ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підвищення якості попередньої обробки даних в текстах низької якості»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Новицький Сергій
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-64
Сергій НОВИЦЬКИЙ

Керівник: Оксана ЗОЛОТУХІНА
к.т.н., доц.

Рецензент: _____
Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Новицькому Сергію Сергійовичу

1. Тема кваліфікаційної роботи: «Підвищення якості попередньої обробки даних в текстах низької якості»

керівник кваліфікаційної роботи Оксана Золотухіна к.т.н., доц.

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методи попередньої обробки текстових даних, методи та алгоритми збагачення текстових даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1.Аналіз методів та технологій опрацювання неструктурованої інформації.

2. Розробка методики попередньої обробки даних в текстах низ якості.

3. Реалізація та апробація методики.

5. Перелік графічного матеріалу: *презентація*

1. Види шумів в текстах низької якості.
2. Методи попередньої обробки текстових даних.
3. Схема попередньої обробки тексту.
4. Схема токенізації тексту.
5. Схема виправлення орфографічних помилок в тексті
5. Векторизація тексту методом TF-IDF.
6. Приклад обробки тексту.
7. Результати моделювання.

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Дослідження методів, алгоритмів та технологій обробки неструктурованих даних	06.11-12.11.23	
3	Аналіз видів шумів в неструктурованих даних	13.11-19.11.23	
4	Дослідження методів обробки шумів в текстах низької якості	20.11-26.11.23	
5	Розробка методики попередньої обробки даних в текстах низької якості	27.11-03.12.23	
6	Застосування методів обробки текстів низької якості	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Сергій НОВИЦЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 79 стор., 8 табл., 11 рис., 21 джерел.

Мета роботи – зменшення розміру вектору представлення тексту низької якості за рахунок усунення шуму та надлишкової інформації методами попередньої обробки текстів.

Об'єкт дослідження – процес попередньої обробки даних у текстах низької якості.

Предмет дослідження – методи і технології обробки текстових даних.

Короткий зміст роботи:

Дипломна робота присвячена питанню підвищення якості обробки текстових даних, з особливим фокусом на роботі з неструктурованою інформацією. Проведено аналіз методів та технологій обробки неструктурованої інформації, розглянуто різні ступені структурованості даних, проблематика обробки текстових даних за допомогою методів машинного навчання, а також застосування технологій обробки природної мови, в тому числі, з застосуванням методів машинного навчання. Розроблено загальну схему алгоритму методики попередньої обробки даних в текстах низької якості. Методика комбінує методи токенізації, нормалізації токенів, аналізу видів шумів, а також методи їх видалення і виправлення. Особливу увагу приділено моделі Noisy Channel, обробці стоп-слів, векторизації тексту, а також опрацюванню скорочень, абревіатур, вульгаризмів і жаргонізмів. Результати апробації демонструють скорочення вектору тексту на коротких повідомленнях до 46%, на середніх – до 55%, на довгих – до 51% у порівнянні з кількістю токенів вхідного тексту.

КЛЮЧОВІ СЛОВА: ОБРОБКА ТЕКСТОВИХ ДАНИХ, НЕСТРУКТУРОВАНА ІНФОРМАЦІЯ, ТЕХНОЛОГІЇ ОБРОБКИ ПРИРОДНОЇ МОВИ, УСУНЕННЯ ШУМУ І ПОМИЛОК, ВЕКТОРИЗАЦІЯ ТЕКСТУ.

ABSTRACT

Text part of the master's qualification work: 79 pages, 11 pictures, 8 table, 21 sources.

The purpose of the work – reducing the size of the low-quality text representation vector by removing noise and redundant information using text preprocessing methods.

Object of research – processing data in low-quality texts.

Subject of research – methods and technologies of text data processing.

Summary of the work:

The thesis is dedicated to the issue of improving the quality of text data processing, with a special focus on working with unstructured information.

The main part of the work consists of three sections, introduction and conclusions. The first chapter analyzes the methods and technologies of text data processing, including the study of various machine learning methods and natural language processing technologies. The second chapter is devoted to the development of techniques for preprocessing data in low-quality texts, including tokenization, token normalization, noise removal, and other aspects. The third section describes the implementation and testing of the developed methodology, as well as presents the simulation results.

An analysis of methods and technologies for processing unstructured information was conducted, including the examination of various degrees of data structuring, the challenges of processing text data using machine learning methods, as well as the application of natural language processing technologies, including machine learning methods. A general scheme of the algorithm for preliminary data processing in low-quality texts has been developed. The methodology combines methods of tokenization, token normalization, noise type analysis, as well as methods for their removal and correction. Special attention is given to the Noisy Channel model, processing of stop

words, text vectorization, and the handling of abbreviations, vulgarisms, and slang. The results of the testing demonstrate a reduction in the text vector by 46% for short messages, 55% for medium-length texts, and 51% for long texts, compared to the number of tokens in the input text.

KEYWORDS: TEXT DATA PROCESSING, UNSTRUCTURED INFORMATION, NATURAL, LANGUAGE PROCESSING TECHNOLOGIES, NOISE AND ERROR ELIMINATION, TEXT VECTORIZATION.

ЗМІСТ

ВСТУП

У зв'язку з великим обсягом інформації, що потребує обробки організаціями, існує актуальна необхідність у спеціалістах, які здатні ефективно впроваджувати вказаний процес. Застосування відповідних алгоритмів дозволяє компаніям оптимізувати процеси в маркетингу, вдосконалювати сервіс, що призводить до збільшення клієнтської бази та доходів. Вдосконалення процесів статистики та аналізу продукту на основі відгуків клієнтів допомагає уникнути помилок у майбутньому. Застосування подібних підходів спроможне зекономити ресурси та збільшити прибутковість. Загальна висновок полягає в тому, що методи аналізу неструктурованої інформації приводять до вдосконалення бізнес-процесів та збільшення прибутку.

Сучасний світ важко уявити без наявності інформації, яка є всеосяжною і поширеною, починаючи від новин у газетах і закінчуючи телебаченням чи Інтернетом. Інформацію можна представити у різних форматах: зображення, відео, музика, текст у статтях чи книгах, листи або електронні повідомлення. Дані поділяються на два основних типи: структуровані та неструктуровані. Структуровані дані можуть бути представлені у вигляді Excel-таблиць, баз даних або категоризації відео та музики. Це те, що можна легко розмістити в певному порядку. Неструктуровані дані, як відео, музика, листи, статті та соціальні мережі, не вписуються в чіткі рамки баз даних чи категорій. Обробка та робота з такою інформацією виявляються важчими завданнями. За деякими джерелами, до 80-90% інформації є неструктурованою, що підкреслює великий обсяг корисної інформації, вивчення та систематизація якої є складним завданням.

У ідеальному сценарії для організацій та компаній всі дані мають бути структурованими - чітко класифікованими за категоріями, мітками, стовпцями та полями, злагоджено та зібраними по всій організації. Такий підхід робить дані більш доступними та полегшує їх ефективне та швидке використання. Економія часу у такому контексті є критично важливою, оскільки дозволяє оптимізувати робочі процеси та підвищувати продуктивність.

Об’єкт дослідження – процес попередньої обробки даних у текстах низької якості.

Предмет дослідження – методи і технології обробки текстових даних.

Мета роботи – зменшення розміру вектору представлення тексту низької якості за рахунок усунення шуму та надлишкової інформації методами попередньої обробки текстів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) проаналізувати наявні методи і технології обробки текстової інформації;

2) проаналізувати види текстів та процеси зменшення текстового шуму;

3) розробити загальну схему алгоритму зменшення векторного представлення тексту;

4) реалізувати алгоритм на основі розробленої схеми;

5) провести аналіз роботи алгоритму для текстів різного розміру та ступеня і видів зашумленості та визначити рівень зменшення векторного представлення.

Методи дослідження: математичні – статистичні методи; емпірико-теоретичні: абстрагування, аналіз, синтез, емпіричні методи, евристики, візуалізація; методи проектування та розробки програмного забезпечення; методи обробки текстів природною мовою.

Основні частина роботи складається з трьох розділів, вступу та висновків. У першому розділі проводиться аналіз методів та технологій обробки текстових даних, включаючи вивчення різних методів машинного навчання та технологій обробки природної мови. Другий розділ присвячений розробці методики попередньої обробки даних у текстах низької якості, включаючи токенізацію, нормалізацію токенів, видалення шуму та інші аспекти. Третій розділ описує реалізацію та апробацію розробленої методики, а також подає результати моделювання.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ОПРАЦЮВАННЯ НЕСТРУКТУРОВАНОЇ ІНФОРМАЦІЇ

1.1 Аналіз даних за ступенем структурованості

Дані, у своїй суті, виконують неабияку роль як важливий ресурс для функціонування будь-якого бізнесу, і ця роль, безумовно, вражає своєю широкою гамою форматів, що охоплюють спектр від жорстко структурованих реляційних баз даних до різних повідомлень у месенджерах. Усі ці дані, розподілені в різні форми, визначаються як структуровані та неструктуровані залежно від рівня попередньої обробки перед зберіганням у даних, відомих як сховище інформації.

Структуровані дані, визначені й сформатовані перед вводом їх у сховище даних, що також називається схемою для запису, є виявом точної організації. Найяскравішим прикладом таких данних слугує реляційна база даних, де дані точно структуровані і додані у визначені поля, такі як ім'я, прізвище, e-mail, адреса та інші. Це надає можливість ефективно використовувати SQL для формулювання запитів, оновлення і управління реляційними базами даних, а також для моделювання схеми баз даних та встановлення систем контролю доступу.

Плюси використання структурованих даних виявляються у трьох основних перевагах:

- Легко використовується алгоритмами: Однією з ключових переваг структурованих даних є їхній легкий доступність для обробки за допомогою алгоритмів машинного навчання. Чіткий та організований характер цих даних сприяє простоті маніпуляції та формулюванню запитань.
- Легко використовується алгоритмами бізнес-користувачами: Іншою перевагою є можливість простого використання структурованих даних бізнес-користувачами з базовим розумінням теми. Немає необхідності в

глибокому розумінні різних типів даних чи їх взаємозв'язків, що надає бізнес-користувачам можливість самообслуговування.

– Легкість в підтримці: Структуровані дані легко піддаються різноманітним операціям, таким як сортування, фільтрування та оновлення.

Мінуси використання структурованих даних пов'язані з їхньою обмеженою гнучкістю:

Заздалегідь визначена мета обмежує використання: Визначеність даних у схемі запису, хоча і дає переваги, обмежує їхню гнучкість та можливості використання.

Обмежені можливості зберігання: Структуровані дані, як правило, зберігаються в сховищах даних, що мають жорсткі схеми. Будь-яка зміна вимог вимагає оновлення всіх структурованих даних, що призводить до великих витрат ресурсів і часу. Хмарне сховище даних може зменшити частину цих витрат, але витрати на технічне обслуговування все одно залишаються суттєвими. Це може призводити до значних втрат доходів та бюджету компанії.

Структуровані дані служать основою для різноманітних систем управління запасами та банкоматів, і можуть бути створені як людьми, так і машинами. Ці дані, зазвичай представлені у форматах, таких як JSON, XML та інші, знаходять своє місце збереження в хмарних сховищах, базах даних або різноманітних файлах (наприклад, XLS, CSV).

Протилежність становлять неструктуровані дані, які зберігаються у своєму первинному форматі, не пройшовши обробки до моменту їхнього використання. Ці дані, які можна знайти у електронній пошті, соціальних мережах, презентаціях, чатах, зображеннях, аудіо та відео, є предметом значного інтересу.

Переваги неструктурованих даних визначаються кількома ключовими аспектами:

– Свобода рідного формату: Не визначені заздалегідь, неструктуровані дані зберігаються у своєму власному форматі, що збільшує їхню

універсальність і адаптивність. Це дозволяє використовувати різноманітні формати файлів та реагувати на зміни вимог без обмежень.

- Швидкість накопичення: Неструктуровані дані швидко накопичуються, оскільки вони не вимагають попередньої структуризації, що полегшує їх збір і аналіз.
- Велика кількість даних: Ці дані можуть зберігатися в хмарних «озерах», надаючи масштабованість та зменшуючи витрати на зберігання через модель оплати за використання.

Незважаючи на переваги, існують певні недоліки неструктурованих даних, включаючи вимогу до експертів в галузі науки про дані та необхідність спеціалізованих інструментів для їх обробки. Такі дані часто мають якісний, а не кількісний характер, і можуть використовуватися для визначення ефективності маркетингових кампаній чи аналізу тенденцій у соціальних мережах та веб-сайтах.

Оцінка різниці між структурованими та неструктурованими даними може бути виражена через кілька ключових питань:

Хто може використовувати?

- Структуровані дані: Легкі у доступі, можуть бути використані широким колом користувачів.
- Неструктуровані дані: Вимагають спеціальних навичок та експертизи для ефективного використання.

Який тип даних підтримується?

- Структуровані дані: Обмежені відомими типами даних, часто використовують стандартні формати.
- Неструктуровані дані: Підтримують конгломеративні, різноманітні типи даних, такі як тексти, аудіо, відео тощо.

Де можна зберігати?

- Структуровані дані: Зазвичай зберігаються в хмарних сховищах або базах даних.

– Неструктуровані дані: Можна зберігати практично в будь-якому форматі, включаючи тексти, аудіо та відео.

Як можна зберігати?

– Структуровані дані: Зберігаються відомими форматами(наприклад JSON), заздалегідь визначені та легко розпізнавані системами.
– Неструктуровані дані: Зберігаються у своєму нативному форматі, забезпечуючи більшу гнучкість та адаптивність.

Структуровані дані володіють високою специфічністю та зберігаються в строго визначеному форматі, у той час як неструктуровані дані представляють собою агрегат різноманітних типів даних, які зберігаються у своїх природних форматах. Це означає, що структуровані дані використовують переваги схеми при їхньому занесенні, тоді як неструктуровані дані використовують схему при їхньому читанні. Втім, неструктуровані дані виявляють значну перевагу в сфері бізнесу, оскільки вони несуть велику кількість цінної інформації, яку, хоча зазвичай складніше аналізувати, проте приносить набагато більше користі.

Ці аспекти вказують на те, що структуровані дані частіше використовуються там, де потрібна чіткість та однорідність, тоді як неструктуровані дані дозволяють більш велику різноманітність та гнучкість у роботі із змішаними типами інформації.

1.2 Проблематика обробки текстових даних за допомогою методів машинного навчання

Тенденції щодо того, чи отримують компанії більше структурованих чи неструктурованих даних, можуть змінюватися залежно від галузі та конкретних потреб організацій. Однак можна сказати, що багато компаній сьогодні працюють із різноманітними видами даних, включаючи як структуровані, так і неструктуровані.

Наприклад, згідно з дослідженням "Big Data Executive Survey 2017" від NewVantage Partners, компанії все більше використовують неструктуровані дані. Зазначено, що 95,4% великих компаній зазначили, що вони мають ініціативи, спрямовані на використання неструктурованих даних.

Аналіз тексту за допомогою методів машинного навчання стикається із рядом викликів.

Складність обробки:

– Конвертація текстового вмісту у формат, придатний для обробки комп'ютером, вимагає виконання кількох кроків. Наприклад, вирішення завдання класифікації тексту передбачає збір даних, виявлення ключових слів, визначення кількості класів, групування даних за цими класами та математичне описання цих процесів. Це представляє виклик як з точки зору інтелекту, так і з точки зору людських/фінансових/часових ресурсів.

Концептуальні труднощі:

– Комп'ютери не розуміють концепцій, що стоять за словами, ускладнюючи роботу з омографами. Програмісти повинні розробити ефективні інструменти для визначення неоднозначностей слів, наприклад в такому словосполученні, як "замок під замком".

Розуміння культурного контексту:

– Розуміння людської мови передбачає врахування емоцій, але для комп'ютерів однією з найскладніших емоцій є сарказм. При врахуванні багатозначності, те ж саме значення може виражатися різними словами в різних культурах, таких як сленг чи місцеві варіанти. Наприклад, для британця "джемпер" означає те саме, що і для американця "светр". Комп'ютерні програми повинні мати культурний контекст та досвід для ефективного взаємодії з носіями мови, які використовують менш традиційні форми мовлення.

Для подолання зазначених проблем ефективними інструментами є технологія обробки природної мови (NLP) та використання метаданих. Ці методи

надають кілька переваг у підготовці текстових даних для подальшого використання методами машинного навчання.

