожному з перелічених застосувань.

1.2. Основні напрямки застосування текстової аналітики.

Визначення (extraction) фактів. У ході аналізу текстових документів дослідникові часто доводиться витягувати і структурувати цікаві для нього

факти. Наприклад, компанії необхідно витягти із стрічки новин усі факти, що стосуються її конкурентів — відносини з іншими компаніями, технології, продукція, бюджет на операції з придбання компаній, ключові співробітники, тощо. Інший приклад: фармацевтична компанія має намір вивчити записи в медичних книжках пацієнтів та витягти з них всі факти, що описують використання медичних препаратів її виробництва: показання до застосування, дозування, тривалість прийому, спосіб застосування, можливі побічні ефекти.

Кодування документів. Працюючи з текстовими документами часто виникає необхідність виявлення у них спеціальних текстових шаблонів, які співвідносяться із заздалегідь заданими категоріями. Цю операцію прийнято називати "кодуванням документів". Вона часто використовується при аналізі записів розмов call-центрів, даних опитувань, коментарів, оглядів, тощо. Система об'єднує в одну категорію текстові записи, в яких виражена та сама ідея, навіть незважаючи на те, що використовуються різні лексичні одиниці. В основі інструментів, які дозволяють кодувати документи в автоматичному режимі, лежить машинне навчання або комбінація технологій лінгвістичного та семантичного аналізу із розпізнаванням текстових шаблонів.

Кластеризація текстів. Досить часто, коли перед користувачем стоїть завдання організації великої кількості документів, він не має жодного уявлення ані про зміст, ані про структуру цих текстів. У разі на початковому етапі аналізу зазвичай виконується кластеризація текстів, коли він документи розподіляються по групам залежно від своїх загальної тематики.

Класифікація текстів. У результаті класифікації документи розподіляються по групам залежно від основний теми. Основна тема виділяється із застосуванням методів семантичного аналізу або машинного навчання. Наприклад, деяка компанія отримує тисячі запитів від клієнтів до служби підтримки щодня. Кожен із цих запитів має бути адресований одному із спеціалістів служби підтримки. Це повинен бути спеціаліст, що найкраще розбирається саме в потрібній тематиці. Запит, що потребує знань з налаштування телевізора не повинен бути адресований спеціалісту з ремонту

холодильного обладнання. Для того, щоб вирішити, якому з співробітників служби підтримки повинен бути направлений запит необхідно автоматично визначити його тему, основні потрібні компетенції фахівця підтримки, та переадресувати запит саме такому фахівцю.

Аналіз тональності текстів. Точна оцінка тональності текстів, що описують реакцію клієнтів компанії на ту чи іншу подію, бренд чи продукцію, дозволяє компанії приймати поінформовані рішення. Класифікувати тональності, які у тому чи іншому документі, легко, якщо автор відгуку чи коментаря прямо висловлює своє ставлення до предмета. Однак у текстах часто зустрічаються твердження, які на перший погляд містять лише фактичну інформацію. Наприклад, в одному з відгуків про готель хтось із її постояльців написав таке: "У душі не було води". Хоча твердження не містить негативних одиниць тональності, ми розуміємо, що йдеться про негативну оцінку готелю.

Ще одним частим застосуванням оцінки тональності є аналіз новин та коментарів до них на сайтах новин та у соціальних мережах.

Анотація текстів. Можливість ознайомитися з коротким змістом, перш ніж читати тексти цілком, значно підвищує швидкість та ефективність роботи аналітиків. Традиційно завдання щодо складання коротких анотацій доручали співробітникам компанії, що займаються аналізом даних. Проте сьогодні, в умовах зростаючих обсягів та кількості текстових документів, які необхідно проаналізувати, анотування текстів співробітникам не є доцільним. Замість цього застосовуються алгоритми, що дозволяють отримати з кожного тексту вибірки ключові слова та тези. Це також значно полегшує роботу з великою кількістю документів.

1.3. Типова послідовність дій при обробці текстових документів.

Більшість із розглянутих вище задач текстової аналітики розпадається на декілька підзадач – кроків, серед яких можна виділити:

а) попередню обробку документів;

b) технічну задачу перетворення корпусу текстів на матричну, векторну або одну з інших моделей;

c) виконання задачі класифікації, кластеризації, аналізу тональності, тощо;

d) інтерпретація результатів.

Кроки а)-с) можна вважати майже стандартними, а останній з означених кроків суттєво залежить від предметної області. В подальшому розгляді сфокусуємось на перших трьох кроках текстової аналітики.

Перший крок – попередню обробку – графічно представлено на рисунку 1.2.

Рисунок 1.2 – Алгоритм перетворення текстів корпусу, що розглядається.

Попередня обробка тексту. Попередня обробка тексту складається з наступних кроків[4]:

- фільтрація – видалення спеціальних символів та пунктуації;
- токенізація – розбивання тексту на терми (терміни) – слова чи словосполучення;
- стемінг/лематизація - приведення слова до основи/початкової морфологічної форми;
- видалення стоп-слів;

- скорочення – видалення низькочастотних слів (є необов'язковим параметром);
- обробка заперечень.

Як правило, у відповідних програмних комплексах для попередньої обробки текстів використовуються спеціальні інструменти обробки натуральної (природньої) мови. Перш за все це пов’язано з тим, що алгоритми підготовки вимагають натренованих математичних моделей обробки великого розміру для кожної мови, що обробляється. У багатьох випадках натренована модель є тематично орієнтованою: призначеною, наприклад, для обробки та визначення тональності новин, або навпаки, спеціалізованою – націленою на певну предметну область.

Попередня обробка текстів корпусу не є повністю незалежним кроком. Комбінація та результат попередньої обробки залежать від моделей, що будуть застосовані в подальшому.

Наприклад, часто в текстовій аналітиці застосовується модель “Мішок слів”. При застосуванні моделі текст розбивається на n-грами. Це можуть бути окремі слова, або словосполучення, кількість слів в яких найчастіше складає два або три. При використанні окремих слів усі вони, за виключенням стоп – слів, що підлягають фільтрації, перетворюються у називний відмінок однини. При використанні словосполучень потрібно зберігати залежності між словами. Тому перетворення у називний відмінок не є доцільним.

Лематизація. У процесі лематизації можна знайти кілька застосувань. Насамперед, лематизацію використовують пошукові системи. Вона допомагає їм прискорити індексування та обробку запитів, а також підвищити релевантність своєї видачі. Пошукові системи пропускають кожну сторінку через алгоритм-лематизатор, щоб зберегти її в базі в компактній та зручній для пошуку форми. Запити теж відбуваються через лематизацію. Неважливо, що ввів користувач: "куплю машину" або "купити машину" – пошукова система перетворює слова в леми («купити машина») і покаже той самий результат.

Інше застосування лематизації – перевірка унікальності.

Лематизація по суті більш складний підхід до пошуку основи слова, ніж стемінг. Щоб зрозуміти, як працює лематизація, потрібно знати, як створюються різні форми слова. Більшість слів змінюється, коли вони використовуються в різних граматичні форми. Кінець слова замінюється на граматичне закінчення, і це призводить до нової форми вихідного слова. Лематизація виконує зворотне перетворення: замінює граматичне закінчення суфіксом або закінченням початкової форми.

Фільтрація, видалення стоп – слів, скорочення низькочастотних слів. Ці операції, як правило, проводяться безпосередньо із застосуванням інструментів мови програмування, з використанням якої створюється та чи інша аналітична система. Також алгоритми фільтрації, токенизації та інші вбудовані у деяких системах повнотекстового пошуку, наприклад, Elasticsearch.

Фільтрація використовується для видалення знаків пунктуації, спеціальних знаків ти символів та інших елементів початкового тексту, що не приймають участь в текстовому аналізі. Наявність таких символів, використання різних систем кодування можуть призводити до того, що два ідентичних терми система не буде вважати такими.

До стоп – слів найчастіше відносять ті слова, що не впливають на визначення тематики тексту. Це переважно прийменники, спілки, вигуки, займенники, а в іноземних текстах також артиклі.

Скорочення низькочастотних слів виконується не завжди. Основною метою цієї операції є видалення з тексту слів, що не сприяють віднесенню тексту до однієї з груп, а навпаки, “виділяють його” в окрему групу (кластер, тощо). З урахуванням того, що кількість кластерів/груп для повноцінної їх інтерпретації має бути значно обмеженою, такі слова погіршують якість кластеризації або інших операцій текстової аналітики.

Обробка заперечень. Для збільшення точності обробки текстів використовується алгоритм обробки заперечень, описаний Десом та Ченом. Суть його полягає в наступному[5]: у разі появи частки «не» на початок кожного слова

між цією частинкою і наступним розділовим знаком або іншою частинкою «не» приписується приставка «not_» ().

Приклад.

Речення: “Мені не сподобався цей фільм”.

Перетворюється на вид: “Мені сподобався not_цей not_фільм”.

1.4. Перетворення корпусу текстів в числову модель.

Оскільки більшість статистичних алгоритмів, наприклад машинне навчання та методи глибокого навчання, працюють із числовими даними, тому нам доводиться перетворювати текст у числа[6]. Щодо цього існує кілька підходів. Однак найвідоміші з них - Bag of Words (мішок слів), TF-IDF і word2vec.

Модель “Мішок слів”. Модель тексту "мішок слів" (bag-of-words) є сумативною єдністю (не системою) складаючих текст слів[7].

Основний об'єкт моделі мішка слів - це слово, забезпечене єдиним атрибутом, частотою цього слова.

У моделі тексту "мішок слів" враховується лише кількість входжень конкретних слів у вихідному тексті, інше – у тому числі: порядок слів у документі та морфологічні форми уявлення слів – ігнорується.

Модель тексту "мішок слів" була запропонована в 1975 році Дж. Солтоном, і в даний час є однією з найпоширеніших у найрізноманітніших областях лінгвістичних досліджень і сервісів, як правило, як основа для більш складних, перш за все "векторних моделей тексту".

Мішок слів використовується в "машинному навчанні на основі текстів" як один з основних об'єктів вивчення.

Корисним узагальненням формальної моделі тексту "мішок слів" (bag-of-words) може бути модель "мішок термів" (bag-of-terms).

Основний об'єкт моделі мішка термів - це терм, з єдиним атрибутом, частотою народження в тексті.

Терм – символьне вираження об'єкта формальної моделі (мови, системи)

Терм (формальне визначення): символьний вираз: $t(X_1, X_2, \dots, X_n)$, де літера t - ім'я терма, так званий функтор або «функціональна літера», а $X_1, X_2, \dots, X_n$ - терми, структуровані або найпростіші.

Терми можуть містити вільні змінні (параметри), фіксація значень яких однозначно визначає відповідно до семантичних правил мови деякий об'єкт - значення терму при даних значеннях його вільних змінних.

У логіко-математичному обчисленні терм є аналогом підмету або доповнення природної мови.

У моделі bag-of-terms, як терми можна розглядати будь-які символьні висловлювання тексту, зокрема - розділові знаки.

Таке узагальнення моделі мішку слів корисне при переході від окремих слів до n-грам: груп, що містять два та більше слів. На відміну від поодиноких слів модель n-gram дозволяє зберігати значення відносин між термами у тексті.

Параметризація корпусу текстів. Для застосування математичних моделей до корпусу текстів необхідно поставити у відповідність кожному елементу корпуса деякий набір числових параметрів.

Для кожного слова з набору моделі "мішок слів" може вказуватися деяка "вага". Таким чином, модель тексту є множиною пар «слово - вага». При цьому ваги можуть надаватися словам або основам слів.

Серед методів визначення ваги можна виділити такі [8, 9]:

Бінарний метод (поширене позначення - BI, від binary). Визначається лише наявність чи відсутність деяких термінів у документі. Застосовується для логічного інформаційного пошуку та автоматичної рубрикації текстів методами нейромережевих класифікаторів ART та SOM.

Кількість входжень (слів у документ). Передбачає невідповідність оцінки для текстів різної довжини — більша вага отримуватимуть більші тексти, оскільки в них більше слів;

Частота входження слова у документі (TF - term frequency). Частота обчислюється як відношення числа входження слова до кількості слів тексту.

При відносній простоті ця характеристика забезпечує прийнятний результат для методів інформаційного пошуку та класифікації (передбачає невідповідність оцінки для текстів різної довжини – недооцінюються довгі документи, оскільки в них більше слів та середня частота слів у тексті нижче).

