Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Аналіз неструктурованих текстів, записаних природньою мовою

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи
К-т фіз. – мат наук, доцент
Жежерун О.П.

_____
(підпис)
“____” _____ 2021 р.

Виконав студент ІПЗ-4:
Вєдєніков М.В.
“____” _____ 2021 р.

Київ 2021

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к. ф.-м. н. С. С. Гороховський

_____

(підпис)

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Дата видачі “____” _____ 2021 р. Керівник _____

(підпис)

Завдання отримала _____

(підпис)

Зміст

Календарний план виконання робот

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	09.10.2020	
2.	Пошук літератури за темою роботи	10.11.2020	
3.	Ознайомлення з науковою літературою	11.12.2020	
4.	Огляд існуючих методів токенізації	05.01.2021	
5.	Огляд існуючих методів нормалізації	12.01.2021	
6.	Огляд існуючих методів векторизації	15.01.2021	
7.	Вивчення специфічності реалізації алгоритмів для української мови	22.01.2021	
8.	Визначення структури програми	01.02.2021	
10.	Реалізація практичної частини	11.02.2021	
11.	Написання першої частини курсової роботи	10.03.2021	
12.	Написання другої частини курсової роботи	24.03.2021	
13.	Написання третьої частини курсової роботи	29.03.2021	
14.	Написання висновків курсової роботи	05. 04.2021	
15.	Перегляд змісту роботи з керівником	06.04.2021	
16.	Внесення змін до роботи	07.04.2021	
17.	Створення презентації	11.04.2021	
18.	Завантаження курсової роботи	12.04.2021	

Студент Вєдєніков М.В.

Керівник Жежерун О.П.

“____” _____

Вступ

На сьогоднішній день кожна система стикається з величезними об’ємами текстової інформації. Найчастіше це тексти, шо не мають загальної структури, тобто написані природньою мовою, які не можуть сприймати традиційні алгоритми. Для вирішення цієї проблеми існує спеціальний напрям під назвою обробка природніх мов (Natural Language Processing, NLP). Він займається створенням систем, що допомагають в аналізі текстових документів.

Даний аналіз застосовується з метою спрощення роботи з накопиченими матеріалами, описаними різноманітними мовами світу. Це питання було досліджено великою кількістю вітчизняних та зарубіжних вчених, однак результати не покривають усі аспекти специфічних мов. Більшість алгоритмів направлено на роботу з латиницею або ж російською мовою.

Метою дослідження є реалізація етапів попередньої обробки для неструктурованих текстів написаних українською мовою для подальшого аналізу. Предметом дослідження є реалізація системи для попереднього аналізу тексту підручників з геометрії для полегшення процесу підготовки учнів до ЗНО.

Завданнями даної роботи є:

1. Проаналізувати необхідні етапи попередньої обробки неструктурованих текстів;
2. Розглянути основні етапи та варіанти токенізації тексту;
3. Дослідити існуючі підходи до нормалізації тексту;
4. Визначити особливості алгоритмів Стемінга;
5. Проаналізувати існуючі методи векторизації даних;
6. Дослідити приклади застосування обробки природніх мов;
7. Реалізація алгоритмів попередньої обробки україномовних наукових текстів.

Розділ 1. Обробка неструктурованих текстів, записаних природньою мовою

Початковий, звичний для людини текст є неструктурованим, і не може бути використаним для подальшого аналізу комп'ютером. Він уявляє собою набір речень, що складаються з слів, різної форми та знаків пунктуації. Для того щоб привести початковий текст до певної форми, структури, зазвичай необхідна попередня обробка тексту, що складається з етапів токенізації та нормалізації.

1.1. Токенізація

Задля надання певної структури попередньо неструктурованому тексту, його розбивають на менші блоки, що мають назву токени. Ці токени можуть відрізнятися своїм розміром, залежно від запланованого подальшого аналізу. Найчастіше сегментами тексту виступають речення, слова або символи.

1.1.1. Токенізація речень

Токенізація (Сегментація) за реченнями – процес розбиття неструктурованого тексту на речення. На перший погляд задача є доволі простою. Адже для більшості мов таких як англійська, українська, тощо, речення виокремлюються певним знаком пунктуації – крапкою, знаком питання, знаком оклику, тощо. Але якщо розглянути детальніше, то крапка не завжди буде позначати межу речення, бо вона може бути позначенням і абревіатури, скорочення, числа, тощо[2] (див табл. 1).

Таблиця 1.1.

Винятки для крапки при токенізації на речення

Виняток	Приклад	Регулярний вираз

Акроніми	U.S.A	([A-Z][.][A-Z][.](?:[A-Z][.])?)
Префікси	Mr. John	(Mr|St|Mrs|Ms|Dr)[.]
Число	9.0 31.006%	/^\d*\.+\d*$/
Суфікси	John Johnson Jr. Nike Inc.	(Inc|Ltd|Jr|Sr|Co)
Посилання	Google.com wikipedia.org	[.](com|net|org|io|gov|ru|ua)

Для виділення речень можуть бути використані наступні варіанти алгоритмів на основі:

- Правил
- Регулярних виразів
- Машинного навчання
- Дерев прийняття рішень

Один з можливих варіантів реалізації для токенізації речення полягає у заміні всіх виняткових точок на певний символ на основі регулярних виразів описаних у табл.1 Наприклад на ‘<prd>’, а всіх валідних меж речень на ‘<stop>’. Потім сформування списку речень, розділенням за ‘<stop>’ та повернення знаку крапки для винятків.