Обробка та "чистка" тексту:

– NLP використовується для обробки текстового вмісту, що спрощує завдання алгоритмам машинного навчання. Вона дозволяє виявляти та враховувати ключові аспекти тексту, такі як синоніми, антоніми, або інші лінгвістичні варіації.

Використання метаданих:

– Метадані, такі як інформація про час, автора, місце тощо, можуть слугувати додатковою інформацією для алгоритмів машинного навчання. Ця інформація може виявитися важливою при аналізі тексту та допомагати в уникненні неоднозначностей.

Спрощення процесу:

– Застосування NLP та метаданих спрощує підготовку даних, роблячи її більш ефективною та швидкою. Вони визначають ключові аспекти тексту та надають алгоритмам зрозуміліші вхідні дані.

Ці технології і підходи допомагають підвищити ефективність обробки текстових даних, зменшуючи складність та забезпечуючи адекватні вхідні дані для методів машинного навчання.

1.3 Технології обробки природної мови

Natural Language Processing (NLP), або обробка натуральної мови, грає важливу роль у роботі з неструктурованими текстовими даними. Оскільки ці дані часто мають нативний формат у вигляді людської мови, методи NLP дозволяють їх ефективно обробляти перед проведенням аналізу. Розглянемо основні та важливі методи обробки натуральної мови:

Bag of Words (Мішок слів):

– Цей метод конвертує текст у множину слів без врахування порядку. Кожне слово розглядається як окремий елемент, що дозволяє ефективно враховувати частоту вживання слів у тексті.

Токенізація:

– Розбиває текст на окремі елементи, так звані токени (слова або фрази), для подальшої обробки. Це дозволяє враховувати кожне слово окремо.

Вилучення зайвих слів (Stop Words Removal):

– Це процес усунення артиклів, прийменників та інших частотно вживаних слів, які не несуть суттєвої інформації для аналізу тексту. Більшість методів аналізу неструктурованих текстових даних розпочинаються з обробки базового списку стоп-слів, які потім можуть доповнюватися іншими ключовими словами, залежно від контексту та конкретних завдань аналізу. Цей процес сприяє вдосконаленню точності та ефективності аналізу, виключаючи менш значущі елементи мови.

Стемінг (Stemming):

– Процес обрізання афіксів (приставок перед словом або суфіксів після слова) для отримання кореневого слова. Цей процес сприяє уніфікації різних форм одного слова, що дозволяє ефективніше групувати та аналізувати текстові дані. Наприклад, слова "гітара" та "гітарист" після stemming можуть обидві відображатися як "гітар". Це полегшує подальший аналіз та витягування суттєвої інформації з тексту.

Лематизація (Lemmatization):

– Етап обробки текстових даних, де слова зводяться до їхньої словникової (базової) форми, відомої як "лема". Цей процес допомагає уніфікувати слова до їхнього базового вигляду та спрощує аналіз тексту. Наприклад, слова "навчання" і "навчене" піддаються лемматизації та перетворюються в "навчати". Це допомагає уникнути зайвої різноманітності форм слова, а також ураховує контекст вживання, що робить лемматизацію більш точною та контекстуалізованою.

Моделювання тем (Topic Modeling):

– Інструмент аналізу тексту, спрямований на виділення тем та груп слів, що пов'язані з різними концепціями. Частота появи конкретних слів у документі служить індикатором теми, що вони відображають, та їх значущості. Наприклад, якщо слова "економіка" та "ВВП" частіше зустрічаються у тексті, можна припустити, що він стосується економічних аспектів. Аналогічно, якщо терміни "судноплавство" та "логістика" виявляються більш поширеними, текст ймовірно має зв'язок із логістикою. Такий підхід дозволяє визначити відсоткове співвідношення тем у документі, що полегшує аналіз та інтерпретацію його змісту. Ці методи грають ключову роль у підготовці текстових даних для подальшого використання в алгоритмах машинного навчання.

Підсумовуючи, варто відзначити, що всі наведені методи призначені для збагачення текстового матеріалу, роблячи його більш інформативним та цінним.

Обробка природної мови взаємодіє з аналізом неструктурованих текстових даних та текстовою аналітикою. Цей процес включає в себе підрахунок, групування та класифікацію слів для вилучення структури та значення із обширних обсягів даних. Аналіз тексту застосовується для розуміння текстового вмісту та отримання нових змінних із необробленого тексту. Ці змінні можуть бути візуалізовані, відфільтровані або використані як вхідні дані для моделей прогнозування чи інших статичних методів.

Обробка природної мови є невід’ємним етапом обробки та аналізу неструктурованих обсягів даних, що надходять щоденно.

1.4 Визначення додаткової інформації через метадані

Метадані – це дані, які визначають і надають інформацію про інші дані, виконуючи важливу роль в управлінні, зберіганні та аналізі неструктурованих даних .

Зручним прикладом таких даних є інформація, пов'язана з фотографіями. При зйомці за допомогою цифрової камери чи смартфону кожне фото має прикріплені метадані, такі як дата, час, назва файлу та геолокація. У блозі кожна публікація включає метадані, такі як заголовок, автор, URL-адреса, дата публікації, теги, категорія та інше. Веб-сторінки містять метадані, такі як заголовок, URL-адреса та опис.

Навіть за стандартними полями, можна визначити додаткові метадані відповідно до контексту. Таким чином, метадані полегшують подальший пошук та аналіз.

Оскільки немає єдиної галузевої стандартизації для метаданих, кожен розробник програмного забезпечення може визначити власні стандарти. Ефективне використання метаданих сприяє організації та автоматизації процесу отримання даних.

1.5 Огляд методів машинного навчання

Машинне навчання спрямоване на передбачення і відповідь на питання "Що далі?" за допомогою методів, які допомагають аналізувати та прогнозувати.

Два ключові компоненти машинного навчання - це дані і алгоритми.

Дані: Це інформація, яка вводиться в систему машинного навчання. Наприклад, при створенні алгоритму для передбачення погоди, необхідно коректно структурувати дані, такі як температурні показники, швидкість вітру тощо.

Алгоритми: Це методи, які використовуються для обробки введених даних. Вони аналізують, представляють, та заповнюють пробіли, щоб визначити закономірності у введених даних.

Формула Дані + Алгоритми = Машинне навчання відображає сутність цього процесу, який використовується в багатьох платформах та ресурсах.

Машинне навчання дозволяє глибоко аналізувати закономірності у вхідних даних, наприклад, визначати взаємозв'язки між температурними діапазонами чи швидкістю вітру. Алгоритми вирішують завдання аналізу, представлення даних та заповнення прогалин в інформації.

1.6 Метод Байєса

Naive Bayes - це метод класифікації, заснований на теоремі Байєса, з припущенням незалежності серед предикатів. Простими словами, класифікатор Naive Bayes припускає, що наявність об'єкта в класі не пов'язана з наявністю будь-якого іншого об'єкта [3].

Наприклад, банан може бути визнаний як фрукт, якщо він жовтий чи зелений і має форму дуги. Навіть якщо ці ознаки взаємозалежні чи залежать від інших ознак, усі ці властивості, вважаються незалежними одна від одної, що сприяє збільшенню ймовірності того, що цей фрукт є яблуком, і тому він називається "наївним" (Naive).

Модель Naive Bayes проста в побудові і особливо корисна для обробки великих наборів даних. На додаток до простоти, відомо, що Naive Bayes може перевершувати навіть дуже складні методи класифікації.

Переваги цього алгоритму включають:

- Простота та ефективність: Метод наївного Байєса - це відносно простий алгоритм, який легко реалізувати та розуміти. Незважаючи на його наївні припущення, він може працювати ефективно для багатьох класифікаційних завдань.
- Добре працює з великими об'ємами даних: Метод наївного Байєса може бути дуже ефективним для великих об'ємів даних та високорозмірних наборів ознак.

- Висока швидкість навчання та класифікації: Цей метод може працювати дуже швидко, оскільки йому не потрібно вивчати складні взаємозв'язки між ознаками.
- Добре працює з категоріальними ознаками: Він добре вирішує проблему класифікації з категоріальними ознаками та дискретними даними.

Мінуси метода Баєса:

- Наївні припущення про незалежність: Одним з основних недоліків цього методу є наївні припущення про незалежність між ознаками. У реальних даних це припущення може бути досить обмеженим.
- Чутливість до неправильностей в даних: Метод наївного Байєса може бути чутливим до неправильностей чи викидів у даних. Він може надавати неточні результати, якщо в наборі даних є шум чи аномалії.
- Не враховує взаємозв'язки між ознаками: Так як він припускає незалежність ознак, метод не враховує взаємозв'язки між ними, що може впливати на точність прогнозування.
- Проблема зі спільним винятком: Якщо в наборі даних є ознаки, які завжди спільно виникають разом, метод наївного Байєса може давати неправильні ймовірності.
- Проблема нульових ймовірностей: Якщо в тестовому наборі даних зустрічається нова ознака, яку відсутня в навчальному наборі, ймовірність для такої ознаки може бути рівною нулю, що призводить до невизначеності.

Не зважаючи на ці недоліки, метод наївного Байєса залишається популярним в багатьох областях через свою простоту та ефективність для певних видів даних.

1.7 Метод опорних векторів

Метод опорних векторів (SVM) є одним із найпопулярніших методів класифікації, а також вирішує регресійні задачі. Однак, на даний момент нас цікавлять його властивості для класифікації.

Основною метою алгоритму є створення "ідеальної" лінії чи межі рішення, яка може ефективно розділити n-вимірний простір на класи, дозволяючи легко визначити положення нової точки даних в конкретній категорії. Цю межу часто називають гіперплощиною.

SVM визначає крайні точки (вектори), які допомагають створити гіперплощину. Ці крайні випадки називаються опорними векторами, відси і назва методу - опорні вектори.

Переваги методу включають:

- Ефективність у випадках з чітким розділенням класів.
- Ефективність у великих просторах.
- Ефективність у випадках, коли розмірність простору перевищує кількість зразків.
- Використання підмножини навчальних точок у функції прийняття рішень (опорні вектори), що сприяє ефективності в роботі з пам'яттю.
- Недоліки методу:
- Не ефективний при великих обсягах даних, оскільки час навчання високий.
- Може показати гірші результати у випадках, коли в наборі даних багато шуму, тобто коли цільові класи перекриваються.

SVM не надає прямих оцінок ймовірності, вони розраховуються за допомогою витратних обчислень, таких як п'ятикратна перехресна перевірка.

1.8 Дерево прийняття рішень

Дерево прийняття рішень – це алгоритм, який вирішує завдання на основі ієрархічних правил, представлених у вигляді вузлів і листків. Вузли містять правила, які потрібно вирішити, та проводиться перевірка відповідності прикладів цим правилам за будь-яким атрибутом навчальної множини. У найпростішому випадку множина прикладів, що потрапила в вузол, розбивається на дві підмножини відповідно до заданих правил. Далі до кожної підмножини застосовується відповідне правило, і цей процес рекурсивно повторюється до досягнення умови завершення алгоритму. В останньому вузлі не відбувається подальша перевірка та розбиття, і цей вузол стає листком.

Переваги алгоритму:

- Однією з переваг дерев рішень є їхня легкість читання та інтерпретації без необхідності в статистичних знаннях. Наприклад, використовуючи дерева рішень для подання демографічної інформації про клієнтів, працівники відділу маркетингу можуть аналізувати графічне представлення даних без глибокого розуміння статистики. Ці дані також можуть надавати цінну інформацію щодо ймовірностей, витрат та альтернатив стратегій маркетингу.
- У порівнянні з іншими методами прийняття рішень, дерева рішень вимагають менше зусиль для підготовки даних. Важливою умовою залишається наявність відомостей для створення нових змінних та передбачення цільової змінної. Крім того, користувачі можуть створювати класифікації даних без проведення складних обчислень. В складних сценаріях можна комбінувати дерева рішень з іншими методами.
- Іншою перевагою дерев рішень є менша потреба в очищенні даних після створення змінних. Випадки відсутніх значень та відхилень мають менш важливе значення для даних дерева рішень.

Недоліки алгоритму:

- Одним з обмежень дерев рішень є їхня значна нестійкість порівняно з іншими методами прийняття рішень. Навіть невелика зміна вихідних даних може призвести до суттєвих змін у структурі дерева прийняття рішень, що може призводити до результату, відмінного від очікуваного у звичайних умовах. Вплив таких змін можна зменшити використовуючи алгоритми машинного навчання, такі як алгоритми посилення (boosting) та складання пакетів (bagging).
- Зменшена ефективність у прогнозуванні результату для неперервної змінної. Крім того, дерева рішень менше ефективні у випадках, коли головною метою є передбачення результату для неперервної змінної. Це пояснюється тим, що дерева рішень, зазвичай, втрачають інформацію при класифікації змінних у декілька категорій.

1.9 Лінійний дискримінаційний аналіз

Лінійний дискримінаційний аналіз (ЛДА) представляє собою лінійний класифікатор прийняття рішень, який формується шляхом підбору умовних "щільностей" класів до даних і використання правил Байєса.

Цей метод є одним із найпопулярніших для моделювання тематики. Простий приклад його застосування включає кожен документ, який складається з різних слів, і кожну тему, яка також містить різні слова, що до неї відносяться. Основна мета методу - визначити теми, які відносяться до документу, на основі його словесного змісту.

ЛДА також відомий як метод зменшення розмірності, розроблений Рональдом А. Фішером у 1936 році під назвою "Лінійний дискримінант" або "Дискримінантний аналіз Фішера". Початковий лінійний дискримінант описувався як двокласовий метод, пізніше К.Р. Рао узагальнив його для роботи з

багатьма класами у вигляді аналізу множинної дискримінації. Всі ці підходи отримали загальну назву "лінійний дискримінаційний аналіз".

ЛДА є методом класифікації, включеним до сімейства конкурентних моделей машинного навчання. Використовується у різних областях, таких як розпізнавання зображень та прогнозований аналіз у маркетингу.

Переваги:

- Простий, швидкий та портативний алгоритм, який все ще може перевершувати деякі алгоритми (наприклад, логістична регресія), коли виконуються його припущення.

Недоліки:

- Вимагає нормального розподілу характеристик предикторів.
- Іноді не є оптимальним для змінних з кількома категоріями.

1.10 Випадковий ліс

Випадкові ліси у машинному навчанні представляють собою комплексний метод, що використовує ансамбель "дерев рішень" для завдань класифікації, регресії та інших операцій. Ці ліси володіють швидкістю та гнучкістю, що робить їх надійним інструментом для роботи з великими обсягами даних. Вони є розширенням класифікаційних та регресійних дерев, які були описані раніше.

Ансамблеве навчання описує модель, що генерує прогнози, комбінуючи внесок окремих моделей. Ці моделі ансамблю, зазвичай, виявляються більш гнучкими з меншою упередженістю та меншою дисперсією. Ансамблеве навчання використовує два популярні методи:

Багатократна випадковість (багато-бутстреп): Кожне окреме дерево випадковим чином обирає підмножину даних і тренується на цій підмножині, що призводить до різних дерев.

Підвищення (boosting): Кожне окреме дерево чи модель навчається на помилках попередньої моделі та вдосконалюється.

Важливо, що "випадковий ліс" має мати значну кількість дерев, зазвичай від 64 до 128.

Різниця між випадковим лісом та деревом рішень:

Випадковий ліс - це ансамбль, що складається з декількох дерев рішень. Дерево рішень будується на всьому наборі даних, використовуючи всі ознаки, тоді як випадковий ліс випадковим чином вибирає підмножину спостережень та конкретних ознак для кожного дерева, а потім усереднює їх результати .

Плюси:

- Ефективно працює з нелінійними даними.
- Висока ефективність на великих наборах даних.
- Зазвичай забезпечує кращу точність порівняно з іншими алгоритмами класифікації.

Мінуси:

- Показує упередженість при роботі з категоріальними змінними.
- Може бути повільним під час процесу навчання.

РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ ПОПЕРЕДНЬОЇ ОБРОБКИ ДАНИХ В ТЕКСТАХ НИЗЬКОЇ ЯКОСТІ

2.1 Загальна схема алгоритму попередньої обробки текстів низької якості

Для досягнення мети роботи були обрані процеси токенізації, перевірка та виправлення орфографічних помилок, пошук і видалення стоп слів, опрацювання токенів, що не є словами природної мови, розкриття скорочень та абревіатур, обробка вульгарних слів, обробка жаргонних слів також до схеми входить блок векторизації для, які входять до загальної схеми обробки даних що представлені на рисунку 2.1.