Логарифм частоти входження слова (LOGTF). Вага документа, що входить у текст, визначається як $1 + log(TF)$, де TF — частота терміну (терма). Використання логарифмічної шкали дозволяє зробити модель стійкішою до переоцінки текстів різного обсягу.

Зворотна частота документів (IDF - inverse document frequency). Параметр є інверсією частоти, з якою зустрічається термін у документах.

Модель TF-IDF. З перелічених вище частотних моделей найбільше розповсюдження отримала модель TF-IDF. Ця модель дозволяє виправити деякі з проблем, що притаманні моделі мішок слів. В різних документах одні й ті ж слова зустрічаються із різною частотою. Величина TF – IDF коригує частоту слова з урахуванням рідкості.

Модель TF-IDF в числовому вигляді оцінює важливість слова, що зустрічається в документі відносно всієї колекції документів (корпусу текстів). Найчастіше TF-IDF виступає у ролі вагового коефіцієнта в інформаційному пошуку та аналізі текстів. Значення TF-IDF зростає пропорційно числу входжень слова в документ, але водночас знижується пропорційно частоті слова у всьому корпусі. Тому TF-IDF правильно враховує слова, які в одному документі здаються важливими, але насправді є часто вживаними, так що не можуть бути значними маркерами.

Корисність TF-IDF пов'язана з тим, що він враховує той факт, що деякі слова зустрічаються найчастіше. Величина TF-IDF дорівнює добутку частоти терму (term frequency - TF) на зворотну частоту документа (inverse document frequency - IDF). Її обчислення починається з обчислення частоти термів.

Вага терму. У пошукових системах TF-IDF часто використовується як механізм оцінювання та ранжування документів за релевантністю запиту користувача. Ефективно також застосування як фільтр стоп-слів. Для стоп-слів

вага TF-IDF мала, а термам, які рідко зустрічаються у всьому корпусі, призначається велика вага.

У таких важливих слів зазвичай велике значення TF і IDF, тому твір того та іншого виявляється великим. Слова, у яких TF-IDF відносно велике, можна вважати визначальними тему документа.

Крім пошукових систем, показник TF-IDF дуже корисний для цілей автоматичного реферування та класифікації текстів. Процес векторизації точніше, ніж у моделі мішка слів, але вимагає більше обчислювальних ресурсів, оскільки ми повинні підраховувати частоту входження слів не тільки в одному документі, але й всьому корпусі. Це вимагає більше одного проходу набору даних, а також декількох етапів попередньої обробки.

У найпростішій формі **TF** визначається як кількість входжень слова до документа. У складніших випадках лічильники нормуються на довжину документа. Довжина документів зазвичай різниться; щоб врахувати це, ми ділимо частоту терму на довжину документа (загальна кількість термів у ньому), що дає більш правильну міру. Якщо позначити t – терм, а d – документ, то

$$tf_{t,d} = \frac{count(t)}{count(alltermsind)} \quad (1.1)$$

Наступний крок – визначити, скільки інформації несе слово, навіщо слід обчислити його **IDF**. Ми хочемо виміряти, часто чи рідко зустрічається слово, і включити цю інформацію у ознаки вектор. Інтуїція підказує, що більшу вагу слід приписати термам, які зустрічаються у небагатьох документах корпусу.

Щоб визначити "зворотну частоту документа", ми спочатку ділимо спільну кількість документів у корпусі на кількість документів, що містять слово. Але в такій формі значення IDF не є ідеальним, тому що маскує вплив TF на остаточну вагу терму. Щоб згладити цю проблему, зазвичай беруть логарифм цієї величини (у табл. 1.1 наведено визначення змінних у формулі).

$$id\ f_t = \log\ \left(\frac{N}{df_t}\right) \quad (1.2)$$

Таблиця 1.1 – члени IDF.

Змінна	Опис
df_t	Частота документа – кількість документів, що містять терм t.
N	Загальна кількість документів в корпусі.

Таким чином, вага TF-IDF слова набуває вигляду

$$tf - idf_{t,d} = tf_{t,d} \times id\ f_t \quad (1.3)$$

велика вага буде у слів, що мають відносно високу частоту TF і відносно низьку частоту по всьому корпусі документів. Часто терми, що зустрічаються, відфільтровуються, тому що у них низька TF і висока частота зустрічальності у всіх документах. Легко бачити, що значення IDF завжди більше або дорівнює 0, оскільки аргумент логарифму більший або дорівнює 1. Зрозуміло також, що якщо терм зустрічається в багатьох документах, то відношення під знаком логарифму близьке до 1. Тому IDF виявляється близько до 0, а значить, і TF-IDF близько до 0. На практиці оцінка TF-IDF більш ефективна, ніж мішок слів, але її можна поліпшити за рахунок застосування N-грам.

Модель word2vec. Word2vec – технологія, розроблена Google, для знаходження семантичних зв'язків між словами[10].

Word2Vec включає набір алгоритмів для розрахунку векторних уявлень слів, припускаючи, що слова, що використовуються в схожих контекстах, означають схожі речі, тобто. семантично близькі.

$$\frac{(w_v \times w_c)}{\sum(w_v \times w_{c1})} \quad (1.4)$$

У чисельнику - близькість слів контексту та цільового слова.

У знаменнику — близькість інших контекстів і цільового слова.

Технологія Word2Vec використовує два різні методи:

– CBOW - пророцтво слова на підставі прилеглих слів.

– Skip-gram – передбачення прилеглих слів на підставі одного слова.

При розрахунку застосовуються штучні нейронні мережі. Під час навчання алгоритм формує оптимальний вектор кожного слова за допомогою CBOW або skip-gram.

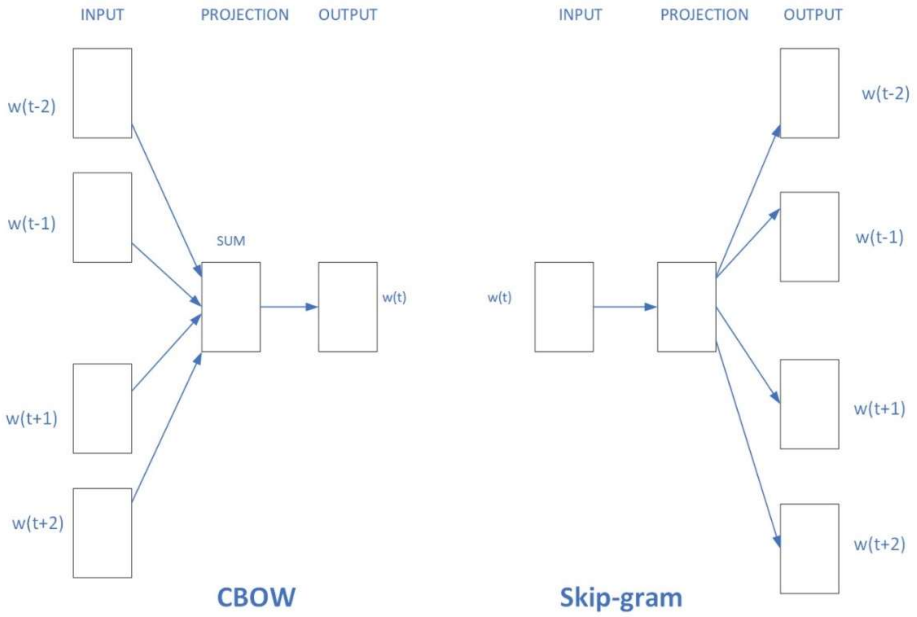


Рисунок 1.3 – схеми основних алгоритмів word2vec.

На рисунку 1.3 позначено: $w(t)$ – задане слово, $w(t-1), w(t-2)$ - прилеглі слова.

Метод представлення слів як векторів використовується для кластеризація слів та виявлення їх семантичної близькості, тобто. поділяє незв'язані слова та з'єднує пов'язані, що допомагає у завданнях кластеризації та класифікації текстів.

Працює метод у наступній послідовності.

- зчитується корпус, і розраховується частота кожного слова в корпусі;
- масив слів сортується за частотою та видаляються рідкісні слова;
- будується дерево Хаффмана (для кодування словника це значно знижує обчислювальну і тимчасову складність алгоритму);
- з корпусу зчитується так звана субпропозиція (базовий елемент корпусу - пропозиція, абзац, стаття) і проводиться субсемплірованія (процес вилучення найбільш частотних слів) з аналізу;

– за субпропозицією проходимо вікном (максимальна дистанція між поточним та передбачуваним словом у реченні);
– застосовується нейрона мережа прямого поширення з функцією активації ієрархічний софтмакс та/або негативне семплювання.

1.5. Обробка векторних моделей корпусу текстів.

Після отримання векторної, матричної або якоїсь іншої моделі тексту відбувається перехід безпосередньо до застосування математичної моделі кластеризації, класифікації, аналізу тональності, тощо.

Розглянемо деякі з основних моделей, що застосовуються.

Кластерний аналіз - завдання розбиття заданої вибірки об'єктів (ситуацій) на непересічні підмножини, звані кластерами, так, щоб кожен кластер складався з схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися. Кластерний аналіз можна вважати найпоширенішим завданням навчання без учителя. У класичній задачі кластеризації навчальна вибірка є набір окремих об'єктів

$$X = \{x[i]\}_{i=1}^{n} \quad (1.5)$$

що характеризуються вектором дійсних ознак $x[i] = (x_1[i], \dots, x_d[i])$. Потрібно побудувати алгоритм (кластеризації), який розбив би вибірку на групи, що не перетинаються (кластери)

$$X = \cup_{j=1}^{k} C_k, C_j \subset \{x_1, \dots, x_m\}, C_j \cap C_i = \emptyset \quad (1.6)$$

Класифікація - один з розділів машинного навчання, присвячений вирішенню наступного завдання [11, 12].

Є множина об'єктів (ситуацій), розділених, певним чином, на класи. Задано кінцеву множину об'єктів, для котрих відомо, до яких класів вони відносяться. Ця множина називається навчальною вибіркою. Класова приналежність інших об'єктів невідома. Потрібно побудувати алгоритм, здатний класифікувати довільний об'єкт із вихідної множини.

Класифікувати об'єкт — означає, вказати номер (або найменування класу), до якого належить даний об'єкт. У класичній задачі класифікації навчальна

вибірка є набором окремих об'єктів $X = \{x[i]\}_{i=1}^{n}$, що характеризуються вектором дійсних ознак $x_i = (x_{i,1}, \dots, x_{i,d})$. Як результату для об'єкта $\boldsymbol{x}$ фігурує змінна t, що приймає кінцеве число значень, зазвичай з множини $\overline{1,n}$ $(\overline{0,n})$. Потрібно побудувати алгоритм (класифікатор), який за вектором ознак $\boldsymbol{x}$ повернув би мітку класу $\hat{t}$ або вектор оцінок належності (апостеріорних ймовірностей) до кожного з класів $\{p(s|x)\}_{x=1}^{l}$.

Аналіз тональності. Фактично, аналіз тональності є особливим випадком побудови бінарного/тернарного класифікатора. В першому випадку тексти корпусу ранжуються на тексти позитивної або негативної тональності, в другому випадку то можливих тональностей додається нейтрально тональність.

Однією з методик побудови класу нейтральних відгуків можна розглядати метод відмови від класифікації – умову, при виконанні якої клас не призначається.

Наприклад, класифікатор не призначає клас, якщо

$$|\ln p(pos) - \ln p(neg)| < t \quad (1.7)$$

$p(pos)$ - ймовірність того, що відгук позитивний,

$p(neg)$ - ймовірність того, що відгук негативний.

Параметр t знаходиться емпіричним шляхом при налаштуванні класифікатора.

1.6. Математичні моделі класифікаторів.

Розглянемо деякі математичні моделі класифікаторів, що можуть бути використані при побудові додатку оцінки тональності текстів.