Токенізація - процес перетворення вхідного тексту на токени, які можуть бути словами, фразами або іншими елементами тексту.

Перевірка на орфографічні помилки - цей крок включає виправлення помилок у словах для забезпечення їх правильного написання.

Пошук і видалення стоп слів - видалення загальновживаних слів, які зазвичай не несуть значної інформації (наприклад, "і", "але", "якщо"). Текст проходить подвійну обробку на видалення стоп слів.

Опрацювання токенів, що не є словами природної мови - це може включати видалення або обробку чисел, символів та інших елементів, які не є частиною природної мови.

Розкриття скорочень та абревіатур - перетворення скорочень та абревіатур на їх повні форми з допомогою словника скорочень та абревіатур.

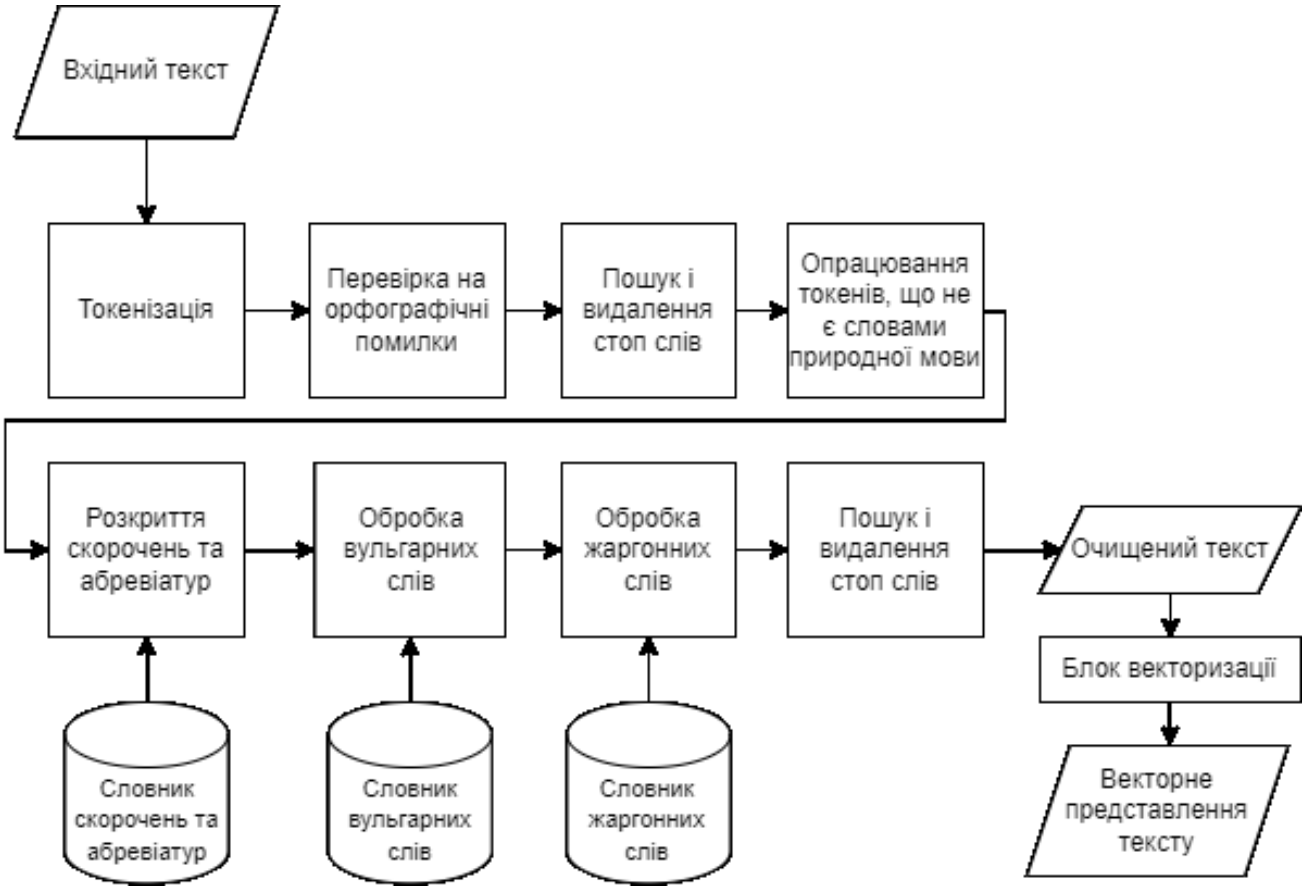


Рис. 2.1 – Загальна схема обробки текстів низької якості

Обробка вульгарних слів - виявлення та обробка (наприклад, видалення або заміна) вульгарної лексики з використанням словника вульгарних слів.

Обробка жаргонних слів - подібно до обробки вульгарних слів, цей крок включає виправлення або заміну слів, які є жаргоном, і може використовувати словник жаргонних слів.

Блок векторизації - перетворення очищеного тексту на векторне представлення, яке може бути використане в машинному навчанні.

На схемі вхідними даними є вхідний текст який може мати повідомлення, статті, відгуку, новини, або будь-який інший вигляд неструктурованого тексту.

Вихідні данні алгоритму є вихідний текст, що після попередньої обробки проходить процес векторизації. Векторне представлення тексту має вигляд вектору де слова відсортовані за алфавітом і значення компонента є його частота використання в тексті.

2.2 Токенізація

Токенізація тексту на слова — це процес розділення письмового тексту на мінімальні одиниці, які мають семантичне значення, звані токенами. У контексті розбиття тексту на слова, токени зазвичай відповідають словам, числам та іншим словоформам. Далі описані етапи процесу токенізації.

Визначення кордонів слів: Спочатку система визначає, де в тексті починається і закінчується кожне слово. Це робиться шляхом визначення розділових знаків, пробілів та інших символів пунктуації, які слугують кордонами між словами.

Розділення тексту: Після визначення кордонів текст розбивається на токени шляхом розділення за визначеними кордонами. Наприклад, речення "Це — приклад." буде розділене на токени "Це", "—", "приклад", і ".".

Видалення пунктуації: Символи пунктуації видаляються, щоб залишити лише слова.

Нормалізація : Це крок перетворення всіх літер на нижній регістр для забезпечення уніформності, видалення або заміну спеціальних символів і цифр.

Обробка винятків: Деякі слова або символи можуть вимагати особливого врахування. Наприклад, "New York" слід токенізувати як одне слово, а не два різних токени.

Після завершення цих етапів, текст буде перетворений на послідовність токенів, які будуть використовуватись для подальшої обробки.

Токенізація служить основою для безлічі додатків у цифровій сфері, дозволяючи машинам обробляти та розуміти величезні обсяги текстових даних. Розбиваючи текст на керовані фрагменти, токенізація сприяє більш ефективному та точному аналізу даних. Ось кілька відомих випадків використання, коли токенізація відіграє ключову роль:

– Пошукових. Коли ви вводите запит у пошукову систему, як-от Google, вона використовує токенізацію для аналізу введених даних. Ця розбивка

допомагає механізму просіяти мільярди документів, щоб представити вам найбільш релевантні результати.

- Машинний переклад. Такі інструменти, як Google Translate, використовують токенізацію для сегментації речень мовою оригіналу. Після токенізації ці сегменти можна перекласти, а потім реконструювати цільовою мовою, гарантуючи, що переклад збереже оригінальний контекст.
- Розпізнавання мови. Голосові помічники, такі як Siri або Alexa, значною мірою покладаються на токенізацію. Коли ви ставите запитання або команду, вимовлені вами слова спочатку перетворюються на текст. Потім цей текст токенізується, що дозволяє системі обробляти ваш запит і діяти відповідно до нього.

Навігація в тонкощах людської мови з її нюансами та неоднозначностями представляє набір унікальних викликів для токенізації. Ось більш глибоке занурення в деякі з цих перешкод:

- Двозначності. Мова за своєю суттю неоднозначна. Розгляньмо речення: «Літаки, що літають, можуть бути небезпечними». Залежно від того, як це токенізується та інтерпретується, це може означати, що акт пілотування літаків є ризикованим або що літаки в польоті становлять небезпеку. Такі двозначності можуть призвести до дуже різних тлумачень.
- Мови без чітких кордонів. Деякі мови, такі як китайська або японська, не мають чітких пробілів між словами, що робить токенізацію більш складним завданням. Визначення того, де закінчується одне слово і починається інше, може бути значною проблемою в таких мовах.
- Робота зі спеціальними символами. Тексти часто містять більше, ніж просто слова. Адреси електронної пошти, URL-адреси або спеціальні символи може бути складно токенізувати. Наприклад, чи слід розглядати

"john.doe@email.com" як один токен або розділяти його на крапку або символ "@"?

Для вирішення таких неоднозначностей були розроблені передові методи токенізації, такі як контекстно-залежні токенізатори, такі як токенізатор BERT. Для мов без чітких меж слів токенізація символів або підслів може запропонувати більш ефективний підхід. Крім того, попередньо визначені правила та регулярні вирази можуть допомогти в обробці спеціальних символів і складних рядків.

2.3 Нормалізація токенів

Стемінг - це процес визначення основи слова для заданого вихідного слова. Основа слова не обов'язково збігається з морфологічним коренем слова.

Задача визначення основи слова є давньою проблемою в галузі комп'ютерних наук. Перша публікація з цього питання датується 1968 роком. Стемінг використовується в пошукових системах для розширення пошукового запиту користувача і є частиною процесу нормалізації тексту.

Існує кілька типів алгоритмів стемінгу, які відрізняються за продуктивністю, точністю і способами подолання певних проблем стемінгу.

Простий стеммер використовує таблицю пошуку для визначення флексійних форм. Цей підхід вигідний завдяки своїй простоті, швидкості та легкості обробки винятків. Проте, він має свої обмеження. Недоліками є необхідність явно перераховувати всі можливі флексійні форми в таблиці, що призводить до ігнорування нових або невідомих слів, навіть якщо вони мали б правильні флексійні варіанти (наприклад, "Apples" ~ "appl"). Крім того, розмір таблиці пошуку може стати проблемою, особливо для мов зі складною морфологією, такої як турецька, де для кожного кореня може існувати сотні можливих флексійних форм.

Зазвичай, таблиці пошуку, що використовуються в стеммерах, створюються напівавтоматично. Наприклад, для англійського слова "run" автоматично

генеруються форми "running", "runs", "runned" і "runly". Правда, останні дві форми є граматично припустимими, але не використовуються в звичайному англійській мові.

Алгоритм пошуку може використовувати попередню частинну розмітку, щоб уникнути такого виду помилок лематизації, коли різні слова відносяться до однієї леми (overstemming).

Алгоритми усічення закінчень дійсно не використовують довідникову таблицю флексійних форм і відношень між коренем і формою. Замість цього, зазвичай зберігається менший список "правил", який використовується алгоритмами з урахуванням форми слова для пошуку його основи. Деякі приклади таких правил виглядають так:

- Якщо слово закінчується на 'ed', видалити 'ed'.
- Якщо слово закінчується на 'ing', видалити 'ing'.
- Якщо слово закінчується на 'ly', видалити 'ly'.

Ці правила допомагають алгоритму визначити основу слова шляхом видалення певних суфіксів. Цей підхід дозволяє адаптувати алгоритм до різних мов та вирішувати проблеми стемінгу без потреби в таблицях флексійних форм.

Алгоритми усічення закінчень насправді ефективніші, ніж алгоритми повного перебору. Для розробки таких алгоритмів потрібен програміст, який має гарне розуміння лінгвістики, зокрема морфології, і може кодувати "правила усічення". Алгоритми усічення закінчень не є ефективними для виключних випадків (наприклад, 'ran' і 'run'). Рішення, отримані за допомогою алгоритмів усічення закінчень, обмежуються тими частинами мови, які мають добре відомі закінчення і суфікси, з деякими винятками. Це серйозне обмеження, оскільки не всі частини мови мають добре сформульований набір правил. Лематизація намагається зняти це обмеження.

Алгоритми усічення префіксів також можуть бути реалізовані. Однак не в усіх мовах слова мають приставки і суфікси.

Алгоритми усічення закінчень можуть різнитися у вихідних результатах з числа кількох причин. Однією з таких причин є особливість алгоритму: чи повинно слово на виході алгоритму бути реальним словом в даній мові. Деякі підходи не вимагають наявності слова в відповідному мовному словнику. Крім того, деякі алгоритми ведуть базу даних всіх відомих морфологічних коренів, які існують як реальні слова. Ці алгоритми перевіряють наявність терміна в базі даних для прийняття рішення. Зазвичай, якщо термін не знайдений, застосовуються альтернативні дії. Ці альтернативні дії можуть використовувати кілька інших критеріїв для прийняття рішення. Несуществующий термін може послужити застосуванню альтернативних правил усічення.

Може бути так, що два або більше правил усічення застосовуються до одного і того ж вхідного терміна, що створює невизначеність щодо того, яке правило використовувати. Алгоритм може визначити пріоритет виконання таких правил (за допомогою людини або стохастичним способом). Або алгоритм може відхилити одне з правил, якщо воно призводить до неіснуючого терміна, в той час як інше - ні. Наприклад, для англійського терміна "friendlies" алгоритм може визначити суфікс "ies", застосувати відповідне правило і отримати результат "friendl". Термін "friendl", ймовірно, не буде знайдений у словнику, і, отже, це правило буде відхилено.

Одним з поліпшень алгоритмів усічення закінчень є використання підстановки суфіксів і закінчень. Схоже до правила усічення, правило підстановки замінює суфікс або закінчення альтернативним. Наприклад, може існувати правило, яке замінює "ies" на "y". Оскільки правила усічення призводять до неіснуючого терміна в словнику, правила підстановки вирішують цю проблему. У цьому прикладі "friendlies" перетворюється на "friendly" замість "friendl".

Зазвичай застосування цих правил має циклічний або рекурсивний характер. Після першого застосування правила підстановки для даного прикладу алгоритм вибирає наступне правило для терміна "friendly", в результаті чого визначається та використовується правило усічення суфікса "ly". Таким чином, терм "friendlies"

за допомогою правила підстановки перетворюється на "friendly", і після застосування правила усічення стає "friend".

Цей приклад демонструє різницю між методом, заснованим на правилах, і методом "грубої сили". За допомогою повного перебору алгоритм буде шукати терм "friendlies" серед сотень тисяч флективних словоформ і, в ідеалі, знайде відповідну основу "friend". У методі, заснованому на правилах, відбувається послідовне виконання правил, що призводить до того самого результату. Ймовірно, метод на основі правил буде швидшим.

У лінгвістиці найпоширенішими термінами для позначення аффіксів є суфікс і префікс. У додаток до підходів, які обробляють суфікси або закінчення, деякі з них також обробляють префікси. Наприклад, для англійського слова "indefinitely" цей метод визначить, що конструкція "in", яка розташована на початку слова, є префіксом і може бути вилучена для отримання основи слова. Багато з методів, згаданих вище, також використовують цей підхід. Наприклад, алгоритм усічення закінчень може обробляти як суфікси та закінчення, так і префікси, тоді він буде мати іншу назву і використовувати цей підхід.

Лемматизація є більш складним підходом до вирішення завдання визначення основи слова. Щоб зрозуміти, як працює лемматизація, потрібно знати, як створюються різні форми слова. Більшість слів змінюються, коли вони використовуються у різних граматичних формах. Кінець слова замінюється граматичним закінченням, і це призводить до нової форми вихідного слова. Лемматизація виконує зворотне перетворення: замінює граматичне закінчення на суфікс або закінчення початкової форми[4].

Також лемматизація включає визначення частини мови слова та застосування різних правил нормалізації для кожної частини мови. Визначення частини мови відбувається перед спробою знайти основу, оскільки для деяких мов правила стемінгу залежать від частини мови даного слова.

Цей підхід сильно залежить від точного визначення лексичної категорії (частини мови). Незважаючи на те, що існує часткове співпадіння між правилами

нормалізації для деяких лексичних категорій, неправильне вказування категорії чи нездатність визначити правильну категорію може знищити перевагу цього підходу над алгоритмом усічення закінчень. Основна ідея полягає в тому, що якщо стеммер має можливість отримувати більше інформації про оброблюване слово, то він може застосовувати більш точні правила нормалізації.

Ripple-down rules (прокладывание правил сверху вниз) в першу чергу були розроблені для набуття знань і обслуговування систем, що базуються на правилах. У цьому підході знання набуваються на основі поточного контексту і додаються поступово. Правила створюються для класифікації випадків, які відповідають певному контексту.