Логістична регресія. Логістична регресія — це тип лінійного класифікатора [13]. Він використовується для передбачення, чи належить об’єкт, що розглядається, до класу «1» або класу «0». Найпоширенішим типом задач, що розв’язуються з використанням логістичної регресії, є бінарні задачі. Логістична регресія також може бути використана, коли існує більше двох типів залежних змінних. Тоді буде використаний багаточлен логістичної регресії. Основою класифікації, яка використовується в логістичній регресії, є логістична функція,

створена для розподілу ймовірностей, що відповідають зваженим векторам ознак. Логістична функція має вигляд

$$h_W(X) = \frac{1}{1 + e^{-W^T X}} \quad (1.8)$$

У наведеному вище рівнянні незалежна змінна x - вектор ознак для кожного компонента зважений незалежними характеристиками об'єкта. Загальний «вхід» у логістичну функцію може бути записаний в матричному вигляді наступним чином

$$X = W^T x \quad (1.9)$$

де W – вектор відповідних ваг компонентів, включно із зміщенням, а x – n-мірний вектор ознак.

$$W = (w_n, w_{n-1}, \dots, w_2, w_1, b) \quad (1.10)$$

$$x = (x_n, x_{n-1}, \dots, x_2, x_1, 1) \quad (1.11)$$

У випадку логістичної регресії нашою цільовою функцією є від'ємна логарифмічна функція подібності або логарифмічна функція подібності. Це функція, яку потрібно оптимізувати, щоб дізнатися межу прийняття рішення, пов'язану з класифікацією. Функція правдоподібності визначається наступним чином

$$L(W, x) = Pr(Y|X; W) \quad (1.12)$$

що є добутком усіх окремих ймовірностей, оскільки ми вважаємо, що всі компоненти вектору ознак є незалежними один від одного. Фактично цільовою функцією, яка є негативною логарифмічною функцією подібності, є функція, яка визначається наступним чином

$$-logL(W|x) = -\sum_{i=1}^{N} \log Pr(y_i|x_i; W) \quad (1.13.1)$$

$$-logL(\theta|x) = -\sum_{i=1}^{N} y_i \log h_W(x) + (1 - y_i) \log\big(1 - h_W(x)\big) \quad (1.13.2)$$

Навчанню в моделях логістичної регресії відповідає мінімізація вищезазначеної функції, яка також відома як функція витрат. Знову ж, як уже згадувалося,

алгоритм використовує певні припущення для простоти реалізації. До них належать:

- припущення незалежності спостережень одне від одного. Це дозволяє реалізувати ймовірності вимірювань як добуток окремих ймовірностей у функції ймовірності;
- кореляція між незалежними змінними є досить низькою;
- вимога лінійної залежності між логарифмом ймовірності та незалежними змінними.

Байєсівський класифікатор. Існує сімейство класифікаторів, заснованих на імовірнісних класифікаціях за теоремою Байєса з сильною незалежністю між векторами ознак. Це популярний на практиці метод застосовується для категоризації тексту, наприклад, класифікації електронної пошти, що не містить спам, базуючись на частоті слів як векторах ознак. Попри спрощену побудову, такі класифікатори мають застосування та показують досить непогані результати при практичних застосуваннях.

Модель Байєса – модель умовної ймовірності, яка присвоює клас C_k вектору ознак x з імовірністю, що визначена як

$$Pr(C_k|x_1, x_2, \dots, x_n) \qquad (1.14)$$

Цей тип моделі буде обчислювати умовну ймовірність для кожного класу k на основі вектора ознак. Це означає, що якщо вектор ознак має великі розміри, то цей процес неможливий.

Тому ймовірність представляють в іншому вигляді

$$Pr(C_k|\mathrm{x}) = \frac{Pr(C_k)Pr(\mathrm{x}|C_k)}{Pr(\mathrm{x})} \qquad (1.15)$$

Оскільки знаменник не залежить від C і дані значення вектору ознак наведені, то знаменник є константою й можна зосередитись на чисельнику. Чисельник можна розбити в наступний вираз, використовуючи правило ланцюга

$$Pr(C_k, x_1, x_2, \dots, x_n) = Pr(x_1, x_2, \dots, x_n, C_k) \qquad (1.16.1)$$

$$Pr(C_k, x_1, x_2, \dots, x_n)$$
$$= Pr(x_1|x_2, \dots, x_n, C_k)Pr(x_2|x_3, \dots, x_n, C_k) \dots Pr(x_{n-1}|x_n, C_k) \quad (1.16.2)$$

Остаточно модель можна представити у вигляді

$$Pr(C_k|x_1, x_2, \dots, x_n) = Pr(C_k)\prod_{i=1}^{N} Pr(x_i|C_k) \quad (1.17)$$

Цільова функція — це апостеріорна ймовірність, яка максимізується під час процесу. Функція подібності $Pr(\text{x}|C_k)$ складається спочатку. Потім за допомогою рівняння Байєса, поданого вище, проводиться обчислення апостеріорної ймовірності. Клас з найбільшою ймовірністю - це передбачення. Апостеріорна ймовірність задається через $Pr(C_k|\text{x})$.

Ця методика передбачає умову незалежності змінних ознак. Це можна розглядати як недолік класифікатора, оскільки в більшості випадків у випадку реальних даних припущення про незалежність може виявитись хибним, наслідком чого стануть недостовірні результати.

Однак цей класифікатор є простим і швидким і може добре працювати в багатокласовій класифікації. Незважаючи на недолік цієї методики, який полягає у припущенні незалежності, вона працює досить ефективно і потребує менше навчальних даних у порівнянні з іншими методиками класифікації.

Метод опорних векторів (SVM). Метод опорних векторів - це набір навчальних моделей машинного навчання, які контролюються в природі [14]. Модель навчається з набором точок даних навчання, які належать будь-якому з двох класів двійкового класифікатора. На основі цього SVM реалізує вивчену модель на новіших точках даних, що належать до тестової вибірки, і розміщує їх в один з двох класів. Таким чином, SVM є неімовірнісним двійковим класифікатором.

Ідея класифікації полягає у реалізації $(n-1)$ - вимірної гіперплощини для лінійної класифікації n-вимірних векторів ознак у два окремі класи. Таких гіперплощин, які розділяють точки даних, може бути нескінченно багато, тому потрібно вибрати гіперплощину або лінійний класифікатор, що має максимальну

різницю. Різниця визначається як відстань найближчих точок кожного класу до обраної гіперплощини. Цільовою функцією є відстань найближчих точок в обох класах до гіперплощини, а метою оптимізації є максимізація цієї відстані.

SVM має параметр для регуляризації, який допомагає алгоритму оптимізації уникати перенавчання. Він також може вмістити дані нелінійно, використовуючи техніку, яка називається kernel-trick. Також він базується на опуклій задачі оптимізації, яка має дуже ефективні методи вирішення. Недоліками є залежність від регуляризації параметрів та вибору ядра.

Тож, алгоритм заснований на припущенні, що чим більша різниця або відстань між паралельними гіперплоскостями, тим менше буде середня помилка класифікатора.

Градієнтний бустінг. Градієнтний бустінг (підвищення градієнту) - це сімейство потужних методів машинного навчання, які продемонстрували значний успіх у широкому діапазоні практичних застосувань [15]. Якщо вивчати, наприклад, статистику змагань останніх років на Kaggle, лідируючі рішення багатьох змагань побудовані з використанням саме градієнтного бустінгу.

Методи градієнтного бустінга налаштовуються під конкретні потреби задачі, наприклад, за рахунок використання різних функцій втрат.

Коротка базовий алгоритм моделей градієнтного бустінгу може бути описаний наступним чином.

Алгоритм. Можна довільно вказати як функцію збитків, так і моделі базового. На практиці, враховуючи якусь конкретну функцію втрат $\Psi(y, f)$ та базового учня $h\,(x, \theta)$, отримання рішення для оцінки параметрів може бути важким. Для вирішення цього було запропоновано вибрати нову функцію $h\,(x, \theta_t)$, яка буде найбільш паралельною від'ємному градієнту $\{g_t(x_i\}_{i=1}^{M}$ вздовж спостережуваних даних

$$g_t(x) = E_y\left[\frac{\partial \Psi(y, f(x))}{\partial f(x)}|x\right]_{f(x)=\hat{f}^{t-1}(x)} \qquad (1.18)$$

Замість того, щоб шукати загальне рішення для приросту прискорення у просторі функцій, можна просто вибрати новий приріст функції, який найбільш корелює з $-g_t(x)$. Це дозволяє замінити потенційно дуже складне завдання оптимізації на класичне завдання мінімізації найменших квадратів

$$(\rho_t, \theta_t) = \arg\min_{\rho,\theta} \sum_{i=0}^{N} [-g_t(x) + \rho h(x_i, \theta)]^2 \quad (1.19)$$

У якості підсумку можна сформулювати алгоритм в редакції Фрідмана (2001).

Початкові дані.

– вхідні дані $(x, y)_{i=1}^{N}$
– кількість ітерацій M
– вибір функції втрат $\Psi(y, f)$
– вибір базового учня $h\ (x, \theta)$

Алгоритм

– ініціалізація $\hat{f}_0$ константою;
– ітераційний цикл (від 1 до M):
  - розрахунок від'ємного градієнту $g_t(x)$
  - підгонка нового базового учня $h(x, \theta_t)$
  - знаходження найкращого градієнт спуску з розміром кроку ρ_t:

$$\rho_t = \arg\min_{\rho} \sum_{i=0}^{N} \Psi\left[y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right] \quad (1.20)$$

  - оновлення оцінки функції

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x, \theta_t) \quad (1.21)$$

Лінійний дискримінантний аналіз (LDA)

Лінійний дискримінантний аналіз (LDA) зазвичай використовується для ідентифікації лінійних ознак, що максимізують поділ даних між класами, мінімізуючи варіацію ознак у середині класів.

Розглянемо навчальний набір даних, що містить N прикладів $\{x^1 \dots x^N\}$, де кожен приклад x^i є вектором стовпця довжиною d. Кожен навчальний приклад належить до одного з K класів. Нехай C_k - множина всіх прикладів класу k, і нехай $n_k = |C_k|$ - кількість прикладів у класі $k = 1 \dots K$. У LDA обчислюються матриці розсіювання всередині класу та між класами:

$$S_w = \frac{\sum_k \sum_{i \in C_k} (x^i - m_k)(x^i - m_k)^T}{N}, \qquad S_b = \frac{\sum_k (m_k - m)(m_k - m)^T}{N} \quad (1.22)$$

де $m_k = \frac{1}{n_k} \sum_{i \in C_k} x^i$, є середнім значенням k-го класу та $m = \frac{1}{N} \sum_i x^i$ є середнім значенням набору даних.

Ми шукаємо лінійне перетворення $x \rightarrow W^T x$, яке максимізує дисперсію між класами щодо дисперсії всередині класу, де W - матриця $d \times d'$, причому d'- бажана кількість розмірностей.

Можна показати, що стовпці оптимального W є узагальненими власними векторами, такі що $S_b w = \lambda S_w w$, що відповідає d' найбільшим власним значенням. Одним із наслідків цього результату є те, що W одночасно діагоналізує матриці розсіювання $W^T S_b W$ і $W^T S_w W$. Іншими словами, LDA пов'язує дані як між класами, так і всередині них.

1.7. Оцінка якості класифікації.

Для оцінки якості вирушення задачі класифікації звичайно використовують наступні оцінки [16, 17].

Accuracy

Обчислюється як відношення загальної кількості правильних прогнозів до загальної кількістю прогнозів (ці прогнози включають правильну класифікацію та правильну некласифікацію).

Математично це виглядає наступним чином

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.23)$$

TP, TN – кількості правильних позитивних та негативних передбачень

FP, FN – кількість неправильних позитивних та негативних передбачень.
А їх сума, відповідно, є загальною кількістю передбачень.

Точність та повнота

Точність визначається як правильно передбачена класифікація (позитивна) над загальною прогнозованою класифікацією[18] Висока точність стосується низької похибки моделі при класифікації точок даних, які не належать до певного класу в цьому класі.

$$\text{Точність} = \frac{TP}{TP + FP} \quad (1.24)$$

Повнота визначається як відношення правильно передбаченої класифікація до усіх членів певного класу.

$$\text{Повнота} = \frac{TP}{TP + FN} \quad (1.25)$$

Метрика F1 є спільна метрика точності та повноти. Вона дозволяє отримати більш збалансовану оцінку моделі, аніж кожна із загаданих метрик окремо.

$$F1\ Score = \frac{2 \cdot \text{Точність} \cdot \text{Повнота}}{\text{Точність} + \text{Повнота}} \quad (1.26)$$

Площа під кривою (AUC)

ROC-крива - графік, що дозволяє оцінити якість бінарної класифікації, відображує співвідношення між часткою об'єктів від загальної кількості носіїв ознаки, правильно класифікованих до загальної кількості об'єктів, що не несуть ознаки, помилково класифікованих, як такі, що мають ознаку. Також відома як крива похибок [19].