На відміну від стандартних правил класифікації, Ripple-down Rules використовує винятки з існуючих правил, тому зміни стосуються лише контексту правила і не впливають на інші. Інструменти набуття знань допомагають знайти і змінити конфліктуючі правила.

Стохастичні алгоритми пов'язані з ймовірнісним визначенням кореневої форми слова. Ці алгоритми будують ймовірнісну модель і навчаються за допомогою таблиці відповідності між кореневими та флексивними формами. Зазвичай ця модель представлена у вигляді складних лінгвістичних правил, аналогічних правилам, які використовуються в алгоритмах усічення закінчень та лемматизації. Стемінг виконується шляхом введення змінених форм для навчання моделі і генерації кореневої форми відповідно до внутрішнього набору правил моделі. Однак рішення, пов'язані з вибором найбільш відповідного правила чи послідовності правил, а також вибором кореневої форми слова, приймаються на основі того, що результатуюче правильне слово матиме найвищу ймовірність (невірні слова матимуть найменшу ймовірність).

Деякі алгоритми лемматизації мають стохастичний характер в тому сенсі, що слово може належати декільком частинам мови з різною ймовірністю. Ці алгоритми також можуть враховувати оточуючі слова, які називаються контекстом. Контекстно-вільні граматики не враховують жодної додаткової

інформації. У будь-якому випадку після призначення ймовірності кожній можливій частині мови, обирається частина мови з більшою ймовірністю, а також відповідні їй правила для отримання нормалізованої форми.

2.4 Аналіз видів шумів. Методи та алгоритми видалення та виправлення орфографічних шумів

"Шуми в тексті" зазвичай вказують на непотрібну, випадкову або завадливу інформацію, яка може перешкоджати чіткому розумінню тексту. Термін "шуми" використовується в різних контекстах та має кілька значень:

– непотрібна інформація - у тексті може бути велика кількість непотрібної, зайвої інформації, яка не несе суттєвого змісту та може відволікати читача від головної думки;

– помилки та неточності - шум в тексті може виникнути через помилки у написанні, граматичні або стилістичні неточності, які роблять текст менш зрозумілим;

– завадлива інформація - завадливий шум може включати в себе відповіді на запитання, які не стосуються теми, або інші види інформації, які вносять непорозуміння чи переплутують читача;

– електронний шум - у великих обсягах текстової інформації, також може бути "електронний шум", тобто випадкова або несуттєва інформація, яка може виникнути під час передачі, зберігання або обробки даних.

Для зменшення шумів у тексті важливо використовувати чітку мову, перевіряти текст на граматичні та стилістичні помилки, а також старатися уникати введення несуттєвої інформації. У контексті обробки природної мови (NLP), розробка алгоритмів для виявлення та видалення шумів в тексті є одним із завдань, оскільки це допомагає поліпшити точність обробки текстової інформації.

Щоб досягти мети проекту, а саме зменшення вектору представлення текстів будуть видалятись та оброблятись такі текстові шуми та

використовуватись наступні методи:

- Орфографічні шуми - використання інструментів автоматичної корекції, які можуть виправляти орфографічні помилки в тексті;
- Синтаксичні шуми - використання алгоритмів граматичного аналізу для виявлення та виправлення синтаксичних помилок;
- Семантичні шуми - використання методів NLP, щоб аналізувати контекст та забезпечити більш точні варіанти значень слів чи фраз;
- Лексичні шуми - використання техніки лематизації та стемінгу для перетворення слова до її базової форми, що дозволяє обробляти слова з різними формами однаково;
- Логічний шум – використання алгоритмів для аналізу логічної структури тексту та виявлення невірних висловлень;
- Шум в інформаційних даних - використання фільтрів або алгоритми для виявлення та вилучення непотрібної або шумової інформації.

Слід зауважити що використання цих методів не є універсальним для всіх видів тексту в залежності від контексту. Якщо текст або тексти мають певну специфіку то є необхідність в попередній підготовці, а саме наповнення словників та вибір конкретної мови словників для. Якщо текст має великий розмір, то є сенс розбиття його на частини, щоб далі обробляти його послідовно. Детальний опис шумів представлені в таблиці 2.1.

Таблиця 2.1

Опис шумів

Вид шуму	**Опис**
Орфографічний шум	Опечатки: Невірно написані слова через помилки при наборі. Помилки в написанні: Невірно використані слова чи неправильно сформульовані фрази.

Продовження таблиці 2.1 – Опис шумів

Вид шуму	**Опис**
Синтаксичний шум	Граматичні помилки: Неправильна граматика або некоректний синтаксис. Неправильна структура речення: Речення, які можуть бути важко зрозуміти через некоректну структуру. Використання токенів що не є словами природної мови: числа, спецсимволи тощо.
Семантичний шум	Неясність: Фрази, які можуть бути двозначними чи неповніми у своєму значенні. Наявність скорочень чи абревіатур.
Лексичний шум	Використання вульгарної лексики: Неприйнятна мова чи слова, які можуть викликати обурення. Неформальна лексика: Використання слів, які не відповідають формальному стилю.
Логічний шум	Відсутність чітких або послідовних логічних зв'язків між ідеями в тексті.
Шум в інформаційних даних	Невірна інформація: Дані, які невірно представлені чи неперевірені. Відсутність доказів: Виборче використання інформації без підтримки доказами.

Виправлення або видалення орфографічних шумів зазвичай включає в себе застосування автоматизованих методів перевірки правопису. Методи та алгоритми видалення та виправлення орфографічних шумів описані нижче.

- Словники та словникові перевірки - використання словників для порівняння кожного слова в тексті із списком припустимих слів. Слова, які не знайдені в словнику, можуть бути визначені як орфографічні помилки.
- Методи розпізнавання тексту (OCR) - якщо маємо справу з текстом зображення, можна використовувати методи OCR для переведення зображення в текст і потім застосувати перевірку правопису.
- Використання регулярних виразів - регулярні вирази можна використовувати для виявлення шаблонів, які вказують на можливі орфографічні помилки, такі як подвійні букви, неправильні комбінації тощо.
- Методи машинного навчання - використання алгоритмів машинного навчання для навчання моделі виявлення орфографічних помилок на основі великого обсягу текстових даних.
- Контекстуальний аналіз - врахування контексту слова у реченні для виявлення орфографічних помилок, які можуть бути визначені тільки враховуючи слова, які оточують його.
- Системи автоматичної корекції - використання програм, які автоматично коригують орфографічні помилки в тексті. Вони можуть використовувати різні підходи, включаючи словники, правила та машинне навчання.

Ці методи можуть використовуватися окремо або комбінуватися для досягнення більш високої точності виправлення орфографічних шумів. Зазвичай, ефективний підхід - це використання комплексу методів для найкращих результатів.

Для досягнення мети проекту було обрано метод словника в якості основного методу через такі переваги:

- Висока точність для стандартного тексту - У випадках, коли текст стандартний і використовує загальновживані слова, словники можуть

забезпечити високу точність виправлення орфографічних помилок. Багато стандартних текстів включаються в словники, що дозволяє ефективно виявляти неправильно написані слова.

- Широке використання - Метод словників є універсальним і може застосовуватися до різних мов та текових жанрів. Відомі словники можуть включати слова з різних галузей та тематик.
- Швидкість обробки - В порівнянні з деякими більш складними методами, такими як машинне навчання, використання словників може бути швидше і менш витратним за рахунок менших обчислювальних витрат.
- Добре пристосований для чистого тексту - Використання словникових перевірок ефективно для чистого тексту, де головна проблема - це окремі орфографічні помилки, а не неточності в синтаксисі чи семантиці.
- Зручність для користувача - Деякі програми автоматичної корекції тексту використовують словники, що робить їх доступними та зручними для широкого кола користувачів без глибоких знань в галузі обробки природної мови.

Хоча метод словників має свої переваги, варто враховувати, що він може не бути стільки ефективним у випадках з специфічною лексикою, новими словами, діалектами чи тематичними виразами, які не включені в стандартний словник.

Для порівняння слів в тексті зі словами в словнику було обрано концепцію Noisy channel.

2.5 Модель Noisy Channel

Модель Noisy Channel (шумовий канал) є концепцією, яка використовується в обробці природної мови та інших областях для вирішення завдань, таких як машинний переклад, автоматична корекція тексту, розпізнавання мови, тощо.

Основна ідея моделі Noisy Channel базується на визначенні ймовірностей та штучному введенні "шуму" або помилок в процес передачі інформації через канал. Зазвичай, це моделюється за допомогою теорії інформації та ймовірнісних моделей.

У моделі шумного каналу ми уявляємо, що поверхнева форма, яку ми бачимо, насправді є "викривленою" формою оригінального слова, яке пройшло через шумний канал. Декодер проходить кожну гіпотезу через модель цього каналу і вибирає слово, яке найкраще відповідає поверхневому шумному слову.

Інтуїція моделі шумного каналу (див. рис. 2.2) полягає в тому, щоб розглядати помилкове написане слово так, ніби правильно написане слово було "викривлено", пройшовши через шумний комунікаційний канал.

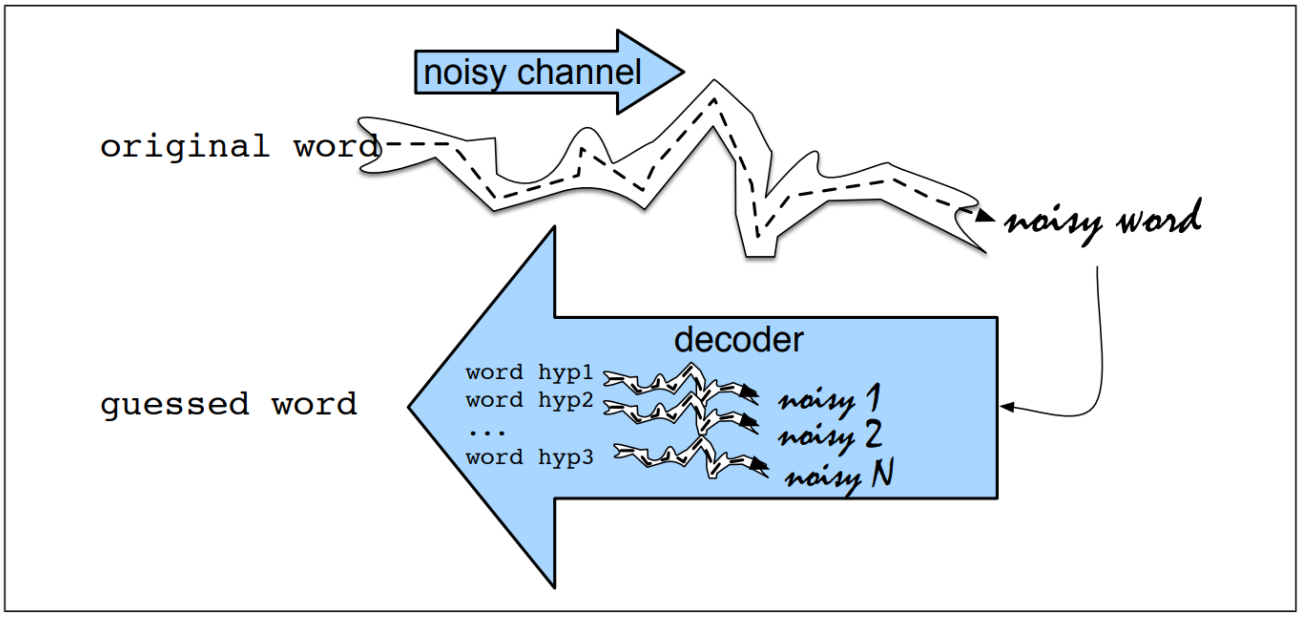


Рис. 2.2 – Модель шумового каналу

Цей канал вводить "шум" у формі заміщень або інших змін літер, що робить важким визначення "істинного" слова. Таким чином, нашою метою є побудова моделі цього каналу. Знаючи цю модель, ми потім визначаємо істинне слово, проходячи кожне слово мови через нашу модель шумного каналу і бачимо, яке найбільше відповідає помилковому написаному слову.

Ця модель шумного каналу є своєрідним байєсівським висновком. Ми маємо спостереження x (невірно написане слово), і наше завдання - знайти слово w, яке згенерувало це невірно написане слово. З усіх можливих слів в словнику V ми хочемо знайти слово w так, щоб P(w|x) було найвищим. Ми використовуємо символ ˆ для позначення "нашої оцінки правильного слова".

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w|x) \tag{2.1}$$

Функція $\operatorname{argmax}_x f(x)$ означає "таке x, при якому f(x) максимальне". Рівняння (2.1) означає, що серед усіх слів у словнику ми шукаємо конкретне слово, яке максимізує праву частину P(w|x).

Інтуїція байєсівської класифікації полягає в використанні правила Байєса для трансформації рівняння (2.1) в набір інших ймовірностей. Правило Байєса представлене в рівнянні (2.2); воно дає нам засіб розкласти будь-яку умовну ймовірність P(a|b) на три інші ймовірності:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)} \tag{2.2}$$

Ми можемо підставити рівняння (2.2) в рівняння (1.1), щоб отримати рівняння (2.3):

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} \frac{P(x|w)P(w)}{P(x)} \tag{2.3}$$

Ми можемо зручно спростити рівняння (2.3), виключивши знаменник P(x). Оскільки ми обираємо потенційне виправлення слова серед усіх слів, ми будемо обчислювати P(x|w)P(w) / P(x) для кожного слова. Але P(x) не змінюється для кожного слова; ми завжди запитуємо про найбільш ймовірне слово для однієї й тієї ж помилки x, яка має однакову ймовірність P(x). Таким чином, ми можемо вибрати слово, яке максимізує цю простішу формулу:

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(x|w)\, P(w) \tag{2.4}$$

У підсумку модель шумного каналу стверджує, що у нас є певне істинне базове слово w, і у нас є шумний канал, який модифікує слово в певну можливу помилкову спостережувану поверхневу форму. Події чи модель каналу шуму, яка створює будь-яку конкретну послідовність спостережень x, моделюється за допомогою P(x|w). Ймовірність прихованого слова моделюється за допомогою P(w). Ми можемо обчислити найбільш ймовірне слово ˆw, знаючи, що ми бачили деяку помилкову форму x, множачи апріорну ймовірність P(w) і ймовірність P(x|w) та обираючи слово, для якого цей добуток є найбільшим.

Ми використовуємо підхід шумного каналу для виправлення орфографічних помилок, які не є словами, зазначеними в нашому словнику, створюючи список кандидатів, ранжуючи їх за рівнянням (2.4) і вибираючи той, який має найвищий ранг. Ми можемо модифікувати рівняння (2.4), щоб воно вказувало на цей список кандидатів слов, а не на повний словник V, наступним чином:

$$\hat{w} = \underset{w \in C}{\operatorname{argmax}} \overbrace{P(x|w)}^{\text{channel model}} \overbrace{P(w)}^{\text{prior}} \tag{2.5}$$

Алгоритм шумового каналу показано на рис.2.3.

function NOISY CHANNEL SPELLING(*word x*, *dict D*, lm, *editprob*) **returns** *correction*

if $x \notin D$
  candidates, edits ← All strings at edit distance 1 from x that are $\in D$, and their edit
  for each c, e in candidates, edits
    channel ← editprob(e)
    prior ← lm(x)
    score[c] = log *channel* + log *prior*
  return argmax$_c$ *score*[c]

Рис. 1.3 - Модель шумового каналу для виправлення помилок

Щоб детально розглянути обчислення ймовірності та апріорного (мовного моделювання), давайте розглянемо приклад, застосовуючи алгоритм до прикладу помилкового написання "acress". Перший етап алгоритму пропонує варіанти корекції, знаходячи слова, які мають схожий напис до вхідного слова. Аналіз даних про помилки в написанні показав, що більшість помилок в написанні складаються з зміни однієї літери, і тому ми часто використовуємо спрощену припущення, що ці кандидати мають відстань редагування 1 від слова з помилкою. Щоб знайти цей список кандидатів, ми використовуємо алгоритм мінімальної відстані редагування, але розширений так, щоб, крім вставок, видалень та замін, ми додамо четвертий тип редагування - транспозиції, при якій місцями міняються дві літери. Версію відстані редагування з транспозицією називають відстанню редагування Дамерау-Левенштейна. Застосування всіх таких одиночних трансформацій до "acress" дає список слів-кандидатів, представлений в таблиця 2.2.

Таблиця 2.2

Кандидати на корекцію

Помилка	Виправлення	Правильна буква	Помилкова буква	Позиція букви	Тип перетворення
acress	actress	t	-	2	видалення
acress	cress	-	a	0	вставка
acress	caress	ca	ac	0	транспонування
acress	access	ca	r	2	заміна
acress	across	o	e	3	заміна
acress	acres	-	s	5	вставка
acress	acres	-	s	4	вставка

Після того, як ми маємо набір кандидатів, для оцінки кожного з них за допомогою рівняння (2.5) потрібно розрахувати апріорну ймовірність та модель каналу.

Апріорна ймовірність кожної корекції P(w) - це ймовірність мовленнєвої моделі слова w в конкретному контексті, яку можна обчислити за допомогою будь-якої мовленнєвої моделі, від уніграм до триграм або 4-грам. Для цього прикладу давайте розпочнемо з такої таблиці, припускаючи використання уніграмної мовленнєвої моделі. Ми обчислили мовленнєву модель з 404 253 213 слів в Корпусі сучасної англійської мови (COCA).

Таблиця 2.3

Апріорна ймовірність кожної корекції

w	Count(w)	p(w)
actrees	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

Ідеальна модель ймовірності помилкового введення слова враховувала б всілякі фактори: хто був таємницею, чи був таємницею ліво- чи праворуч, та інші. Можна отримати досить розумну оцінку P(x|w), просто розглядаючи локальний контекст: саму правильну літеру, помилку при написанні та навколишні літери. Наприклад, літери m і n часто замінюють одна одну; це частково факт їхньої ідентичності (ці дві літери вимовляються схоже та розташовані поруч на клавіатурі), а також факт контексту (вони вимовляються схоже і зустрічаються в схожих контекстах).

Проста модель може оцінювати, наприклад, ймовірність p(acress|across), використовуючи лише кількість разів, коли літеру "e" було підставлено замість

літери "o" у великому корпусі помилок. Для обчислення ймовірності для кожної правки таким чином нам знадобиться матриця плутанини, що містить підрахунки помилок. Загалом матриця плутанини перераховує кількість разів, коли одна річ була сплутана з іншою. Таким чином, наприклад, матриця заміщення буде квадратною матрицею розміром 26x26 (або загалом |A| × |A|, для алфавіту A), що представляє кількість разів, коли одну літеру невірно використовували замість іншої. Згідно з Kernighan et al. (1990), ми будемо використовувати чотири матриці плутання.

del[x, y]: count(xy typed as x)

ins[x, y]: count(x typed as xy)

sub[x, y]: count(x typed as y)

trans[x, y]: count(xy typed as yx)

Ймовірності вставки та видалення зумовили попередньою літерою; замість цього можна було вибрати зумовлення наступною літерою. Один із способів отримання матриць плутання - витягувати їх із списків помилок написання, подібних до таких:

- для "additional" існують варіанти, такі як "addional" і "additonal", що свідчить про заміщення;
- для "environments" спостерігаються варіації, такі як "enviornments", "enviorments" і "enviroments", що свідчить про заміщення та можливо вставки;
- для "preceded" помічено помилку в написанні "preceeded", що свідчить про заміщення.

Альтернативний підхід, використаний Кернігеном та ін. (1990), полягає в обчисленні матриць ітеративно за допомогою самого алгоритму корекції помилок у написанні. Ітеративний алгоритм спочатку ініціалізує матриці рівними значеннями; отже, будь-яка літера має однакову ймовірність бути вилученою, однакову ймовірність бути заміненою будь-якою іншою літерою і т. д. Далі алгоритм корекції помилок у написанні виконується на наборі помилок у

написанні. З урахуванням набору помилок, зіставлених із їх передбаченими виправленнями, матриці плутання можуть бути перераховані, алгоритм корекції помилок у написанні виконується знову, і так далі.

Коли є матриці плутання, можна оцінити P(x|w) за допомогою наступного виразу (де wi - це і-тий символ правильного слова w і xi - це і-тий символ помилкового написання x :

$$P(x|w) = \begin{cases} \frac{\text{del}[x_{i-1}, w_i]}{\text{count}[x_{i-1} w_i]}, \text{if deletion} \\ \frac{\text{ins}[x_{i-1}, w_i]}{\text{count}[w_{i-1}]}, \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, \text{if transposition} \end{cases} \quad (2.6)$$

Використання підрахунків від Кернігена та ін. (1990) призводить до ймовірностей моделі помилок для слова "acress", які показані в таблиці 2.4.

Таблиця 2.4

Модель каналу для слова "acress"

Кандидат виправлення	Правильна буква	Помилкова буква	x|w	P(x|w)
actress	t	-	c|ct	.000117
cress	-	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	ca	r	r|c	.000000.209
across	o	e	e|o	.0000093
acres	-	s	es|e	.0000321
acres	-	s	ss|s	.0000342

В таблиці 2.5 вказані кінцеві ймовірності для кожного потенційного варіанту виправлення; уніграмна апріорна ймовірність помножується на ймовірність (обчислена за допомогою рівняння (6) та матриць плутання). Кінцевий стовпець показує добуток, помножений на 10^9 для зручності читання.

Таблиця 2.5

Обчислення рейтингу для кожного варіанту виправлення

Кандидат виправлення	Правильна буква	Помилкова буква	x|w	P(x|w)	P(w)	10^9*P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2,7
cress	-	a	a|#	.00000144	.00000054 4	0,00078
caress	ca	ac	ac|ca	.00000164	.00000170	0,0028
access	ca	r	r|c	.000000.209	.0000916	0,019
across	o	e	e|o	.0000093	.000299	2,8
acres	-	s	es|e	.0000321	.0000318	1
acres	-	s	ss|s	.0000342	.0000319	1

Розрахунки, представлені в таблиці 2.5, показують, що реалізація моделі шумового каналу вибирає "across" як найкращий варіант виправлення, а "actress" - як другий найбільш ймовірне слово. Нажаль, алгоритм допустив помилку у цьому випадку. Навколишні слова вказують на те, що "actress", а не "across", було призначено словом.

З цього приводу важливо використовувати більші мовні моделі, ніж уніграми. Наприклад, якщо використовуємо Корпус сучасної американської англійської мови для обчислення ймовірностей біграм для слів "actress" і "across"

у їхньому контексті, використовуючи вирівнювання add-one, отримуємо такі ймовірності:

P(actress|versatile) = .000021

P(across|versatile) = .000021

P(whose|actress) = .0010

P(whose|across) = .000006

Помноживши їх, ми отримаємо оцінку мовної моделі для двох кандидатів в контексті:

P(“versatile actress whose”) = .000021 ∗ .0010 = 210×10−10

P(“versatile across whose”) = .000021 ∗ .000006 = 1×10−10

Об'єднуючи мовну модель з моделлю помилок в таблиці 2.3, біграмна шумова модель каналу тепер вибирає правильне слово "actress".

Оцінка алгоритмів виправлення помилок у написанні, зазвичай, здійснюється за допомогою відокремлення наборів даних із списків помилок.

2.6 Обробка стоп слів

Пошук та видалення стоп-слів — це стандартний крок в обробці тексту, особливо при підготовці до задач машинного навчання та обробки природної мови. Стоп-слова — це зазвичай загальновживані слова в мові, які самі по собі не несуть значної інформації для аналізу тексту, наприклад, "і", "але", "в", "на" тощо.

Перш ніж почати обробку тексту, необхідно мати готовий список стоп-слів. Цей список може бути стандартним, наданим популярними бібліотеками NLP, такими як NLTK для англійської мови, або може бути спеціалізованим для конкретної задачі.

Після токенізації кожен токен перевіряється на наявність у списку стоп-слів. Якщо токен виявляється стоп-словом, він видаляється з масиву токенів.

Іноді стоп-слова можуть мати значення в певному контексті. Наприклад, в пошукових запитах або при роботі з фразами, де важливий порядок слів. В таких випадках може бути необхідно адаптувати список стоп-слів або процес видалення, щоб не втратити важливу інформацію.

Після видалення стоп-слів токени можуть бути знову об'єднані для створення оновленого тексту, або вони можуть бути залишені як список токенів для подальшої обробки. Приклад видалення стоп слів в тексті наведено в таблиці 2.6.

Таблиця 2.6

Результат процесу видалення стоп слів

Текст зі стоп-словами	Без стоп-слів
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks, Computer Science, Portal, Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

Загально вважається, що стоп-слова - це "один і той же набір слів". Проте це дійсно може мати різний зміст для різних задач. Наприклад, у деяких програмах може бути відповідним списком стоп-слів вилучення всіх стоп-слів, від визначників (наприклад, the, a, an) до прийменників (наприклад, above, across, before) і деяких прикметників (наприклад, good, nice). У деяких випадках це корисно. Проте для деяких ззадачах це може бути шкідливо. Наприклад, у випадку аналізу настрою вилучення прикметників, таких як "гарний" і "приємний", а також відмов, таких як "не", може спотворити роботу алгоритмів. У таких випадках можна вибрати мінімальний список стоп-слів, що складається лише з визначників, або визначників з прийменниками, або лише сполучників залежно від потреб застосування. Ось приклади мінімальних списків стоп-слів, які можна використовувати:

- Визначники – Визначники зазвичай вказують на іменники, де за визначником зазвичай слідує іменник. Приклади: the (у), a (один), an (один), another (інший)
- Сполучники-координатори – Сполучники-координатори з'єднують слова, фрази та речення. Приклади: for (для), an (й), nor (ні), but (але), or (або), yet (ще), so (так)
- Прийменники – Прийменники виражають відносини у часі або просторі. Приклади: in (у), under (під), towards (на кілька), before (перед)

2.7 Опрацювання слів не природної мови

Блок "Опрацювання токенів, що не є словами природної мови" у контексті обробки тексту відноситься до фільтрації та корекції тих елементів тексту, які не відповідають звичайним словам в мові. Це можуть бути числа, знаки пунктуації, спеціальні символи, емодзі, незнайомі або невизначені терміни, та інші елементи, які не вносять значущого вкладу у семантичний або тематичний аналіз тексту. Ось етапи процесу:

Визначення нерелевантних токенів: На цьому етапі система ідентифікує токени, які не відносяться до словесного запасу природної мови. Для цього можуть використовуватися правила, регулярні вирази або списки невідповідних токенів.

Фільтрація токенів: Виявлені токени фільтруються з тексту. Фільтрація може включати видалення токенів або їх заміну. Наприклад, числа можуть бути вилучені або замінені на загальний токен "<число>".

Нормалізація токенів: Деякі нерелевантні токени можуть бути нормалізовані, якщо це має сенс для задачі. Наприклад, різні форми валюти можуть бути приведені до одного стандартного токена.

Корекція помилок: У деяких випадках, таких як текст, введений вручну, можуть бути помилки введення або орфографії, які потрібно виправити, щоб текст міг бути коректно оброблений.

Обробка особливих символів: Символи, які не відносяться до стандартного алфавіту мови (наприклад, математичні символи або емодзі), можуть бути оброблені спеціальним чином або видалені.

Збереження структурно важливих символів: У деяких випадках символи, які технічно не є словами, можуть бути важливими для збереження структури тексту (наприклад, символи нового рядка в коді або розмітці).

Оновлення індексів токенів: Після видалення або заміни токенів, індексація токенів у тексті може бути оновлена для збереження вірної послідовності.

Підготовка до подальшої обробки: Очищені та опрацьовані токени тепер готові до наступних етапів обробки.

Для виявлення слів неприродної мови можна скористатись такими методами:

- Регулярні вирази: Можуть бути сконфігуровані для виявлення послідовностей символів, які не відповідають звичайним словам, таких як електронні адреси, URL, номери телефонів, ідентифікатори тощо.
- Словники спеціальної термінології: Використання предметно-орієнтованих словників або баз даних для виявлення та обробки спеціалізованої лексики, що не є частиною загального вжитку.
- Машинне навчання -Алгоритми класифікації можуть бути навчені виявляти слів, що не входять до природної мови, на основі навчальних даних, що містять приклади обох категорій.
- Ідентифікація за контекстом: Застосування методів обробки природної мови, які розглядають контекст слова, щоб визначити, чи є воно частиною стандартної мови.

- Хешування та підрахунок частот: Токени, які з'являються рідко та не вписуються в розподіл частот відомих слів, можуть бути підозрілими як не належні до природної мови.
- Фільтрація на основі довжини: Встановлення обмежень на довжину слова може допомогти ідентифікувати аномально короткі або довгі токени, які можуть бути несловесними.
- Морфологічний аналіз: Аналіз словоформ може виявити токени, які не відповідають морфологічним правилам мови.

2.8 Скорочення, абревіатури, вульгаризми і жаргонізми

Обробка скорочень, абревіатур, вульгаризмів і жаргонізмів вимагає специфічних підходів, оскільки кожен з цих елементів має свої особливості.

Скорочення і абревіатури — це стислі форми слів або фраз, які використовуються для економії простору або часу. Методи обробки:

- Використання словників абревіатур: Створення або використання існуючих списків, де кожній абревіатурі відповідає її повна форма.
- Правила для розшифровки: Розробка набору правил для перетворення скорочень в повні слова на основі контексту (наприклад, "Dr." перетворюється на "Doctor").
- Алгоритми машинного навчання: Навчання моделей розпізнавати та розшифровувати абревіатури в різних контекстах.

Вульгаризми — це слова або вислови, які вважаються грубими або нецензурними. Методи обробки:

- Фільтрація за списком: Видалення або заміна вульгаризмів, використовуючи списки нецензурних слів.
- Контекстуальне виявлення: Розпізнавання вульгаризмів на основі контексту використання для уникнення помилкової ідентифікації.

Жаргонізми — це слова або вислови, які використовуються певною групою людей і можуть бути незрозумілими для інших. Методи обробки:

- Словники жаргону: Використання спеціальних словників для визначення та розшифровки жаргонних слів.
- Контекстний аналіз: Аналіз контексту, в якому вживається жаргон, для визначення його значення або адекватної заміни.
- Машинне навчання: Використання класифікаційних моделей для виявлення та обробки жаргону в тексті.

2.9 Векторизація тексту

У галузі інформаційного пошуку найпопулярнішою числовою та описовою статистичною технікою є TF-IDF, яка широко використовується як фактор ваги. Механізм ваги TF-IDF описує, наскільки важливими є слова на основі їх присутності у вмісті документів. В результаті TF-IDF може бути використаний для різних завдань, таких як витягування словесних токенів зі статей, розрахунок ступенів схожості між документами, визначення значущого ранжування та іншого. У запропонованому методі три статистичні міри, а саме TF, IDF та TF-IDF, були розраховані для кожного словесного токена у списках як у кластерах, так і в документах.

Частота термінів (TF) - це метрика, яка вимірює, наскільки часто певні слова з'являються у вмісті документа. TF може бути визначений, як у формулі (2.7). Високе значення TF вказує на більш важливе слово у документі.

$$TF_{t,d} = \frac{f_{t,d}}{\max\{f_{t',d}: t' \in d\}} \tag{2.7}$$

де ft,d позначає повторювану появу слова/терміна t у документі d. З іншого боку, інверсна частота документів (IDF) розраховує рідкісність та значимість

слова/терміна у всіх документах. IDF визначається як у формулі (8). Високе значення IDF вказує на рідкісне слово у всіх документах.

$$IDF_{t,D} = log \frac{D}{\{d \in D: t \in d\}} \quad (2.8)$$

де $ID \quad t,D$ - це логарифмічна шкала, яка ділить загальну кількість документів D на кількість документів, які включають слово/термін t.

Формула (2.9) визначає вагу TF-IDF. Коли частота певного слова/терміна у документі є високою, а кількість документів, які включають це слово/термін, є низькою, значення ваги TF-IDF зростає.

$$TF - IDF = TF_{t,d} \times IDF_{t,D} \quad (2.9)$$

Для демонстрації векторизації тексту за допомогою методу TF-IDF, використаємо два коротких речення:

– "Кіт спить на підвіконні."

– "Собака спить у будці."

Після обчислень TF, IDF та TF-IDF утворюється таблиця 2.7

Таблиця 2.7

Результат векторизації

	будці	кіт	на	підвіконні	собака	спить
Речення 1	0.000	0.534	0.534	0.534	0.000	0.380
Речення 2	0.632	0.000	0.000	0.000	0.632	0.449

У цій таблиці, кожен рядок відповідає документу, а кожен стовпець – окремому слову у векторизованому вигляді. Значення в кожній комірці представляють TF-IDF вагу кожного слова у відповідному документі. Наприклад, слово "кіт" має високу TF-IDF вагу у першому документі, оскільки воно

з'являється тільки там, але не у другому. Це показує, що "кіт" є важливим терміном для першого документа у порівнянні з іншим.