Площу під кривою можна використовувати як метрику для оцінки точності моделі. Оскільки графік побудований як справжні позитивні та хибні негативні, то область під кривою (AUC) відповідає ефективності моделі.

Класифікація якості ефективності наведена нижче:

– 0,9 – 1 - відмінно

– 0,8 - 0,9 - добре
– 0,7 - 0,8 - досить добре
– 0,6 - 0,7 - погано
– 0,5 - 0,6 – цілком хибно.

1.8. Математичні моделі кластеризації

Алгоритми кластеризації можна розділити на ієрархічні та неієрархічні. Основна особливість ієрархічних методів полягає в тому, що вони будують певну ієрархію груп, розбиваючи множину досліджуваних об'єктів. Неієрархічні алгоритми при поділі на кластери керуються певною цільовою функцією, значення якої намагаються мінімізувати.

Безпосередньо постановку задачі кластеризації було надано в розділі 1.5.

Спочатку розглянемо загальний підхід до ієрархічної кластеризації.

Агломеративна ієрархічна кластеризація. Алгоритм Ланса – Уільямса.

а) Розглянемо всі кластери, як одно-елементні масиви

$$t \coloneqq 1, \qquad C_t = \{\{x_1\}, \dots, \{x_l\}\}, \qquad R(\{x_i\}, \{x_j\}) \coloneqq d(x_i, x_j) \quad (1.27)$$

б) для всіх $t = 2, \dots, l$ (t – номер ітерації) знаходимо в C_{t-1} два найближчі кластери

$$(U, V) \coloneqq \arg\min_{U \neq V} R(U, V), \qquad R_t \coloneqq R(U, V), \qquad W = U \cup V \quad (1.28)$$

в) з'єднуємо в один кластер

$$C_t \coloneqq C_{t-1} \cup \{W\} \backslash \{U, V\} \quad (1.40)$$

г) для всіх $S \in C_t$ обчислюємо $R(W, S)$ по формулі Ланса – Уільямса

$$R(U \cup V, S) = a_U \cdot R(U, S) + a_V \cdot R(V, S) + \beta \cdot R(U, V) + \gamma \cdot |R(U, S) - R(V, S)| \quad (1.41)$$

де $\alpha_U, \alpha_V, \beta, \gamma$ – числові параметри, значення яких залежать від методики вибору $R(W, S)$ (по ближньому чи дальньому сусіду, груповій середній відстані, відстані між центрами чи відстані Уорда).

Її застосування полягає у тому, щоб визначити відстань $R(W, S)$ між клаастерами $W = U \cup V$ та S, знаючи відстані $R(U, S), R(V, S), R(U, V)$.

Алгоритм k-середніх (k-means).

Алгоритм k-means є одним з найбільш простих та найчастіше використовуванних алгоритмів неієрархічної кластеризації [20].

- задаємо k об’єктів, котрі будуть еталонами. Ці точки вважаються центрами ваги кластерів на першому кроці процедури. Кожному еталону присвоюється порядковий номер L ($L = 1,2,3 \dots, k$).
- з решти (n-k) об’єктів вибирається точка x_i ($i = 1,2,..n$) з координатами $(x_{i1}, x_{i2}, .., x_{im})$ і перевіряється до якого з еталонів вона знаходиться найближче. Для цього використовується евклідова метрика, або деяка інша метрика, наприклад, манхетенська відстань або відстань Махаланобіса. Об'єкт, що перевіряється, приєднується до того центру ваги (еталону), якому відповідає мінімальна відстань. Еталон заміняється новим, перерахованим з урахуванням нової точки, а його статична вага (кількість об’єктів, що входять у даний кластер) збільшується на 1. Аналогічно вибираємо і перевіряємо для решти з $(n - k)$ точок.

Щоб отримати стійкий розподіл об’єктів по кластерах всі точки $x_1, x_2, .., x_n$ знову приєднуються до отриманих в результаті попереднього кроку центрів ваги. Новий розподіл об’єктів порівнюється з попереднім. Якщо вони співпадають, то робота алгоритму припиняється. В іншому випадку цикл повторюється.

Кінцевий розподіл, зазвичай, має центри ваги, що не співпадають з початковими еталонами. Їх можна позначити через $c_1, c_2, .., c_n$. При цьому кожна точка x_i буде відноситись до такого кластеру L, для котрого виконується вимога

$$d(x_j, c_L) = \min_{1 \le L \le k} d(x_j, c_L) \quad (1.29)$$

2. ПОСТАНОВКА ЗАДАЧІ, ПРОЕКТУВАННЯ ДОДАТКУ.

2.1. Постановка задачі.

З використанням технік та моделей, що розглянуто в першому розділі, побудуємо додатки для кластеризації заголовків новин та аналізу тональності коротких повідомлень фінансової спрямованості.

Датасети, що буде використано для налаштування та тестування додатків, розміщені на Kaggle.com та мають ліцензії вільного використання в навчальних та наукових цілях. Мова датасетів – англійська.

Датасет, складений із заголовків новин, містить біля одного мільйона записів та складається з двох полів: дати публікації та власне заголовка (https://www.kaggle.com/datasets/therohk/million-headlines). Усі заголовки представлені в єдиному кодуванні ASCII та в нижньому регістрі. Джерелом новин є Australian Broadcasting Corp.(https://www.abc.net.au/) Новини охоплюють період біля 18 років. Результатом роботи додатку є маркування кожної новини відповідним номером кластера та виділення ключових характеристик кластерів: розмірів, найчастіших суттєвих слів, тощо.

Інший датасет представляє собою твердження фінансової спрямованості, помарковані відповідним класом тональності: “позитивне”, “нейтральне”, “негативне” (https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis). Датасет є поєднанням двох датасетів (FiQA, Financial PhraseBank) в єдиному форматі та файлі. Метою побудови другого додатку є побудова та навчання моделі класифікатора, що буде коректним чином визначати клас тональності повідомлення. Оцінка якості класифікації буде проводитись за метриками, що розглянуто в підрозділі 1.7, що присвячений оцінці якості класифікації.

2.2. Проектування додатку.

Основною метою роботи є побудова додатку, що дозволяє виконувати необхідні операції з обробки текстових корпусів, тому алгоритм буде

реалізовано мовою програмування з використанням Jupiter Notebook, що є популярним середовищем моделювання та розробки проектів DataScience.

Як було зазначено в попередньому розділі, загальна схема алгоритму роботи текстової аналітики є подібною для багатьох застосунків. Схема відображена на рисунку 2.1 у вигляді activity diagram.

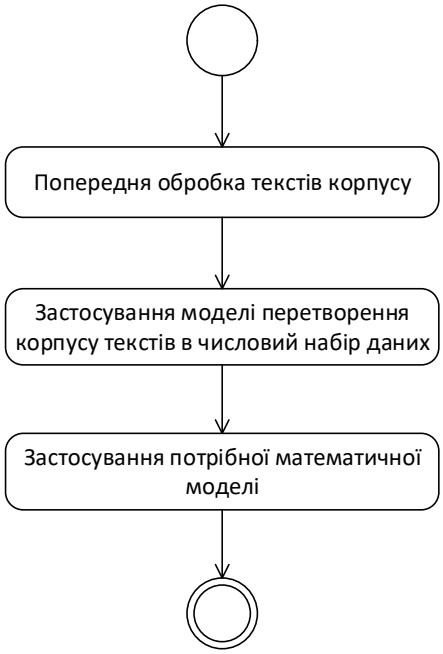


Рисунок 2.1 – Activity diagram додатку.

Треба відзначити, що частини представленого алгоритму не є тісно зв’язаними між собою: заміна моделей та алгоритмів в одному з блоків не створює необхідність застосування змін в інших частинах. Тому вимогою до побудови додатку є максимально можлива незалежність модулів та стандартизація сігнатур вхідних та вихідних даних.

Один з варіантів організації попередньої обробки текстів показано на рисунку 2.2.

В якості методи приведення до початкової форми використовуємо стемінг або лематизацію. Із збільшенням обчислювальних можливостей остання є переважною незважаючи на те, що потребує суттєво навчених моделей суттєво більших обсягів.

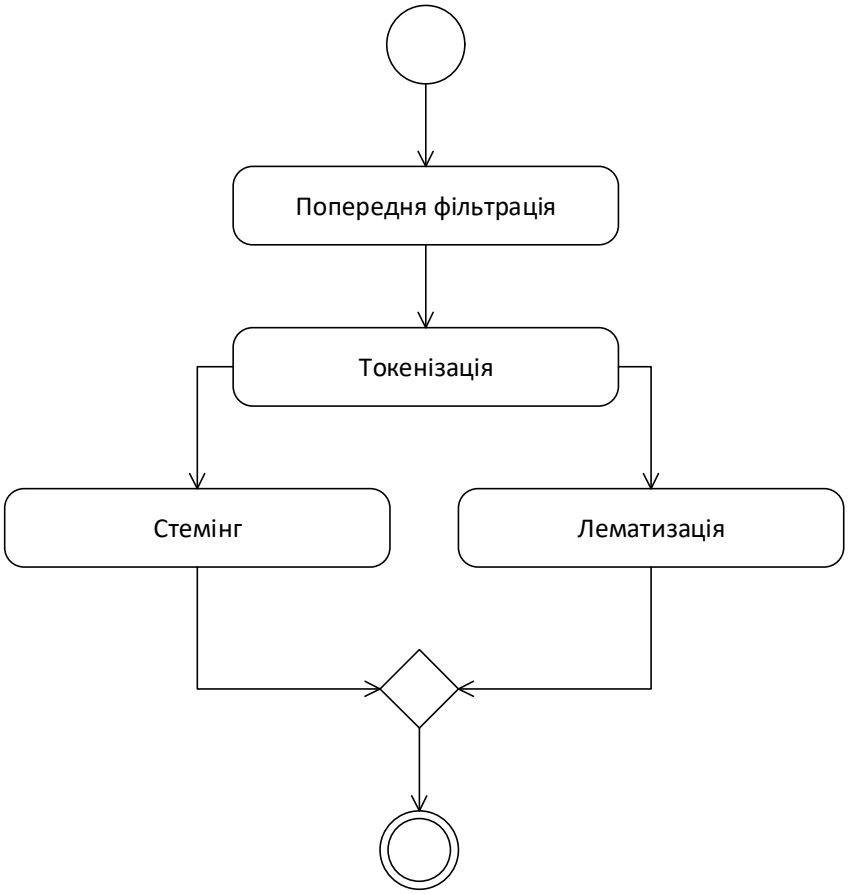


Рисунок 2.2 – Acitivity diagram попередньої обробки текстів корпусу.

В якості алгоритму переведення текстової інформації в нумеричний датасет використаємо розглянуту в попередньому розділі модель TF-IDF. Математичний опис її є досить простим та знаходиться у відкритому доступі, на відміну наприклад, від word2vec. В якості підвищення складності моделі скоріше треба розглядати підвищення якості попередньої обробки текстів та використання бі-грам або три-грам замість моделі мішку слів, що руйнує взаємозв’язки між словами, як було відмічене в першому розділі.

Для класифікації та кластеризації, а також препроцесінгу нумеричного датасету й оцінки результатів отриманої класифікації та кластеризації будемо використовувати стандартні можливості бібліотеки Scikit-learn, що є однією з найбільш популярних бібліотек машинного навчання разом з TensorFlow від Google.

Відповідно до діаграми 2.1 й сам додаток складається з трьох компонентів. На діаграмі 2.3 відтворимо структуру компонентів та їх інтерфейси.