Дані, отримані з векторизації тексту за допомогою методу TF-IDF, мають широкий спектр застосувань у галузі обробки природної мови (NLP) та аналізу даних. Ось декілька прикладів, де вони можуть бути корисні:

- Машинне навчання та класифікація текстів: Векторизовані дані TF-IDF можна використовувати як вхідні дані для моделей машинного навчання для класифікації текстів, наприклад, для визначення теми або сентименту тексту.
- Інформаційний пошук: TF-IDF може допомогти у визначенні релевантності документів у пошукових системах або базах даних на основі запитів користувачів.
- Системи рекомендацій: У системах рекомендацій, таких як рекомендації статей або новин, TF-IDF може допомогти у виборі найбільш релевантного контенту для користувача.
- Кластеризація та аналіз тематик: Вектори TF-IDF можуть бути використані для групування або кластеризації документів на основі їхнього змісту, що дозволяє аналізувати тематики та тренди у великих текстових наборах даних.
- Виявлення плагіату: Векторизація TF-IDF може використовуватися для порівняння документів і виявлення подібності між ними, що є корисним для виявлення плагіату.
- Автоматичне анотування та підсумовування текстів: TF-IDF може допомогти у виокремленні ключових слів та фраз у текстах, сприяючи створенню резюме або анотацій.

Розмір вектору тексту, отриманого через процес векторизації, залежить від кількості унікальних слів у всьому корпусі текстів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ МЕТОДИКИ

3.1 Інструменти обробки природної мови

Для роботи з обробкою природної мови (NLP) найбільш використовуваною та підходящою мовою програмування є Python.

Існує безліч інструментів та бібліотек для обробки природної мови (NLP), які використовуються у програмуванні та аналізі даних. Далі наведено опис найпопулярніших інструментів обробки природної мови.

NLTK (Natural Language Toolkit): Це одна з найбільш відомих бібліотек для NLP у Python. NLTK включає модулі для обробки тексту, класифікації, токенізації, стемінгу, тегування, парсингу та семантичного аналізу.

spaCy: Ця бібліотека зосереджена на швидкості та ефективності. SpaCy надає передові можливості для тегування частин мови, розбору залежностей та розпізнавання іменованих сутностей.

Gensim: Gensim використовується для моделювання тем і документної схожості. Вона ефективна для роботи з великими текстовими наборами даних і забезпечує функціонал для моделей, таких як Word2Vec, FastText та Latent Dirichlet Allocation (LDA).

Scikit-learn: Хоча Scikit-learn переважно відомий як бібліотека для машинного навчання, він також має модулі для векторизації тексту, TF-IDF та інших методів, які використовуються у NLP.

BERT і трансформери (Transformers): BERT (Bidirectional Encoder Representations from Transformers) та інші моделі на основі трансформерів (наприклад, GPT, RoBERTa) забезпечують дуже високу точність у багатьох задачах NLP, включаючи розуміння тексту, класифікацію, переклад та генерацію тексту.

Apache OpenNLP: Це машинно-орієнтована бібліотека для обробки мови, яка підтримує різні NLP завдання, такі як токенізація, тегування частин мови, парсинг, розпізнавання іменованих сутностей та розділення речень.

Stanford NLP: Розроблено Стенфордським університетом, цей набір інструментів підтримує різні мови та надає можливості для тегування частин мови, розбору залежностей та розпізнавання іменованих сутностей.

AllenNLP: Це високорівневий фреймворк для досліджень у галузі NLP, заснований на PyTorch. Він підтримує різні сучасні моделі та методи обробки мови.

В якості основних інструментів було обрано бібліотеки NLTK для виконання основних дій алгоритму таких, як токенізація виправлення орфографічних помилок та видалення стоп слів.

3.2 Блок токенізації

Бібліотека NLTK містить в собі метод токенізації який використовується в реалізації алгоритму.

Перш за все, потрібно імпортувати бібліотеку NLTK та завантажити дані для токенізації. Для цього використовуємо функцію nltk.download('punkt'), щоб завантажити ресурси, необхідні для роботи з NLTK.

Після завантаження ресурсів необхідно використовувати функцію word_tokenize, яка приймає текстовий рядок і повертає список токенів (слів або символів) з цього рядка.

На рисунку 3.1 представлений приклад роботи токенізатора. Токенізатор прийняв на вхід речення і обробивши його в якості вихідних даних видав список розділених слів, знаків і символів як окремі частини.

В процесі токенізації можна одразу видалити символи і звести текст до нижнього регістру (рис 3.2).

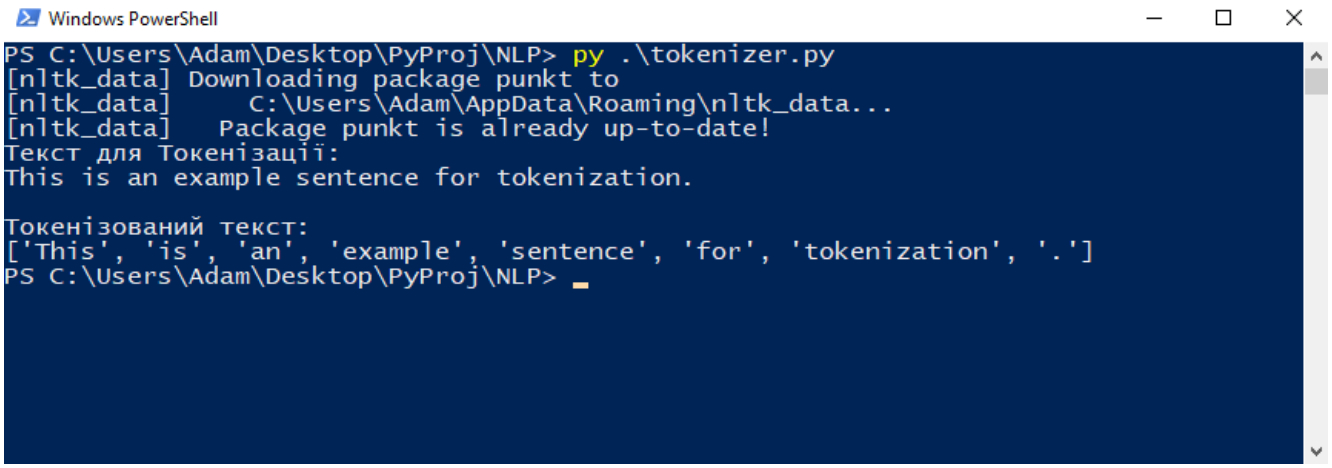


Рис. 2.1 – Приклад роботи блоку токенізації

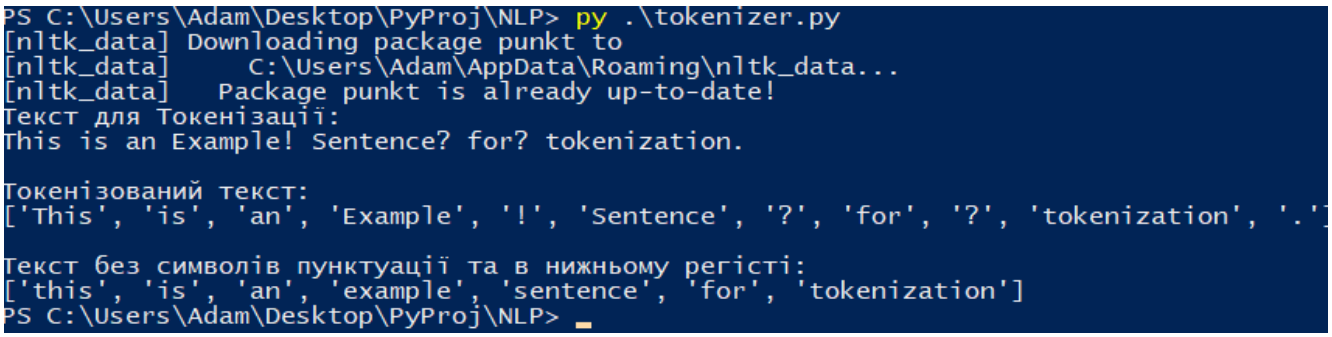


Рис. 3.2 – Приклад роботи токенізатору видалення пунктуації

На рис3.2 зображено вікно терміналу, де запущено Python скрипт для обробки тексту. Скрипт виконує такі дії:

– Спочатку в терміналі відображається спроба завантаження пакета punkt для nltk, який необхідний для токенізації тексту. Повідомлення вказує, що пакет punkt вже встановлено та оновлено до останньої версії.
– Далі йде текст для токенізації: "This is an Example! Sentence? for? tokenization."
– Перший варіант виводу демонструє токени, в якому пунктуація відділена як окремі токени (['This', 'is', 'an', 'Example', '!', 'Sentence', '?', 'for', '?', 'tokenization', '.']).

– Другий варіант показує текст без пунктуаційних знаків і з усіма словами в нижньому регістрі (['this', 'is', 'an', 'example', 'sentence', 'for', 'tokenization']).

Ці два підходи до токенізації використовуються для різних цілей. Токенізація з пунктуацією може бути корисною для завдань, які потребують збереження структури речення, наприклад, для синтаксичного аналізу. Токенізація без пунктуації та в нижньому регістрі часто використовується для спрощення тексту перед подальшим аналізом, наприклад, для машинного навчання або статистичного аналізу тексту.

3.3 Виправлення орфографічних помилок в тексті

Для виправлення орфографічних помилок використовується бібліотека pyspellchecker. Ця бібліотека Python, яка дозволяє виявляти та виправляти орфографічні помилки в текстах. Вона працює для декількох мов та базується на словнику зі словами для кожної підтримуваної мови.

Python алгоритм перевірки орфографії на основі публікації Пітера Норвіга в блозі про створення простого алгоритму перевірки правопису.

Він використовує алгоритм відстані Левенштейна для знаходження перестановок у межах відстані редагування 2 від оригінального слова. Потім він порівнює всі перестановки (вставки, видалення, заміни та транспозиції) із відомими словами у списку частотності слів. Ті слова, які зустрічаються частіше у списку частотності, імовірніше є правильними результатами.

Pyspellchecker підтримує кілька мов, включаючи англійську, іспанську, німецьку, французьку, португальську, арабську та баскську.

Pyspellchecker дозволяє встановити відстань Левенштейна (до двох) для перевірки. Для довгих слів настійно рекомендується використовувати відстань 1, а не за замовчуванням 2.

Створена функція correct_misspelled виглядає виправляэ орфографічні помилок в списку слів. Вона використовує pyspellchecker для виявлення та виправлення помилкових слів.

Спочатку створюється об'єкт SpellChecker. Далі функція spell.unknown приймає ітерований об'єкт слів і повертає множину тих, що не визнані словником. Для кожного невідомого слова знаходиться найбільш імовірне виправлення за допомогою методу spell.correction. Усі входження слова з помилкою замінюються на виправлене слово в списку (рис.3.3).

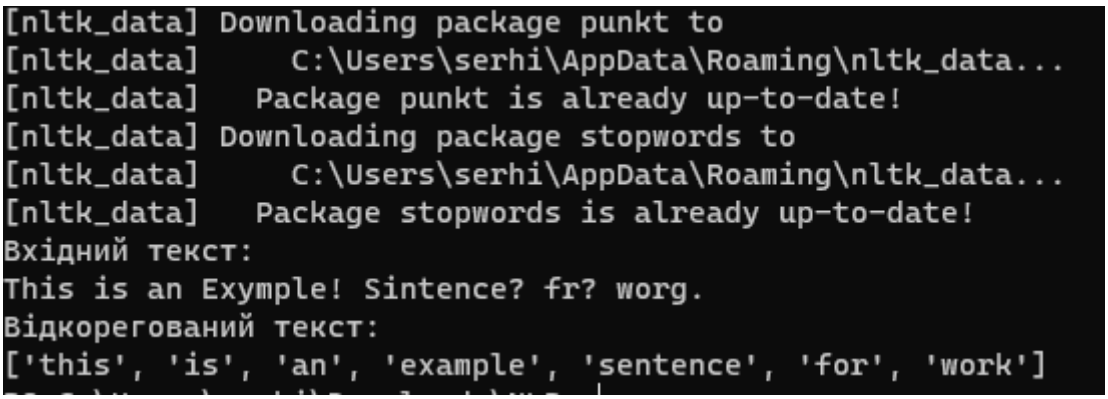


Рис. 3.3 – Приклад роботи блоку виправлення помилок

3.4 Обробка стоп слів

Видалення стоп-слів з тексту - це поширена задача в області обробки природної мови (NLP). Стоп-слова - це зазвичай найбільш часто використовувані слова в мові, такі як "і", "але", "якщо", які часто видаляються з тексту перед його подальшим аналізом, оскільки вони зазвичай не несуть важливої семантичної інформації.

Один з підходів до видалення стоп-слів у Python - використання бібліотеки Natural Language Toolkit (NLTK), яка містить вбудований список стоп-слів для багатьох мов

Для видалення стоп-слів спершу треба перетворити текст в нижній регістр і розбити його на токени, що було виконано в попередніх блоках.

Іноді вам може знадобитися додавати або видаляти слова зі свого списку стоп-слів. Наприклад при класифікації статі про їжу необхідно акцентувати увагу на те які страви мають більшу вагу у статі. Можна очікувати, що слово "їжа" (або подібні слова) будуть часто згадуватися. Вони не надають цінної інформації. Отже, їжа - це стоп-слово, його можна додати до спеціалізовано списку стоп-слів завдяки бібліотеці NLTK. Але це не заміює стандартний список стоп-слів, що спрощує роботу. На рисунку 3.4 зображено приклад роботи блоку видалення стоп слів.

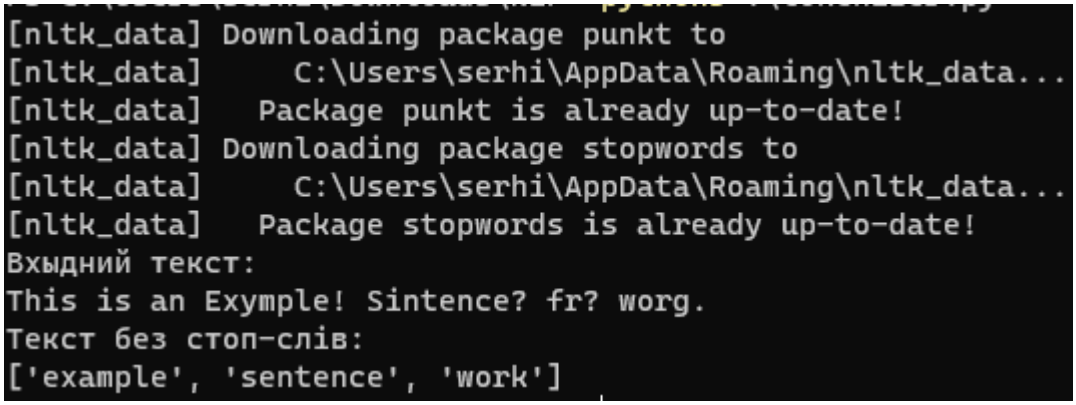


Рис. 3.4 – Приклад роботи видалення стоп слів

3.5 Стемінг тексту

Як було описано в попередньому розділі, стемінг - це метод обробки тексту, який вилучає префікси та суфікси зі слів, перетворюючи їх на їх фундаментальну або кореневу форму. Основна мета стемінгу полягає в тому, щоб спростити та стандартизувати слова, підвищуючи ефективність завдань обробки природної мови. Далі детальніше описано про стемінг та стемери в Python.

NLTK (Natural Language Toolkit) включає в себе різні алгоритми стемінгу, включаючи кілька типів. Розглянемо два найпоширеніших.

Портер стеммер - це один з перших і найбільш широко використовуваних алгоритмів стемінгу. Він був розроблений Мартіном Портером у 1980 році і спрямований на видалення найбільш звичайних морфологічних і інфлективних

закінчень в англійській мові. Портер стеммер працює за серією правил, які вирізають закінчення слова (рис.3.5).

Snowball стеммер - це поліпшена версія стеммера, розроблена також Мартіном Портером. Вона названа "Snowball", оскільки це мова програмування, створена для написання стеммерів, яка легка у вивченні та використанні. Snowball стеммер включає набір алгоритмів для різних мов, і для англійської мови він вважається більш складним і точним у порівнянні з оригінальним Портер стеммером.