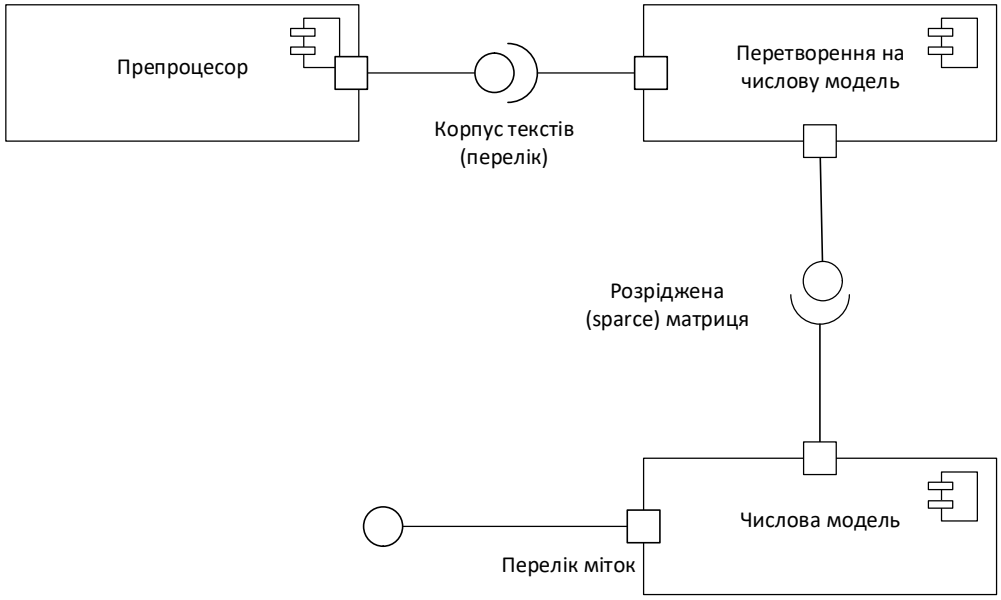


Рисунок 2.3 – Діаграма компонентів.

На вході додатку – корпус необроблених текстів, на виході – перелік міток відповідно до задачі, що реалізується. В розглянутих випадках – мітки класів та кластерів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКІВ.

3.1. Інструменти реалізації.

Основна мова програмування - Python, версія 3.9. Python обрано як одну з найбільш популярних мов програмування. Python – скриптова мова, що не компілюється до запуску додатку[21-24]. Програмувати та використовувати Python – скрипти можна практично на всіх платформах: від iOS та Android до серверних ОС. Перевагами Python є синтаксична простота, що обумовлює швидкість реалізації, та наявність різноманітних бібліотек, що також скорочують та спрощують процес розробки, дозволяючи керувати процесами на високому рівні, без суттєвого занурення у деталі.

До недоліків можна віднести відносно малу швидкість роботи за рахунок виконання скрипту інтерпретатором без попередньої компіляції та обмежену кількість фреймворків для створення графічних інтерфейсів користувача.

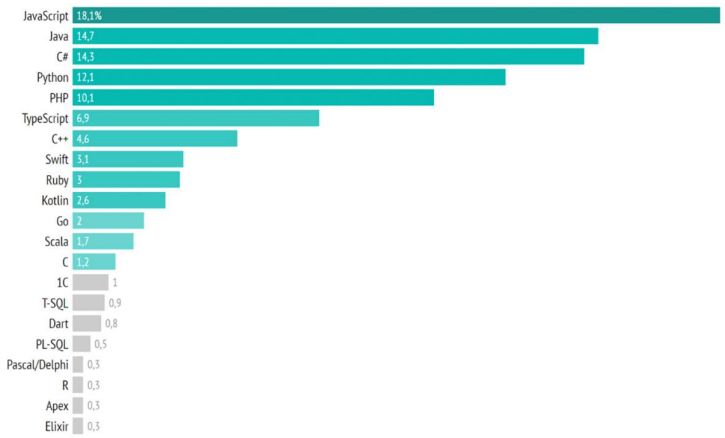


Рисунок 3.1 - Рейтинг мов програмування

Середовище розробки. Jupiter – notebook – web-додаток з відкритим програмним кодом, що часто використовується для Data Science проектів, що реалізуються одним виконавцем.

На відміну від стандартного інтерпретатора мови Python, в Jupiter код виконується покрокове – може бути виконаний лише код, розташований в одній комірці. При цьому зберігаються результати попередніх обчислень. Результати виконання коду розташовуються безпосередньо після відповідної комірки з

кодом. Це дозволяє виконувати окремі операції та аналізувати отримані результати без необхідності виконання скрипту в цілому, від початку.

Такий підхід дає суттєві переваги, наприклад, у налаштуванні моделей, розвідці даних та інших типових завдань Data Science.

В якості ключових бібліотек використовуємо: Scikit-learn та NLTK.

Scikit – learn – бібліотеки реалізація препроцесінга даних, математичних моделей машинного навчання, а також достатньо різноманітний інструментарій налаштування та оцінки таких моделей.

NLTK (Natural language toolkit) - пакет бібліотек та програм для символьної та статистичної обробки природної мови, написаних мовою програмування Python.

Для CRUD – операцій та маніпуляцій із даними використовуємо pandas, для візуалізації даних - Matplotlib та Seaborn.

3.2. Програмна реалізація.

Відповідно до другого розділу, обидва програмні додатки поділяються на три частини: препроцесінг, перехід до числової моделі та власне застосування моделей класифікації та кластеризації. Частини незалежні, мають стандартизований ввід/вивід, тому, розглядатимемо особливості програмної реалізації кожної частини окремо, не повторюючи розгляд перших двох частин для кожного додатку окремо.

Препроцесінг корпусу текстів. Огляд програмної реалізації почнемо з реалізації препроцесінга корпусу текстів.

Завантаження даних відбувається стандартними методами pandas. Після чого відбувається препроцесінг даних.

Спочатку розглядаємо створений власноруч конвеєр препроцесінгу. В якості набору (set) стоп – слів використовуємо перелік стоп – слів з класу sklearn у поєднанні із розділовими знаками.

```
# знаки пунктуації - символи, що не повинні бути враховані моделлю
punc = ['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}',"%"]
# стоп - слова - слова, що виключаються із розрахунку коефіцієнтів моделі
# використовується список стоп-слів бібліотеки NLTK для англійської мови у поєднанні із пунктуацією та без такого поєднання
# різні списки стоп слів будуть використанні при побудові різних препроцесорів
stop_words_wo_punc = text.ENGLISH_STOP_WORDS
stop_words_with_punc = text.ENGLISH_STOP_WORDS.union(punc)
# приклади з отриманого списку стоп - слів (оригінальний набір представляє собою тип frozenset)
list(stop_words)[:15]
```

Рисунок 3.2 – Формування набору стоп – слів.

Для токенізації, стемінгу та лематизації використовуємо стандартні можливості бібліотеки NLTK: ці процеси вимагають навчених моделей на великих корпусах текстів. Такою великою потребою в навчанні пояснюється, в тому числі, нажаль відсутність якісних моделей українською мовою.

```
# використання стеммеру з NLTK
stemmer = SnowballStemmer('english')
```

```
# токенізатор NLTK
tokenizer = RegexpTokenizer(r'[a-zA-Z\']+')

def tokenize(text):
    return [stemmer.stem(word) for word in tokenizer.tokenize(text.lower())]
```

Рисунок 3.3 – Стемер та токенайзер.

В якості альтернативи стемеру використовуємо лематизатор тієї ж бібліотеки, що в подальшому дає кращі результати.

```
# використовуємо лематізатор. Це дозволяє отримувати початкову форму слів замість лише коренів слів
lemmatizer = WordNetLemmatizer()

def tokenize_lemmas(text):
    return [lemmatizer.lemmatize(word) for word in tokenizer.tokenize(text.lower())]
```

Рисунок 3.4 – Використання лематизатору NLTK.

Зазначимо, що після використання токенізатора корпус текстів перетворюється на “мішок слів” – обрана довжина токенів складає 1.

Лематизатор/стемер використовуються для зведення різних форм одного слова до єдиного вигляду, що підвищує якість результатів подальшого застосування різних моделей.

Векторизація перетвореного текстового корпусу. В практичній моделі, що розглядається, кількість ознак (слів) обмежується 7% від загальної їх кількості. Для зменшення кількості ознак (розмірності простору) векторизотор

використовує модель PCA - Principal component analysis. Треба зауважити, що зменшення розмірності простору не є тотожним вибору найсуттєвих ознак з набору.

Відбір моделлю застосовується із використанням косинусів подібності між векторними представленнями слів.

```
# обмежимо кількість слів - ознак в наборі. Модель використає лише найважливіши з слів
vectorizer3 = TfidfVectorizer(stop_words = stop_words, tokenizer = tokenize_lemmas, max_features = int(0.07 * N_ROWS))
X3 = vectorizer3.fit_transform(desc)
words = vectorizer3.get_feature_names_out()
print(f'Загальна кількість слів у отриманій моделі - {len(words)}')
```

```
700
Загальна кількість слів у отриманій моделі - 700
```

Рисунок 3.5 – Векторизація, обмеження кількості ознак.

В процесі побудови моделі класифікації будуємо хмару слів – графічну інтерпретацію “мішку слів”, що був отриманий в ході векторизації.

Рисунок 3.6 – Хмара слів.

Розмір (кегль) шрифту відповідає частоті, з якою зустрічається слово в корпусі тексті. В текстах фінансових нових, звичайно, найчастіше зустрічаються “eur”, “company”, “sale”, “group”, “market” та подібні.

Графічне зображення хмари слів є набільш наочним способом відображення отриманого мішку слів, ранжування мішку слів за частотою появи у корпусі текстів.

За моделлю TF-IDF чим більше частота появи слів у корпусі, тим менше вплив цих слів на кінцевий результат моделювання. Тотбо найбільш великим шрифтом ми бачимо слова, що мають найменший вплив на класифікацію або кластеризацію корпусу.

Результатом застосування моделі TF-IDF є розріджена матриця, що наведена на рисунку 3.7.

	aa	aal	aaland	aalto	aaltonen	aapl	aaron	aava	aazhang	ab	...	zone	zoo	zp	zsl	zte	zu	zurich	zwe	zwhthsvfsf	zy
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Рисунок 3.7 – Розріджена матриця, що репрезентує текстовий корпус.

Класифікація текстів корпусу за емоційним забарвленням. Перейдемо до побудови текстового класифікатора.

Для використання моделей класифікації ознаки класів бажано теж перетворити на числові. Для кодування емоційних ознак використовуємо числове кодування, технічно використовуємо LabelEncoder.

```
le=LabelEncoder()
df_dtm['Sentiment']=le.fit_transform(df_dtm['Sentiment'])
```

```
df_dtm['Sentiment'].value_counts()
```

```
1    3124
2    1852
0     860
Name: Sentiment, dtype: int64
```

Рисунок 3.8 – Застосування LabelEncoder.

Для повернення до початкових міток класів створюємо мапінг, який використаємо після застосування моделі класифікації.

```
: le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
  le_name_mapping # мітки, присвоєні класам для подальшого порівняння із початковим датасетом
: {'negative': 0, 'neutral': 1, 'positive': 2}
```

Рисунок 3.9 – Створення довідника відповідності міток класів.

Для навчання та тестування отриманого класифікатора поділяємо датасет, що використовується на тренувальну та тестові вибірки.

```
# розподіл датасету на тренувальний та тестувальний
X_train, X_test, y_train, y_test = train_test_split(df_dtm.drop('Sentiment',axis=1),
                                                    df_dtm['Sentiment'], test_size=0.1,
                                                    stratify=df_dtm['Sentiment'])
```

Рисунок 3.10 – Поділ на тренувальну та тестову вибірки.

Для побудови класифікатора застосовуємо модель SVM (модель опорних векторів), що дає досить хороші результати для задач текстової аналітики та часто використовується у комерційних задачах. Використовуємо клас бібліотеки sklearn.

Для оцінки результатів роботи класифікатора порівнюємо кількість елементів класів, будуємо матрицю помилок та розраховуємо метрику площу під кривою ROC – метрику ROC-AUC.

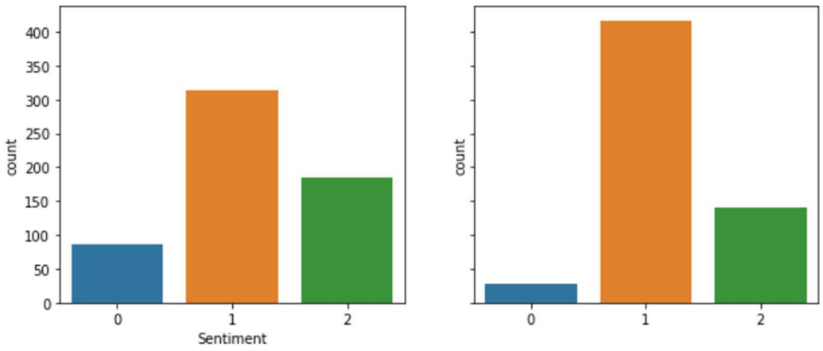


Рисунок 3.12 – Візуалізація результатів: кількість елементів класів.

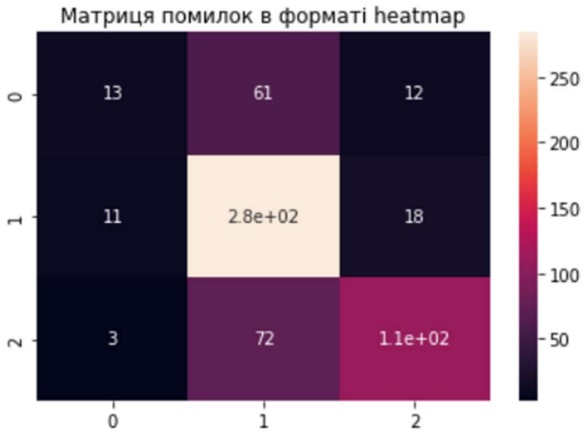


Рисунок 3.13 – Графічне представлення матриці помилок.

Кластеризація. На відміну від класифікації, кластеризація є моделлю навчання без вчителя. Це означає, що кількість та характеристики класів розбиття невідомі та повинні бути визначені. Отже, першим та часто найскладнішим завданням є визначення правильної кількості класів/кластерів.

Для визначення кількості кластерів використаємо метод ліктя. В більш складних задачах, як правило, оптимальна кількість кластерів визначається декількома методами.

Метод ліктя розглядає відсоток дисперсії як функцію кількості кластерів: потрібно вибрати кількість кластерів, щоб додавання іншого кластера не дало кращого моделювання даних. Точніше, якщо побудувати відсоток дисперсії, що пояснюється кластерами, проти кількості кластерів, перші кластери додадуть багато інформації (пояснюють багато дисперсії), але в якийсь момент граничний виграш впаде - падає ефект збільшення кількості кластерів. У цій точці вибирається кількість кластерів, звідси й «критерій ліктя». Цей «лікоть» не завжди можна однозначно ідентифікувати.

Відсоток поясненої дисперсії — це відношення дисперсії між групами до загальної дисперсії, також відомого як F-тест. Невелика варіація цього методу зображує кривизну дисперсії всередині групи.

Залежність середньої суми квадратів відстаней від елементів кластерів до центроїд від кількості кластерів показано на рисунку 3.14.

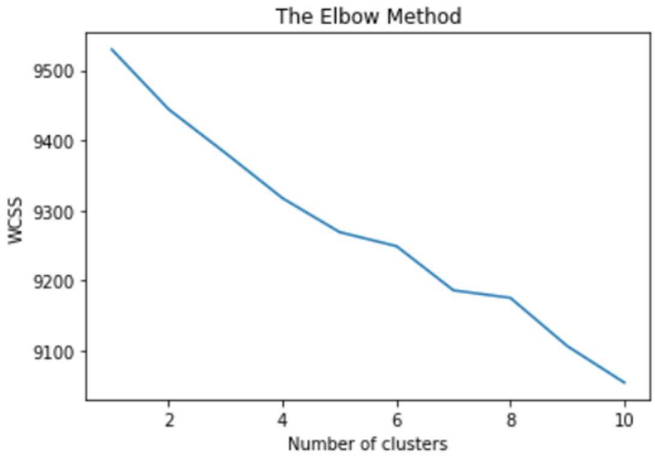


Рисунок 3.14 – Побудована залежність для застосування методу ліктя.

Бачимо згін на кількостях кластерів 5 та 7, тому будуємо кластеризації та порівнюємо результати для обох кількостей кластерів. Як можна бути бачити в наступному підрозділі, при кількості кластерів 7 отримуємо більш чітку тематичну класифікацію.

3.3. Аналіз отриманих результатів.

Проаналізуємо отриману класифікацію. Перш за все треба звернути увагу на той факт, що отриманий клас нейтральних повідомлень більше за його фактичний розмір. Пов'язано це з тим, що використовувалися токени довжиною "1", й багатьом з них помилково надана нейтральна класифікація замість емоційної тональності. Це ж ілюструє й отримана графічна інтерпретація матриці помілок: комірки в другому "стовпчику" мають більші значення аніж інші, розташовані на "недіагональних" місцях.

Для підвищення якості класифікації можна або перейти до бінарної класифікації: позитивна/негативна тональність; або застосовувати біграми чи триграми замість однослівних токенів. Метод збільшення розміру токена потребує відповідного збільшення навчального датасету: частота появи двох чи трьохслівних токенів суттєво менша за появу окремих слів в моделі мішку слів.

Значення метрики ROC-AUC 0.69 також визначає якість класифікації як "задовільну", таку що може бути покращена.

Порівняння ключових слів в кожній з двох отриманих кластеризацій показує, що кластеризація з використання семи кластерів є більш якісною у порівнянні з кількістю кластерів 5.

```
0 : new, govt, win, plan, say, iraqi, claim, nsw, report, baghdad, court, set, hospital, death, mp, crash, woman, qld, wa, fina
l, face, hit, drought, health, boost
1 : police, man, murder, charged, probe, charge, court, face, death, stabbing, search, missing, car, arrest, jailed, investigat
e, fatal, drug, assault, wa, station, accident, crash, sex, hospital
2 : u, iraq, war, anti, say, troop, protest, iraqi, turkey, force, howard, baghdad, attack, pm, missile, protester, soldier, n,
korea, marine, military, denies, rally, killed, blair
3 : world, cup, australia, championship, miss, final, champion, semi, host, win, record, claim, set, ta, bid, injured, eye, sou
th, cross, event, squad, racing, title, red, warne
4 : lead, australia, win, share, open, india, victory, claim, storm, post, final, iraq, field, sri, u, turkey, downer, semi, bl
ue, role, war, adelaide, virus, hit, say
5 : council, security, election, land, seek, fund, welcome, water, change, plan, decision, offer, considers, claim, developmen
t, hold, centre, urged, reject, iraq, cancer, restriction, debate, stock, issue
6 : water, rain, restriction, plan, supply, farmer, drought, boost, despite, low, help, resident, use, hope, group, relief, nee
ded, level, qld, welcome, offer, expected, woe, act, shire
```

Рисунок 3.15 – Ключові слова для семи кластерів.

Окрім вивчення ключових слів, згрупованих навколо центроїд кластерів, тематику кожного з кластерів можна визначити шляхом відсортування новин за отриманою міткою. Вивчимо декілька з отриманих кластерів.

	headline_text	labels
62	greens offer police station alternative	1
72	inquest finds mans death accidental	1
77	irish man arrested over omagh bombing	1
87	man arrested after central qld hijack attempt	1
88	man charged over cooma murder	1
...	...	...
9915	police charge 14 year old over car chase	1
9916	police out to catch drink drivers after hit run	1
9917	police probe truck theft	1
9918	police stage traffic blitz	1
9919	police yet to name traffic victim	1

	headline_text	labels
9	australia is locked into war timetable opp	4
10	australia to contribute 10 million in aid to iraq	4
18	bryant leads lakers to double overtime win	4
85	last minute call hands alinghi big lead	4
212	beckham leads as man u cut down depleted juve	4
...	...	...
9939	sarwan gayle out of first australia test clash	4
9946	soccer australia board under pressure to quit	4
10016	australia leads clare higson trophy golf tourn...	4
10019	australia survives scare in guyana	4
10020	australia will have no large role as iraq	4

Рисунок 3.16 – Вміст кластерів з мітками “1” тап “4”.

Вивчивши зміст кожного з кластерів, призначимо коротку тематичну класифікацію кожному з кластерів

0. Гуманітарна ситуація
1. Поліцейські хроніки
2. Антівійськові протести
3. Спортивні новини
4. Новини Австралії.
5. Передвиборчі дебати
6. Агрономічний сектор

Таким чином отримано кластеризацію корпусу текстів, що є заголовками новин, за тематикою.

ВИСНОВКИ

В рамках представленої роботи розглянуті базові принципи текстової аналітики та конвеєри обробки даних та моделювання у додатках текстової аналітики.

Як правило, додаток текстової аналітики складається з трьох частин: препроцесінга корпусу текстів, параметризації текстового корпусу та застосування моделей машинного навчання до множини параметрів текстового корпусу.

Метою першої ділянки конвеєру обробки є перетворення кожного з текстів корпусу на множину лексичних термів: окремих слів або n-грам та нормалізація отриманих термів – зведення їх до початкової форми.

Друга ділянка конвеєру – параметризація корпусу текстів робить можливим застосування числових математичних моделей до корпусу текстів. В роботі розглянуто усі три найбільш широко застосовувані моделі векторизації текстів та застосовано одну з них, TF-IDF, на практиці.

Третя ділянка конвеєру – застосування математичних моделей. В роботі розглядається класифікація текстів з використанням моделей опорних векторів та кластеризація текстів алгоритмом k-середніх. Перший із згаданих алгоритмів використовується у багатьох додатках текстової аналітики поряд з алгоритмами ієрархічної класифікації. Алгоритм кластеризації k-середніх є простим та наочним алгоритмом кластеризації, який дає відносно непогані результати у багатьох випадках. Разом з алгоритмами розглянуті також метрики оцінки якості застосування відповідних моделей машинного навчання.

В другому та третьому розділі, що відносяться до практичної частини, спроектовано та розроблено програмні додатки текстової аналітики на базі теоретичного розгляду першого розділу.

Додатки використано на відповідних текстових корпусах та проаналізовано результати роботи з використанням типових метрик для застосованих моделей машинного навчання.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.

1. Clifford A. Lynch, “Big data: How do your data grow?” , Nature, vol. 455, no. 7209, вересень, 2008.

2. S. Himanshu. A. Panwar, A. Negi, Text Clustering Algorithms: A Review, International Journal of Computer Applications, червень 2014.

3. Quoc Le, Tomas Mikolov. Distributed Representations of Sentences and Documents, Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2), 2014. P. 1188-1196.

4. Natural Language Toolkit/ NLTK 3.4.5 documentation [Електроний ресурс]. – Режим доступу: https://www.nltk.org/ (дата звернення 28.03.2022).

5. spaCy 3.2 [Електроний ресурс] - Режим доступу: https://www.spacy.io/ (дата звернення 29.03.2022).

6. K. R. Bokka, S. Hora, T. Jain, Deep Learning for Natural Language Processing, Packt publishing ltd., 2019, стор. 357.

7. Ю. Васильєв, Обробка природньої мови Python та spaCy, no starch press, S.-Francisco, 2021, стор 291.

8. С. Рашка, В. Мірджалілі В. - Python та машинне навчання, Packt, 2020, стор. 602.

9. B. Lubanovoch, Introducing Python. O’Reilly, 2016, стор. 480.

10. B. Bangfort, R.Bilbro, T.Ojeda, Applied Text Analysis with Python, O’Reilly, 2019 стор. 368.

11. N. Friedman N., D. Geiger, Goldszmidt M. Bayesian Network Classifiers. Machine Learning, 1997, vol. 29, iss. 2-3, стор. 131–163.

12. V. Sharma Survey of Classification Algorithms and Various Model Selection Methods.

13. S. Kotsiantis, I. D. Zaharakis, P. E. Pintelas, Machine learning: A review of classification and combining techniques.

14. Support vector machine. URL:
https://en.wikipedia.org/wiki/Supportvectormachine (дата звернення 20.04.2022)

15. S. Baik, J. Bala, A Decision Tree Algorithm for Distributed Data Mining: Towards Network Intrusion Detection, Lecture Notes in Computer Science, Volume 3046, стор. 206 – 212.

16. I. Pak, P. L. The, Machine Learning Classifiers: Evaluation of the Performance in Online Reviews.

17. L. Yu, H. Liu, Efficient Feature Selection via Analysis of Relevance and Redundancy, JMLR, жовтень 2017, стор. 1205-1224.

18. Model evaluation, model selection, and algorithm selection in machine learning. URL: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part1.html (дата звернення 04.04.2022).

19. Scoring metric for machine learning method. URL: https://cs.stackexchange.com/questions/76057/scoring-metric-for-machine-learning-method (дата звернення 27.03.2022).

20. S. Kotsiantis, Unsupervised machine learning: A review of clustering techniques. Informatica (Ljubljana). 2007; 31(3), стор. 249–268.

21. 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. URL: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/ (дата звернення 28.03.2023)

22. A Tour of Machine Learning Algorithms. URL: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (дата звернення 29.03.2022)

23. Уес Маккінлі Python та аналіз даних / Пер. з англ. Слінкін А. А. – М.: ДМК Пресс. 2015. С. 482.