Обидва стеммери доступні в бібліотеці Natural Language Toolkit (NLTK) для Python, яка є популярним інструментом для роботи з текстовими даними в області NLP. Вони використовуються для попередньої обробки тексту, зокрема для зменшення розміру словників і підвищення швидкості обчислень у задачах, таких як класифікація текстів, пошук інформації, та інших.

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\serhi\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\serhi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Вхідний текст:
George is at the pet store, looking at what kind of pet he might want to get for his birthday. George asked if he could
have a horse, but his parents said no because horses are too big.
Відкорегований текст:
['george', 'is', 'at', 'the', 'pet', 'store', 'looking', 'at', 'what', 'kind', 'of', 'pet', 'he', 'might', 'want', 'to',
 'get', 'for', 'his', 'birthday', 'george', 'asked', 'if', 'he', 'could', 'have', 'a', 'horse', 'but', 'his', 'parents',
 'said', 'no', 'because', 'horses', 'are', 'too', 'big']
Текст після обробки
['georg', 'pet', 'store', 'look', 'kind', 'pet', 'might', 'want', 'get', 'birthday', 'georg', 'ask', 'could', 'hors', 'p
arent', 'said', 'hors', 'big']
```

Рис. 3.5 – Приклад роботи блоку стемінгу на основі методу PorterStemmer

3.6 Блоки обробки скорочень і абревіатур, вульгаризмів і жаргонізмів.

Блоки обробки скорочень і абревіатур, вульгаризмів і жаргонізмів хоч і виконують різну за призначенням функцію, але мають схожу структуру. Всі блоки використовують списки. Після перевірки токенів відбувається заміна відповідно до списка. Такі списки можна знайти у відкритому доступі або самостійно доповнити їх в залежності від специфіки тексту.

Блок обробки скорочень і абревіатур заміняє у вхідному тексті скорочення і абревіатури на їх повне представлення для збагачення тексту цінною інформацією. Слід зауважити, що після розкриття абревіатур один токен розділитись на декілька, це збільшить векторне представлення тексту. Розкриті абревіатури залишаються у вигляді одного токену (рис.3.6).

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\serhi\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\serhi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Вхідний текст:
Dr.Sorbie and Ms.Helen live in the USA
Розкриття скорочень і абревіатур
['doctor', 'sorbie', 'and', 'miss', 'helen', 'live', 'in', 'the', 'United States of America']
PS C:\Users\serhi\Downloads\NLP>
```

Рис. 3.6 – Приклад роботи блоку заміни абревіатур і скорочень

Для блоків обробки скорочень і абревіатур, вульгаризмів і жаргонізмів можна використовувати один словник. Якщо для аналізу необхідно класифікувати першочерговий вигляд таких слів то цей блок за необхідністю прибирається з загального алгоритму.

3.7 Блок векторизації тексту

Щоб виконати векторизацію текста методом TF-IDF у Python можна використовувати бібліотеку scikit-learn.

TF-IDF, що означає "Term Frequency-Inverse Document Frequency", є статистичною мірою, яка використовується для оцінки важливості слова у контексті документа, який є частиною корпусу. Цей показник використовується в багатьох застосуваннях обробки природної мови та інформаційного пошуку.

Для того щоб обрахувати TF-IDF за допомогою scikit-learn треба виконати такі кроки:

– Створити екземпляр TfidfVectorizer: Це здійснюється через виклик TfidfVectorizer
– Виклик методу fit_transform: Цей метод приймає список рядків (документів) і повертає матрицю TF-IDF, де кожен рядок відповідає документу, а кожен стовпчик відповідає терміну з усього корпусу.

TfidfVectorizer автоматично виконує необхідні підрахунки TF та IDF і комбінує їх для створення векторів TF-IDF. Вектори TF-IDF використовуються в багатьох застосуваннях, включаючи машинне навчання для текстових даних, аналіз тексту та інші. Також блок векторизації рахує векторне представлення тексту (рис.3.7).

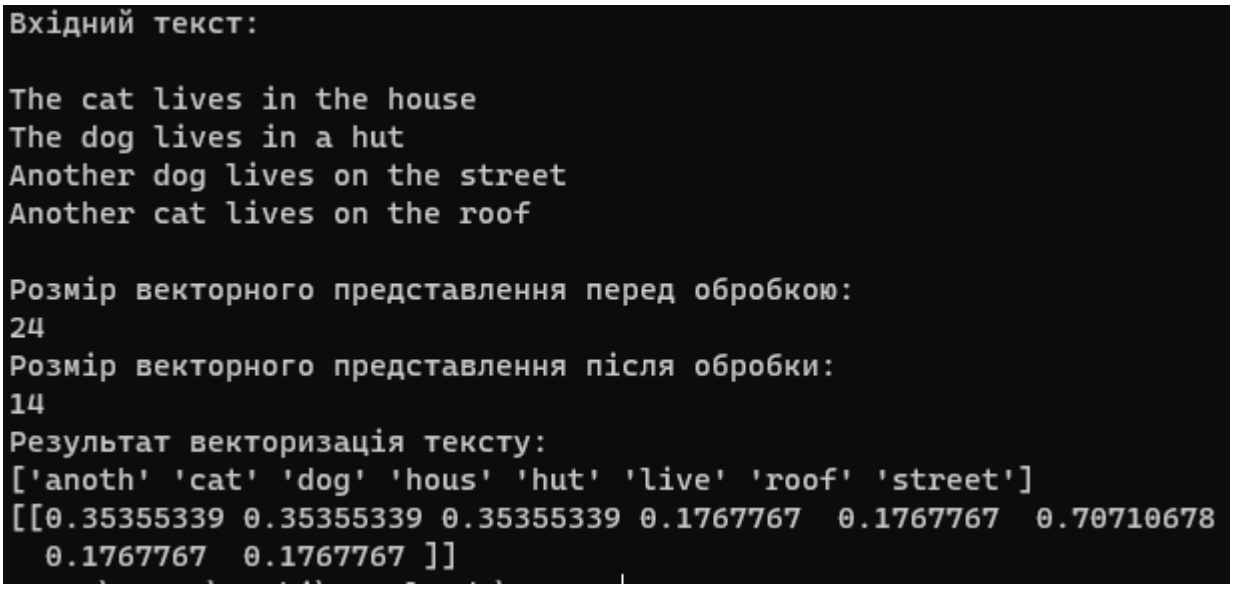


Рис. 3.7 - Приклад роботи блоку векторизації

3.8 Результати моделювання

Для моделювання результатів роботи алгоритму було обрано декілька текстів. Алгоритм отримуючи необроблений текст, як вхідні дані, крім його обробки записує інформацію виконаної роботи. А саме:

– Кількість видалених знаків пунктуації;
– Кількість видалених символів не природної мови;

– Кількість розкритих абревіатур;
– Кількість замімених вульгаризмів;
– Кількість замімених жаргонізмів;
– Кількість орфографічних виправлень;
– Кількість видалених стоп слів;
– Розмір векторного представлення перед обробкою;
– Розмір векторного представлення після обробки;

Ці данні входять в таблицю результатів (таблиця 3.1). Вхідний, оброблений та векторізований текст в таблицю не входить. Приклад виводу результатів обробки зображено на рисунку 3.8.

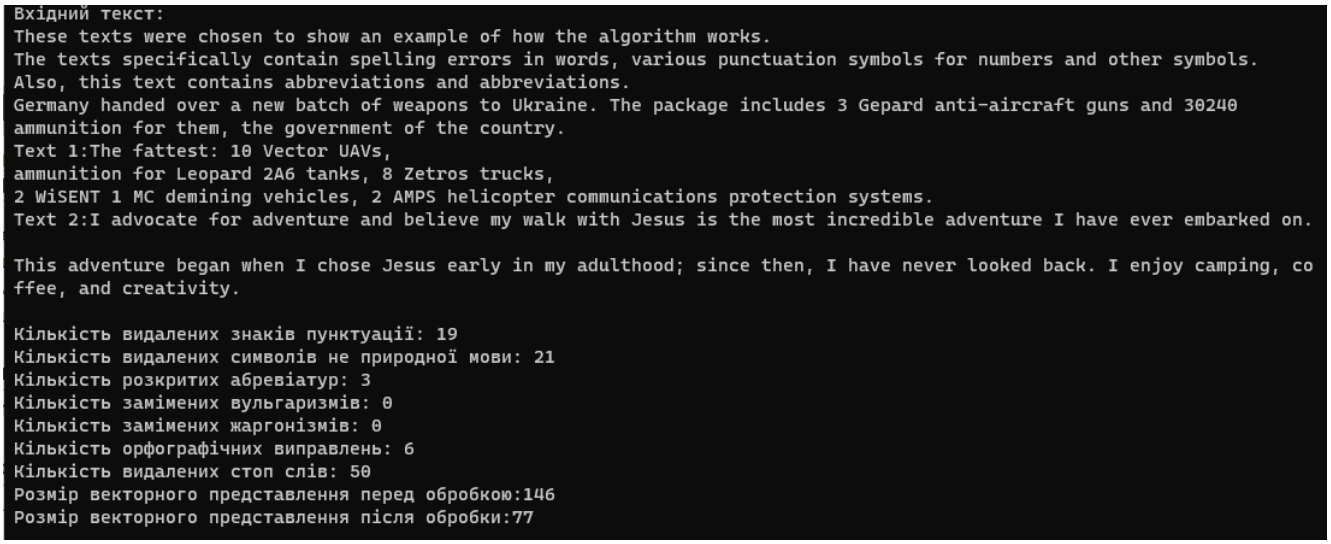


Рис. 3.8 – Приклад виводу результатів обробки тексту

Результати роботи алгоритму можуть бути використані для подальшого використання. Алгоритм перетворює неструктуровані текстові дані в структуровані за рахунок векторизації що дає більше можливостей для їх аналізу і і використання.

Кінцевим використанням таких даних може бути використання для прийняття рішень, покращення та роботи віртуальних помічників, векторизовані

дані можна використовувати для навчання нейромереж, класифікація та вибір категорії текстів великих текстів низької якості.

Таблиця 3.1

Результати моделювання

Кількість токенів вхідного тексту	Кільк. стоп-слів	Кількість токенів з помилками	Кількість токенів, що не є словами природної мови	Кількість абревіатур і скорочень	Кількість вульгарних слів	Кількість жаргонних слів	Розмір вектору після обробки
5	2	0	0	0	0	0	3
15	3	0	0	0	2	0	9
20	5	2	2	0	0	2	11
20	4	0	3	0	0	0	10
25	6	2	1	0	0	0	13
34	18	0	1	0	0	0	14
77	37	2	9	0	0	0	26
152	44	7	35	0	0	0	75
146	50	6	21	3	0	0	77
104	51	2	5	0	0	0	47

В середньому вектор після обробки на 49% менше в порівнянні до кількості вхідних токенів в тексті. Якщо розділити тексти за довжиною то різниця для коротких текстів – 46%, для середніх текстів – 55%, для довгих - 51%.

ВИСНОВКИ

В ході написання дипломної роботи були виконані та досягнуті поставлені цілі та задачі.

1. Проведено аналіз видів тексту за рівнем їх структурованості, визначено проблематику обробки неструктурованих текстів. Проаналізовано методи та технології попередньої обробки текстових даних природньою мовою, та інші методи, що можуть бути використані для підвищення якості попередньої обробки тексту.
2. Проаналізовано види та особливості шумів, які можуть зустрічатись в текстах низької якості. Детально розглянуто найбільш поширені шуми, серед яких: синтаксичний шум, семантичний шум, лексичний шум, орфографічний шум, шум в інформаційних даних.
3. Розроблена загальна схема обробки тексту низької якості, яка включає наступні блоки: токенізація, виправлення орфографічних помилок, видалення стоп-слів, обробка абревіатур, скорочень, вульгаризмів, жаргонізмів і блок векторизації та аналізу результату.
4. Практична реалізація методики виконана мовою програмування Python з використанням бібліотеки обробки текстових даних NLTK, бібліотеки Pyspellchecker для виправлення орфографічних помилок в тексті, бібліотеки scikit-learn та обрахування векторного представлення тексту.
5. Проведено моделювання, яке показало що в середньому розмір вектору, яким представляється текст після попередньої обробки, становить близько 49% по відношенню до вхідної кількістю токенів в тексті, при цьому: для коротких текстів – 46%, для середніх текстів – 55%, для довгих - 51%.

ПЕРЕЛІК ПОСИЛАНЬ

1. Корекція правопису та шумовий канал. [Електронний ресурс] // Деніел Журафскі, Джеймс Мартін Stanford - 2017 – Режим доступу до ресурсу: https://web.stanford.edu/~jurafsky/slp3/B.pdf
2. «Обробка природної мови в Python» // Стівен Берд, Еван Кляйн і Едвард Лопер - 2009
3. «Обробка мови та мови: вступ до обробки природної мови, комп’ютерної лінгвістики та розпізнавання мови» // Деніел Джурафскі та Джеймс Х. Мартін – 2021
4. "Основи статистичної обробки природної мови" // Крістофер Д. Меннінг і Хінріх Шютце - 1999
5. «Текстовий аналіз: концепції, реалізація та проблема великих даних» // Ігнаціус Езеані та Чібуезе Огбонна - 2017
6. «Вступ до інформаційного пошуку» // Крістофер Д. Меннінг, Прабхакар Рагхаван і Хінріх Шютце - 2008
7. "Технології попередньої обробки даних для інтелектуального аналізу даних" // Доріан Пайл - 1999
8. «Наука про дані для бізнесу» // Фостер Провост і Том Фосетт - 2013
9. "Видобуток текстових даних" // Charu C. Aggarwal - 2012
10. «Текстова аналітика за допомогою Python: практичний реальний підхід до отримання корисної інформації з ваших даних» // Діпанджан Саркар - 2019
11. "Машинне навчання: імовірнісна перспектива" // Кевін П. Мерфі - 2012
12. "Розпізнавання образів і машинне навчання" // Крістофер М. Бішоп – 2006
13. «Видобуток і аналіз тексту: практичні методи, приклади та тематичні дослідження з використанням SAS» // Гутам Чакраборті, Муралі Паголу, Сатіш Гарла - 2014

14. «Глибоке навчання для обробки природної мови» // Йоав Голдберг - 2017
15. "Управління та аналіз текстових даних: практичний вступ до пошуку інформації та аналізу тексту" // ChengXiang Zhai та Sean Massung - 2016
16. «Обробка природної мови в дії» // Lane, Howard, and Hapke - 2019
17. «Прикладний аналіз тексту за допомогою Python: увімкнення продуктів з даними з урахуванням мови за допомогою машинного навчання» // Бенджамін Бенгфорт, Тоні Охеда та Ребекка Білбро – 2018
18. "Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data" // EMC Education Services - 2015
19. "Інформаційний пошук: алгоритми та евристики" // Девід А. Гроссман, Офір Фрідер та Юссі Карлгрен - 2004
20. «Обробка тексту в Python» // Девід Мерц – 2003
21. «Практична обробка природної мови: вичерпний посібник із створення реальних систем НЛП» // Совмія Ваджала, Бодхісатва Маджумдер та Анудж Гупта – 2021
22. «Основи глибокого навчання для обробки природної мови» // Йоав Голдберг і Грем Герст - 2021

ДОДАТОК А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«ПІДВИЩЕННЯ ЯКОСТІ ПОПЕРЕДНЬОЇ ОБРОБКИ ДАНИХ В ТЕКСТАХ НИЗЬКОЇ ЯКОСТІ»

Виконав: студент групи ПДМ-64 Новицький Сергій Сергійович

Керівник: к.т.н., доц., доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ - 2024

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: зменшення розміру вектору представлення тексту низької якості за рахунок усунення шуму та надлишкової інформації методами попередньої обробки текстів.

Об’єкт дослідження: процес попередньої обробки даних у текстах низької якості.

Предмет дослідження: методи і технології обробки текстових даних.