24. Олексій Васильєв. Програмування мовою Python. Богдан. 2020. стор. 504.

ДОДАТОК 1. КЛАТЕРИЗАЦІЯ ТЕКСТІВ.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk.stem import WordNetLemmatizer
from sklearn.cluster import KMeans
from nltk.tokenize import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
from sklearn.cluster import KMeans
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
nltk.download('wordnet', quiet=True)
nltk.download('omw-1.4', quiet=True)
```

Out[2]:
```
True
```

Завантаження даних

In [3]:
```
# кількість рядків початквого датасету, що будуть використані в якості датасету для
N_ROWS = 10000
RANDOM_ROWS = False # для створення пояснень виключено можливість випадкового взяття
# читання даних з csv файлу
df = pd.read_csv("abcnews-date-text.csv", usecols =["headline_text"])
# перегляд семплу із завантажених даних
df.sample(10)
```

Out[3]:

	headline_text
454684	nt isolated from rest of nation after
642668	hasler praises churchill winner stewart
581831	warning cyclone may develop later this week
25800	drugs giant injects cash into biotech sector
586487	docklands retailers struggle as wheel sits idle
1061581	nick kyrgios admits its time for a coach after...
676071	gujurat mine start date questioned
605436	bill shorten joins 730
47091	wollongong urged to get behind canadian rugby ...
684364	interview michael maguire

In [4]:
```
# розмір завантаженого датафрейму
df.shape
```

Out[4]:
```
(1103665, 1)
```

In [5]:
```
# взяття частини датафрейму та видалення дублікатів
# в якості альтернативи можна взяти випадкові рядки що збільшить час обробки датафре
data = df.drop_duplicates().sample(N_ROWS) if RANDOM_ROWS else df.drop_duplicates().
# інформація щодо отриманного датафрейму
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 10032
Data columns (total 1 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   headline_text  10000 non-null  object
dtypes: object(1)
memory usage: 156.2+ KB
```

Використання моделі TF-IDF

Як було зазначено в теоретичній частині **TF-IDF** є числовою статистикою, яка призначена для відображення важливості слова для документа в колекції або корпусі. Ця модель часто використовується як ваговий коефіцієнт під час пошуку інформації, аналізу тексту та моделювання користувача. Значення *tf-idf* збільшується пропорційно кількості зустрічей слова в документі і компенсується частотою слова в корпусі, що допомагає налаштувати на те, що деякі слова з'являються частіше. На сьогоднішній день *tf-idf* є однією з найпопулярніших схем зважування термінів; 83% текстових рекомендаційних систем у сфері цифрових бібліотек використовують саме *tf-idf*.

Варіанти моделі "зваження" слів в мішку слів *tf–idf* часто використовуються пошуковими системами як центральний інструмент для оцінки та ранжування релевантності документа за запитом користувача. *tf–idf* можна успішно використовувати для фільтрації стоп-слів у різних предметних областях, включаючи узагальнення та класифікацію тексту.

Одна з найпростіших функцій ранжирування обчислюється шляхом підсумовування tf–idf для кожного терміну запиту, багато складніших функцій ранжирування є варіантами цієї простої моделі.

Препроцесінг датасету

In [6]:
```
# знаки пунктуації - символи, що не повинні бути враховані моделлю
punc = ['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}',"%"]
# стоп - слова - слова, що виключаються із розрахунку коефіцієнтів моделі
# використовується список стоп-слів бібліотеки NLTK для англійської мови у поєднанні
# різні списки стоп слів будуть використанні при побудові різних препроцесорів
stop_words_wo_punc = text.ENGLISH_STOP_WORDS
stop_words_with_punc = text.ENGLISH_STOP_WORDS.union(punc)
# приклади з отриманого списку стоп - слів (оригінальний набір представляє собою тип
list(stop_words_with_punc)[:15]
```

Out[6]:
```
['afterwards',
 'whereby',
 'wherever',
 'himself',
 'its',
 'hasnt',
 'whenever',
 'must',
 'down',
 'fifteen',
 'my',
```

```
'put',
'nowhere',
'nor',
'very']
```

In [7]:
```
# отримання даних датасету у форматі NumPy для навчання моделей Sklearn (Scikit-lear
desc = data['headline_text'].values
```

Перша найпростіша модель

In [8]:
```
# створення та навчання моделі TfIdf. В якості препроцесінгу вкиористовується виключ
vectorizer = TfidfVectorizer(stop_words = stop_words_with_punc)
X = vectorizer.fit_transform(desc)
```

In [9]:
```
word_features = vectorizer.get_feature_names_out()
print(f'Загальна кількість слів у отриманій моделі - {len(word_features)}')
```

```
Загальна кількість слів у отриманій моделі - 9861
```

Розглянемо деякі слайси із списку, що отримано

In [10]:
```
print(word_features[5000:5100])
```

```
['lakes' 'lambie' 'laments' 'land' 'landfill' 'landholder' 'landholders'
 'landing' 'landmine' 'landmines' 'lands' 'landslide' 'langer' 'langmack'
 'language' 'lanka' 'lankan' 'lankans' 'lara' 'large' 'larger' 'larkham'
 'lash' 'lashes' 'lashings' 'lasting' 'late' 'later' 'latest' 'latham'
 'latif' 'latrobe' 'laughing' 'launceston' 'launch' 'launched' 'launches'
 'laundering' 'laureates' 'laureus' 'lavender' 'laverton' 'law' 'lawful'
 'lawnmowers' 'laws' 'lawyer' 'lawyers' 'lax' 'lay' 'lazaridis' 'lazio'
 'lazios' 'lead' 'leader' 'leaderboard' 'leaders' 'leadership' 'leading'
 'leads' 'league' 'leagues' 'leak' 'leaks' 'leaney' 'leap' 'learn'
 'learner' 'learning' 'learns' 'lease' 'leases' 'leave' 'leaves' 'leaving'
 'lebouc' 'lecturer' 'led' 'lee' 'leeds' 'lees' 'left' 'leg' 'legal'
 'legality' 'legend' 'legionella' 'legislation' 'legislative' 'legitimacy'
 'lehmann' 'leicester' 'leisel' 'leisure' 'lemon' 'lend' 'length'
 'leniency' 'lennox' 'lens']
```

learn learner learning learns

laws lawyer lawyers

Яко можна бачити, в перелік слів, що використовуються моделлю, попадає багато однокореневих слів. Для виключення однокореневих слів використаємо стемінг.

Друга та третя моделі: використання стемінгу як частини власного токенізатора, використання стандартного токенізатора

Стемінг – це процес зведення слова до основи, тобто до кореневої форми. Коренева форма не обов'язково є словом сама по собі, але її можна використовувати для створення слів шляхом з'єднання правильного суфікса. Наприклад, слова риба, риба та рибалка перетворюються на рибу, що є правильним словом. З іншого боку, слова «study», «studies» і «studing» походять від «studi», що не є англійським словом.

Токенізація — це розбиття речення на слова та розділові знаки.

In [11]:
```
# використання стеммеру з NLTK
```

```
stemmer = SnowballStemmer('english')
```

In [12]:
```
# токенізатор NLTK
tokenizer = RegexpTokenizer(r'[a-zA-Z\']+')

def tokenize(text):
    return [stemmer.stem(word) for word in tokenizer.tokenize(text.lower())]
```

Td-Idf із стоп словами та токензацією

In [13]:
```
vectorizer2 = TfidfVectorizer(stop_words = stop_words_wo_punc, tokenizer = tokenize)
X2 = vectorizer2.fit_transform(desc)
word_features2 = vectorizer2.get_feature_names_out()
print(f'Загальна кількість слів у отриманій моделі - {len(word_features2)}')
```

```
Загальна кількість слів у отриманій моделі - 6930
```

In [14]:
```
word_features2[1000:1020]
```

Out[14]:
```
array(['carri', 'carrier', 'carrigan', 'carseldin', 'cart', 'cartel',
       'carter', 'cartoonist', 'caryard', 'casa', 'casanova', 'case',
       'cash', 'casino', 'cast', 'castaigned', 'castl', 'castleford',
       'castro', 'casual'], dtype=object)
```

Розглянемо використання альтернативного токенізатора

In [15]:
```
# використовуємо лематізатор. Це дозволяє отримувати початкову форму слів замість ли
lemmatizer = WordNetLemmatizer()

def tokenize_lemmas(text):
    return [lemmatizer.lemmatize(word) for word in tokenizer.tokenize(text.lower())]
```

In [16]:
```
vectorizer2_2 = TfidfVectorizer(stop_words = stop_words_wo_punc, tokenizer = tokeniz
X2 = vectorizer2_2.fit_transform(desc)
word_features2_2 = vectorizer2_2.get_feature_names_out()
print(f'Загальна кількість слів у отриманій моделі - {len(word_features2_2)}')
```

```
Загальна кількість слів у отриманій моделі - 8373
```

In [17]:
```
word_features2_2[1000:1020]
```

Out[17]:
```
array(['buenos', 'buffer', 'buffeted', 'bug', 'buggy', 'build', 'builder',
       'building', 'built', 'bulk', 'bull', 'bulldog', 'bulldozer',
       'bullet', 'bulli', 'bullied', 'bullying', 'bumper', 'bun',
       'bunbury'], dtype=object)
```

In [19]:
```
# обмежимо кількість слів - ознак в наборі. Модель використає лише найважливіши з сл
vectorizer3 = TfidfVectorizer(stop_words = stop_words_with_punc, tokenizer = tokeniz
X3 = vectorizer3.fit_transform(desc)
words = vectorizer3.get_feature_names_out()
print(f'Загальна кількість слів у отриманій моделі - {len(words)}')
```

```
Загальна кількість слів у отриманій моделі - 700
```

In [20]:
```
print(words[:20])
```

```
['aboriginal' 'abuse' 'accident' 'accused' 'accuses' 'act' 'action' 'ad'
```

```
'address' 'adelaide' 'advance' 'afl' 'aged' 'agreement' 'ahead' 'aid'
'aim' 'air' 'airport' 'ajax']
```

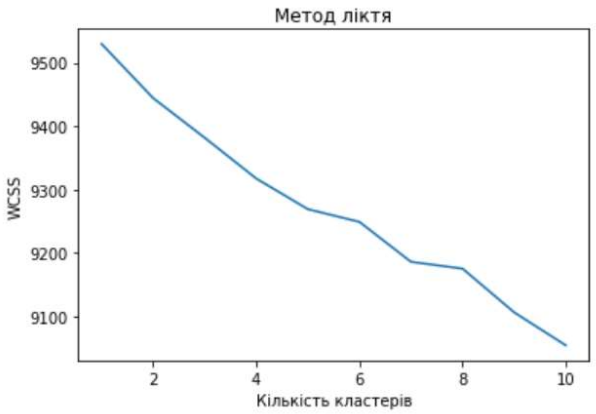
K-means clustering

Використовуємо метод ліктя для пошуку оптимальної кількості кластерів

Метод ліктя розглядає відсоток дисперсії як функцію кількості кластерів: потрібно вибрати кількість кластерів, щоб додавання іншого кластера не дало кращого моделювання даних. Точніше, якщо побудувати відсоток дисперсії, що пояснюється кластерами, проти кількості кластерів, перші кластери додадуть багато інформації (пояснюють багато дисперсії), але в якийсь момент граничний виграш впаде - падає ефект збільшення кількості кластерів. У цій точці вибирається кількість кластерів, звідси й «критерій ліктя». Цей «лікоть» не завжди можна однозначно ідентифікувати. Відсоток поясненої дисперсії — це відношення дисперсії між групами до загальної дисперсії, також відомого як F-тест. Невелика варіація цього методу зображує кривизну дисперсії всередині групи.

In [21]:

```
wcss = [] # within-cluster sums of squares - сума квадратів відстаней елементів клас
for i in range(1,11):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_stat
    kmeans.fit(X3)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('Метод ліктя')
plt.xlabel('Кількість кластерів')
plt.ylabel('WCSS')
plt.show()
```



Як можна бачити, згенеровано більше ніж один згиб. Тому сформуємо та проаналізуємо результати для обох згинів: 5 та 7.

Кластеризація: кількість кластерів 5.

In [22]:

```
kmeans = KMeans(n_clusters = 5, n_init = 20)
kmeans.fit(X3)
common_words = kmeans.cluster_centers_.argsort()[:,-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))
```

```
0 : war, govt, say, claim, world, cup, report, anti, protest, vic, protester, rejec
t, pm, nsw, iraqi, sa, howard, australia, rally, plan, ta, nt, urged, student, qld
```

```
1 : police, new, win, council, plan, water, iraqi, rain, nsw, death, baghdad, court,
set, crash, woman, lead, mp, face, qld, hope, final, hospital, boost, hit, wa
2 : man, charged, murder, court, face, charge, police, jailed, stabbing, missing, de
ath, hospital, accident, search, car, assault, plane, arrested, attack, injured, dy,
sex, u, guilty, child
3 : u, iraqi, troop, war, baghdad, turkey, force, attack, soldier, korea, marine, n,
say, military, killed, saddam, plane, trade, claim, helicopter, death, guard, missil
e, al, airport
4 : iraq, u, war, say, missile, troop, howard, denies, blair, bush, post, pm, britis
h, force, attack, blix, turkey, aid, resolution, tv, kuwait, coalition, downer, sold
ier, uk
```

Кластеризація: кількість кластерів 7.

In [23]:

```
kmeans = KMeans(n_clusters = 7, n_init = 20)
kmeans.fit(X3)
common_words = kmeans.cluster_centers_.argsort()[:,-1:-26:-1]
for num, centroid in enumerate(common_words):
    print(str(num) + ' : ' + ', '.join(words[word] for word in centroid))
```

```
0 : new, govt, win, plan, say, iraqi, claim, nsw, report, baghdad, court, set, hospi
tal, death, mp, crash, woman, qld, wa, final, face, hit, drought, health, boost
1 : police, man, murder, charged, probe, charge, court, face, death, stabbing, searc
h, missing, car, arrest, jailed, investigate, fatal, drug, assault, wa, station, acc
ident, crash, sex, hospital
2 : u, iraq, war, anti, say, troop, protest, iraqi, turkey, force, howard, baghdad,
attack, pm, missile, protester, soldier, n, korea, marine, military, denies, rally,
killed, blair
3 : world, cup, australia, championship, miss, final, champion, semi, host, win, rec
ord, claim, set, ta, bid, injured, eye, south, cross, event, squad, racing, title, r
ed, warne
4 : lead, australia, win, share, open, india, victory, claim, storm, post, final, ir
aq, field, sri, u, turkey, downer, semi, blue, role, war, adelaide, virus, hit, say
5 : council, security, election, land, seek, fund, welcome, water, change, plan, dec
ision, offer, considers, claim, development, hold, centre, urged, reject, iraq, canc
er, restriction, debate, stock, issue
6 : water, rain, restriction, plan, supply, farmer, drought, boost, despite, low, he
lp, resident, use, hope, group, relief, needed, level, qld, welcome, offer, expecte
d, woe, act, shire
```

In [31]:

```
data.loc[data.labels == 4]
```

Out[31]:

	headline_text	labels
9	australia is locked into war timetable opp	4
10	australia to contribute 10 million in aid to iraq	4
18	bryant leads lakers to double overtime win	4
85	last minute call hands alinghi big lead	4
212	beckham leads as man u cut down depleted juve	4
...	...	...
9939	sarwan gayle out of first australia test clash	4
9946	soccer australia board under pressure to quit	4
10016	australia leads clare higson trophy golf tourn...	4
10019	australia survives scare in guyana	4
10020	australia will have no large role as iraq	4

ДОДАТОК 2. КЛАСИФІКАЦІЯ ТЕКСТІВ.

In [1]:

```
import pandas as pd
import pandas_profiling
import re
from nltk.corpus  import stopwords
from nltk.stem import WordNetLemmatizer
import seaborn as sns
import matplotlib.pyplot as plt
import sys
import warnings
warnings.filterwarnings('ignore')

from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from wordcloud  import WordCloud
from textwrap import wrap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import normalize
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.metrics import roc_auc_score
from PIL import Image

import nltk
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('omw-1.4', quiet=True)

%matplotlib inline
```

Завантаження, підготовка даних, EDA

Завантаження даних

In [2]:

```
data=pd.read_csv('sentiment_data.csv').drop_duplicates().reset_index(drop=True) # завантаже
df=data.copy()
df.sample(7) # довільна вибірка з датасету
```

Out[2]:

	Sentence	Sentiment
1212	Infection worries prompt regulatory reviews of...	negative
308	Lieksaare Oy has earlier been regarded under t...	neutral
896	Meanwhile , minority shareholders , expecting ...	neutral
3365	Taking a cue from the playbook of the East Dil...	neutral
5085	RSA Insurance Hires Towergate's Egan as Chief ...	neutral
1397	Below are unaudited consolidated results for A...	neutral
1253	Aker Yards Finland will begin using Chinese su...	neutral

In [3]:

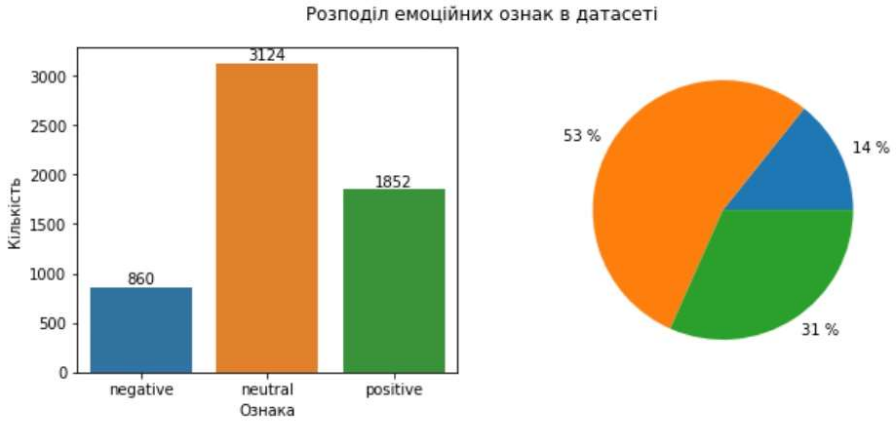
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5836 entries, 0 to 5835
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Sentence   5836 non-null   object
 1   Sentiment  5836 non-null   object
dtypes: object(2)
memory usage: 91.3+ KB
```

Розподіл класів емоційної оцінки датасету

In [4]:

```
fig, ax = plt.subplots(1, 2, figsize=(10,4))
sns.countplot(df['Sentiment'].sort_values(), ax=ax[0])
ax[0].set_xlabel('Ознака')
ax[0].set_ylabel('Кількість')
ax[0].bar_label(ax[0].containers[0])
ax[1].pie(vs:=list(map(int,(100 * normalize(df['Sentiment'].value_counts().sort_index().val
          labels=list(map(lambda x: f'{x} %', vs)))
plt.suptitle('Розподіл емоційних ознак в датасеті')
plt.show()
```



Підготовка даних: очищення речень

In [5]:

```
# метод очищення текстів датасету
def cleaning(df1: str) -> str:
    lowered=df1.lower()    # переведення у нижній регістр
    removed=re.sub(r'[^a-z]',' ',lowered)  # видалення знаків пунктуації
    splitted=removed.split(' ')   # розділення речень на слова
    # лематизація та видалення стоп-слів
    df1= [WordNetLemmatizer().lemmatize(word) for word in splitted if word not in stopwords
    df1=' '.join(df1) # зворотнє поєднання речень
    return(df1)
```

In [6]:

```
# застосування створенного методу очищення речень, перегляд модифікованого датасету
df['Sentence']=df['Sentence'].apply(cleaning)
df.head()
```

Out[6]:

	Sentence	Sentiment
0	geosolutions technology leverage benefon gps ...	positive
1	esi low bk real possibility	negative
2	last quarter componenta net sale doubl...	positive
3	according finnish russian chamber commerce m...	neutral
4	swedish buyout firm sold remaining percen...	neutral

Генерація хмари слів

In [7]:

```
# поєднання усіхї речень в корпус
corpus=' '.join(df['Sentence'])
# генерація облака слів
word_cloud=WordCloud().generate(corpus)
img=word_cloud.to_image()
img
```

Out[7]:

Виконання класифікації

Створення матриці термів

In [8]:

```
tfidf=TfidfVectorizer(max_features=3000) # використання моделі Tf-Idf
data=tfidf.fit_transform(df['Sentence'])
df_dtm = pd.DataFrame(data.toarray(), columns=tfidf.get_feature_names_out())
df_dtm.head()
```

Out[8]:

	aa	aal	aaland	aalto	aaltonen	aapl	aaron	aava	aazhang	ab	...	zone	zoo	zp	zsl
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0

5 rows × 9297 columns

In [9]:

```
df_dtm["Sentiment"]=df['Sentiment']
df_dtm.shape
```

Out[9]:

```
(5836, 9298)
```

Використання LabelEncoder для числового кодування отриманної матриці моделі Tf-Idf

Виконуємо кодування міток цільового об'єкта, оскільки цільовий об'єкт має категоріальний тип даних, а наша модель ML розуміє лише числові ознаки.

In [10]:

```
le=LabelEncoder()
df_dtm['Sentiment']=le.fit_transform(df_dtm['Sentiment'])
```

In [11]:

```
df_dtm['Sentiment'].value_counts()
```

Out[11]:

```
1    3124
2    1852
0     860
Name: Sentiment, dtype: int64
```

In [12]:

```
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
le_name_mapping # мітки, присвоєні класам для подальшого порівняння із початковим датасетом
```

Out[12]:

```
{'negative': 0, 'neutral': 1, 'positive': 2}
```

Побудова класифікатора

In [13]:

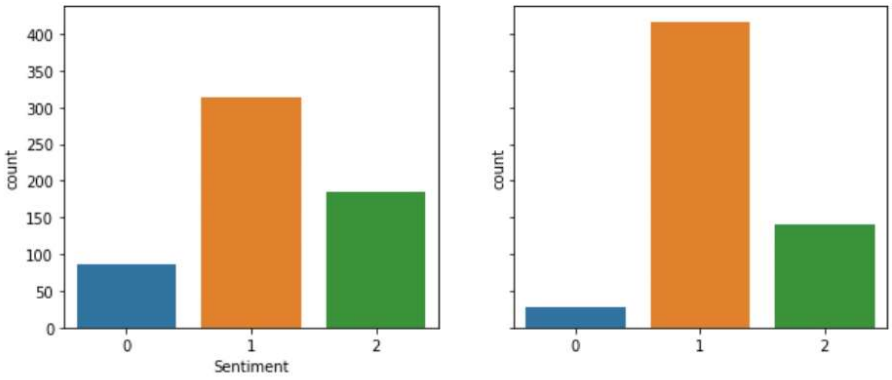
```
# розподіл датасету на тренувальний та тестувальний
X_train, X_test, y_train, y_test = train_test_split(df_dtm.drop('Sentiment',axis=1),
                                                    df_dtm['Sentiment'], test_size=0.1,
                                                    stratify=df_dtm['Sentiment'])
```

In [17]:

```
# тренування SVM моделі
svc=SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred=svc.predict(X_test)
```

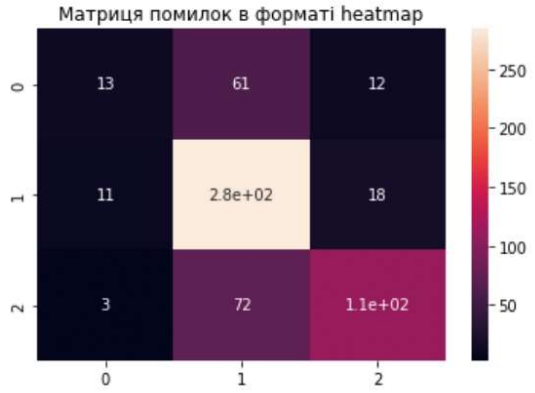
In [18]:

```
# порівняння тестових та отриманних в результаті роботи моделі міток класів
fig, ax = plt.subplots(1, 2, sharey=True, figsize=(10, 4))
ax[0] = sns.countplot(y_test, ax=ax[0])
ax[1] = sns.countplot(y_pred, ax=ax[1])
plt.show()
```



In [19]:

```
# побудова матриці помилок
sns.heatmap(confusion_matrix(y_test,y_pred), annot=True)
plt.title("Матриця помилок в форматі heatmap")
plt.show()
```



In [20]:

```
print(f'Площа під кривою помилок {svc.score(X_test,y_test):.5f}')
```

```
Площа під кривою помилок 0.69692
```