ВИДИ ШУМУ В ТЕКСТАХ НИЗЬКОЇ ЯКОСТІ

Вид шуму	**Опис**
Орфографічний шум	Опечатки: Невірно написані слова через помилки при наборі. Помилки в написанні: Невірно використані слова чи неправильно сформульовані фрази.
Синтаксичний шум	Граматичні помилки: Неправильна граматика або некоректний синтаксис. Неправильна структура речення: Речення, які можуть бути важко зрозуміти через некоректну структуру. Використання токенів що не є словами природної мови: числа, спецсимволи тощо.
Семантичний шум	Невірне використання слів: Використання слів з неправильним значенням або в контексті, що не пасує. Неясність: Фрази, які можуть бути двозначними чи неповніми у своєму значенні. Наявність скорочень чи абревіатур.
Лексичний шум	Використання вульгарної лексики: Неприйнятна мова чи слова, які можуть викликати обурення. Неформальна лексика: Використання слів, які не відповідають формальному стилю.
Логічний шум	Відсутність чітких або послідовних логічних зв'язків між ідеями в тексті.
Шум в інформаційних даних	Невірна інформація: Дані, які невірно представлені чи неперевірені. Відсутність доказів: Виборче використання інформації без підтримки доказами.

3

МЕТОДИ ПОПЕРЕДЬОЇ ОБРОБКИ ТЕКСТОВИХ ДАНИХ

Метод	**Опис**	**Ефект впливу на текст**
Спелчекінг	Процес виявлення та виправлення орфографічних помилок у текстовому документі.	Пошук та випралення офрографічних помилок в тексті.
Стемінг	Це процес скорочення слова до основи шляхом відкидання допоміжних частин, таких як закінчення чи суфікс.	Зменшення довжини слів за рахунок перетворення до базової форми.
Видалення стоп-слів	Вилучення загальноприйнятих слів, які не несуть значущої інформації.	Зменшення векторного представлення вхідного тексту
Виявлення та вилучення HTML-тегів	Вилучення HTML-тегів, що можуть містити непотрібну інформацію.	Зменшення векторного представлення вхідного тексту
Аналіз частоти слів та фраз	Визначення частоти вживання слів та вилучення тих, що вживаються занадто часто.	Розрахунок векторного представлення текста
Застосування правил фільтрації	Встановлення правил для визначення непотрібної інформації та її вилучення.	Зменшення векторного представлення вхідного тексту
Збагачення даних	Синоніми, розкриття абервіатур, заміна представленнями	Збільшення корисної інформації в тексті.

СХЕМА ТОКЕНІЗАЦІЇ ТЕКСТУ

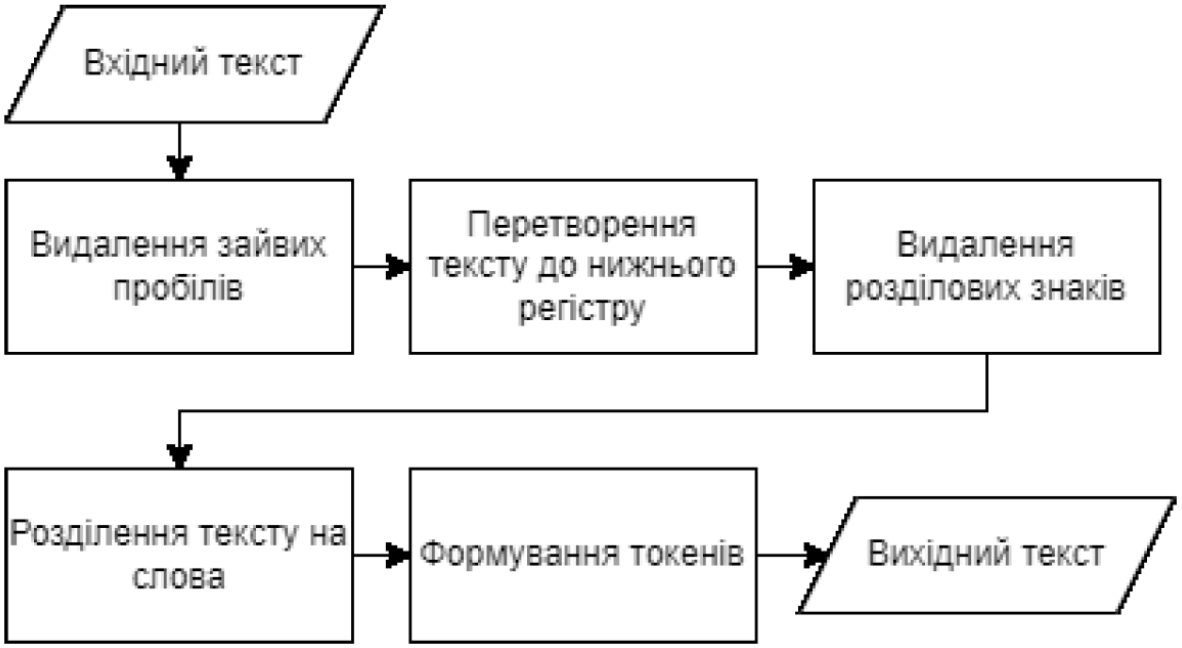
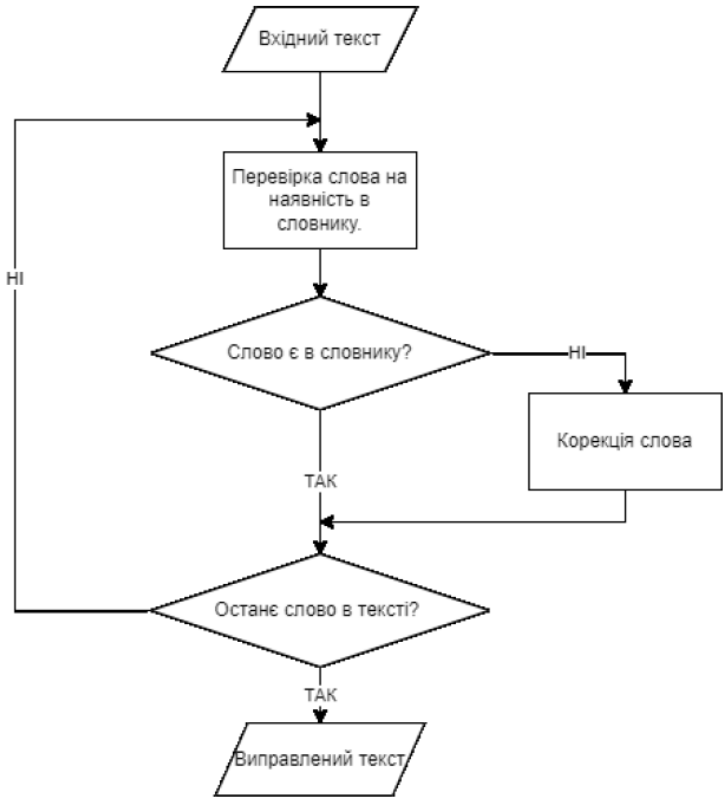


СХЕМА ВИПРАВЛЕННЯ ОРФОГРАФІЧНИХ ПОМИЛОК В ТЕКСТІ



ВЕКТОРИЗАЦІЯ ТЕКСТУ МЕТОДОМ TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

Метод **Term Frequency-Inverse Document Frequency (TF-IDF)** є статистичною мірою важливості слова в тексті в контексті колекції документів.

Term Frequency (TF) вимірює, наскільки часто слово з'являється у конкретному тексті:

$$\mathrm{TF}(t, text) = \frac{\text{Кількість входжень слова } t \text{ у тексті}}{\text{Загальна кількість слів у тексті}}. \quad (1)$$

Inverse Document Frequency (IDF) вимірює, наскільки важливе слово в контексті всіх документів:

$$\mathrm{IDF}(t, Texts) = \log\left(\frac{\text{Загальна кількість текстів у колекції } N}{1+\text{Кількість текстів, які містять слово } t}\right). \quad (2)$$

TF-IDF:

$$\text{TF-IDF}(t, text, Texts) = \mathrm{TF}(t, text) \times \mathrm{IDF}(t, Texts). \quad (3)$$

Векторизація тексту:
Для векторизації тексту створюється вектор, де кожна компонента відповідає значенню TF-IDF для конкретного слова у тексті.

Розмір вектору дорівнює кількості унікальних слів у тексті.

ПРИКЛАД ОБРОБКИ ТЕКСТУ

Вхідний текст

Німеччина передала Україні нову партію зброї. У пакеті 3 зенітні установки Gepard і 30 240 боєприпасів до них, — уряд країни.

Найжирніше: 10 БПЛА Vector,
боєприпаси для танків Leopard 2A6, 8 вантажівок Zetros, 2 машини розмінування WiSENT 1 MC, 2 системи захисту зв’язку AMPS для вертольотів.

Вихідний текст

німеччина передати україна нова партія зброя пакет зеніт установка gepard боєприпаси уряд країна головне безпілотний літати апарат vector боєприпаси танк leopard вантажівка zetros машина міна wisent mc система захист зв’язок amps вертоліт

Числовий вектор:

amps	gepard	leopard	…	**боєприпаси**	…	уряд	установка	зв`язок
0.169031	0.169031	0.169031	…	**0.338061**	…	0.169031	0.169031	0.169031

Розмір вектору до обробки – 38 , після – 32.

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Кількість токенів вхідного тексту	Кільк. стоп-слів	Кількість токенів з помилками	Кількість токенів, що не є словами природної мови	Кількість абревіатур і скорочень	Кількість вульгарних слів	Кількість жаргонних слів	Розмір вектору після обробки
5	2	0	0	0	0	0	3
15	3	0	0	0	2	0	9
20	5	2	2	0	0	2	11
20	4	0	3	0	0	0	10
25	6	2	1	0	0	0	13
34	18	0	1	0	0	0	14
77	37	2	9	0	0	0	26
146	50	6	21	3	0	0	77

ВИСНОВКИ

1. Проведено аналіз видів тексту за рівнем їх структурованості, визначено проблематику обробки неструктурованих текстів. Проаналізовано методи та технології попередньої обробки текстових даних природньою мовою, та інші методи, що можуть бути використані для підвищення якості попередньої обробки тексту.
2. Проаналізовано види та особливості шумів, які можуть зустрічатись в текстах низької якості. Детально розглянуто найбільш поширені шуми, серед яких: синтаксичний шум, семантичний шум, лексичний шум, орфографічний шум, шум в інформаційних даних.
3. Розроблена загальна схема обробки тексту низької якості, яка включає наступні блоки: токенізація, виправлення орфографічних помилок, видалення стоп-слів, обробка абревіатур, скорочень, вульгаризмів, жаргонізмів і блок векторизації та аналізу результату.
4. Практична реалізація методики виконана мовою програмування Python з використанням бібліотеки обробки текстових даних NLTK, бібліотеки Pyspellchecker для виправлення орфографічних помилок в тексті, бібліотеки scikit-learn та обрахування векторного представлення тексту.
5. Проведено моделювання, яке показало що в середньому розмір вектору, яким представляється текст після попередньої обробки, становить близько 49% по відношенню до вхідної кількістю токенів в тексті, при цьому: для коротких текстів – 46%, для середніх текстів – 55%, для довгих - 51%.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Стаття:

Новицький С.С., Золотухіна О.А. “Підвищення якості попередньої обробки даних в текстах низької якості” // Зв'язок, №1, 2024. Подано до друку

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б
ФРАГМЕНТИ ЛІСТИНГУ

```
import nltk
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import RegexpTokenizer
from spellchecker import SpellChecker
from nltk.corpus import stopwords
from nltk.stem.porter import *
nltk.download('punkt')
nltk.download('stopwords')

from nltk.tokenize import word_tokenize

abbreviations = {
    'dr': 'doctor',
    'ms': 'miss',
    'usa': 'United States of America',
    'uav':'Unmanned aerial vehicle',
    'amps':'Advanced Mobile Phone System'
}

vulgarisms = {

}

jargonisms = {

}

def remove_non_language_symbols(text):
    # Підрахунок кількості не-мовних символів перед видаленням
    count_non_language_symbols = len(re.findall(r"[^a-яА-Яa-zA-Z\s.,!?]", text))

    # Видалення символів, крім букв (латиниця або кирилиця), цифр, пробілів та пунктуаційних знаків
    cleaned_text = re.sub(r"[^a-яА-Яa-zA-Z\s.,!?]", "", text)

    return cleaned_text, count_non_language_symbols

def replacer_from_list(tokens, list_of):
    updated_tokens = []
    replaced_count = 0  # Лічильник замінених слів

    for token in tokens:
        for item, replacer in list_of.items():
            if item in token:
                token = token.replace(item, replacer)
                replaced_count += 1  # Збільшуємо лічильник
        updated_tokens.append(token)

    return updated_tokens, replaced_count

def delete_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    new_text = tokenizer.tokenize(text)
    spaces = RegexpTokenizer(r'\s').tokenize(text)
```

```
    # Підрахунок кількості пунктуаційних символів
    punctuation_count = len(text) - len(''.join(new_text)) - len(''.join(spaces))
    return new_text, punctuation_count

def correct_misspelled(misspelled_tokens):
    spell = SpellChecker()

    # Знаходження слів з помилками серед токенів
    misspelled_words = spell.unknown(misspelled_tokens)

    # Проходимо через кожен токен і виправляємо помилки
    corrected_tokens = []
    corrections_count = 0  # Лічильник виправлених слів
    for token in misspelled_tokens:
        if token in misspelled_words:
            # Отримання найкращої пропозиції для кожного слова з помилкою
            correction = spell.correction(token)
            corrected_tokens.append(correction)
            corrections_count += 1  # Збільшуємо лічильник
        else:
            corrected_tokens.append(token)

    return corrected_tokens, corrections_count

input_text = """These texts were chosen to show an example of how the algorithm works.
The texts specifically contain spelling errors in words, various punctuation symbols for numbers and
other symbols.
Also, this text contains abbreviations and abbreviations.
Germany handed over a new batch of weapons to Ukraine. The package includes 3 Gepard anti-aircraft
guns and 30240
ammunition for them, the government of the country.
Text 1:The fattest: 10 Vector UAVs,
ammunition for Leopard 2A6 tanks, 8 Zetros trucks,
2 WiSENT 1 MC demining vehicles, 2 AMPS helicopter communications protection systems.
Text 2:I advocate for adventure and believe my walk with Jesus is the most incredible adventure I
have ever embarked on.
This adventure began when I chose Jesus early in my adulthood; since then, I have never looked back.
I enjoy camping, coffee, and creativity."""

text, count_non_language_symbols = remove_non_language_symbols(input_text)


tokens = word_tokenize(text, language='english')
input_lenght = len(tokens)

without_punkt, punctuation_count = delete_punctuation(text.lower())

no_abbreviations, abbreviations_count = replacer_from_list(without_punkt, abbreviations)

no_vulgarisms , vulgarisms_count =  replacer_from_list(no_abbreviations, vulgarisms)

no_jargonisms , jargonisms_count =  replacer_from_list(no_vulgarisms, jargonisms)

corrected_text, corrections_count = correct_misspelled(no_jargonisms)

temp_lenght = len(no_jargonisms)
filtered_text = [word for word in corrected_text if word.lower() not in stopwords.words('english')]
filtered_count = temp_lenght - len(filtered_text)
```

```
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(plural) for plural in filtered_text]
output_leght = len(stemmed_tokens)



document = [" ".join(stemmed_tokens)]
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(document)

tfidf_array = tfidf_matrix.toarray()
feature_names = vectorizer.get_feature_names_out()

vector_sizes = tfidf_matrix.get_shape()

print ('Вхідний текст: ')
print(input_text)
print()

# print('Токенізований текст:')
# print(tokens)

# print("Текст без символів пунктуації та в нижньому регісті:")
# print(without_punkt)
# print(no_abbreviations)
print('Кількість видалених знаків пунктуації: ' + str(punctuation_count))
print('Кількість видалених символів не природної мови: ' + str(count_non_language_symbols))


# print("Розкриття скорочень і абревіатур")
# print(no_abbreviations)
print('Кількість розкритих абревіатур: ' + str(abbreviations_count))
# print("Заміна вульгаризмів:")
# print(no_vulgarisms)
print('Кількість замімених вульгаризмів: ' + str(vulgarisms_count))
# print("Заміна жаргонізмів")
# print(no_jargonisms)
print('Кількість замімених жаргонізмів: ' + str(jargonisms_count))

# print("Відкорегований текст:")
# print(corrected_text)
print('Кількість орфографічних виправлень: ' + str(corrections_count))

# print("Текст без стоп-слів:")
# print(filtered_text)
print('Кількість видалених стоп слів: ' + str(filtered_count))

print("Розмір векторного представлення перед обробкою:"+ str(input_lenght))
print("Розмір векторного представлення після обробки:" + str(output_leght))
print()

print("Текст після обробки:")
print(' '.join(stemmed_tokens))
print("Результат векторизація тексту:")
print(feature_names)
print(tfidf_array)
```