1.1.2. Токенізація слів

Токенізація за словами – процес розбиття неструктурованого тексту на слова. Найчастіше подібну розподіл виконується за допомогою регулярних виразів. Тіло виразу залежить від цілі поставленого алгоритму, інколи знаки пунктуації залишаються і вважаються за окремі токени, в інших випадках потрібно щоб результуючий список складався лише з слів.

Часто розбиття виконується саме на основі знаків пробілу. Є декілька основних проблем, що можуть виникнути при використанні такого підходу[1]:

1) При розподілі за пробілами можуть виникати ситуація коли, токени, які логічно було б розглядати як одне ціле, розділяється на декілька. Під такі ситуації підпадають сталі словосполучення, такі як: назви міст, особисті імена, скорочення, тощо. Наприклад: “New York”, “Mr. John Walker”, “Mr. John Walker”.
2) В деяких мовах присутня проблема поділу апострофом. Наприклад для англійської мови в деяких ситуаціях апостроф потрібно було б розглядати як роздільник, а в деяких він виступає в ролі постфіксу, шо показує приналежність. Наприклад: “I’m”, “What’re” – слід розділити на 2 слова, а “Alina’s food” – не потрібно ділити. Також ця проблема виникає для французької мови, де апостроф використовується для скорочення артикля перед словом.
3) Проблема опрацювання слів, що мають в собі дефіс. Зокрема в англійській мові дефіс можуть використовувати для поєднання іменників в назвах, між голосними літерами в словах (co-brending) або для групування слів. Наприклад: “Coca-cola”, “co-brending” – мають розглядатись як один токен, а у випадку “state-of-the-art” – як декілька токенів.
4) В німецькій мови в одному слові можуть зберігатися одразу декілька токенів. Наприклад: “Lebensversicherungsgesellschaftsangestellter”, що складається з наступних токенів [життя, страхування, компанії, працівник]
5) В японській та китайських мовах взагалі немає пробілів між словами.

1.1.3. Видалення стоп-слів

Стоп-слова – це слова, які зазвичай не несуть ніякого смислового навантаження і їх видалення не матиме ніякого впливу на опрацювання тексту для вказаної цілі [3]. Найчастіше це слова які зустрічаються багаторазово та

створюють зайвий шум. Тому їх видалення покращить результати та пришвидшить виконання поставлених задач.

Інколи вилучення стоп-слів є необхідним етапом. Наприклад при повнотекстовому пошуку система може повернути майже всі документи, якщо в запиті присутні сполучники чи прийменники.

Найчастіше до стоп-слів належать прийменники, суфікси, вигуки, цифри тощо(див. рис. 1). При цьому слід розуміти, що не існує сталого універсального списку стоп-слів, все залежить від конкретної предметної області [3].

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',
'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were',
'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after',
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there',
'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only',
'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm',
'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

Рис. 1 Список стоп-слів для англійської мови [19]

1.2. Нормалізація

Оскільки вхідні тексти містять в собі різні граматичні форми одних і тих же слів, які не несуть ніякої корисної інформації, після розділення початкового тексту на необхідні токени, їх потрібно нормалізувати. Це і є наступним етапом попередньої обробки тексту.

Під нормалізацією розуміють зведення слів до нормальної форми, де нормальна форма - це канонічна, початкова форма слова. Для іменників під початковою формою мають на увазі форму однини, що стоїть в називному відмінку, для прикметників - прикметник чоловічого роду, однини, в називному відмінку без прийменників [4]. Наприклад: Фруктами -> фрукт, красивими -> красивий.

Для виконання нормалізації можна виділити два основних підходи: стемінг і лематизацію. Обидва ці методи ставлять перед собою за мету привести всі використанні в тексті словоформи до однієї, але використовують дещо різні методи для цього.

1.2.1. Лематизація

Лематизація – тонкий процес, цілю якого є виділення леми слова. Під лемою розуміють базову, або канонічну форму вхідного слова, тобто ту, яку ми можемо знайти в словнику. Цей метод зазвичай працює на основі словника, де зберігається інформація про слова в їх початкових формах[3].

Основна складність цього підходу полягає у формуванні словника для кожної мови. Адже в ньому має зберігатися інформація про основи слів, список всіх можливих словоформ а також частина мови. Якщо ви вже маєте сформований словник під необхідну мову, реалізація значно спрощується.

Один з основних етапів, який необхідний для збільшення точності нормалізації в алгоритмі лематизації – це виділення частини мови для слова. Оскільки залежно від частини мови, до одного і того самого слова можуть бути застосовані різні правила. Наприклад: якщо взяти слово “following”, залежно від

контексту, воно може бути і іменником - “he has a huge following”(з англ.” У нього є величезна кількість прихильників”), і дієсловом – “he was following me” (з англ.” Він слідкував за мною”), і прикметником – “following week” (з англ.” Наступного тижня”). Якщо це іменник чи прикметник , лематизація має повернути “following”, а якщо дієслово - “follow”.

Можна виокремити 4 основні методи реалізації для цього: ручний, на основі правил, стохастичні / імовірнісні методи та глибинного навчання (див табл 2)[6].

Таблиця 1.2.

Основні методи визначення частини мови для слів

Назва методу	Опис методу
Ручний метод	Люди, які володіють інформацією про правила синтаксису, мають визначати частини мови для кожного слова у фразі вручну.
Метод на основі правил	Перший автоматизований спосіб позначення слів частиною мови. Складається з ряду правил (Наприклад: для англійської мови, якщо попереднє слово артикль, а наступне – іменник, то обране зараз слово – прикметник). Має складатися спеціалістом та може з легкістю бути ускладнене.
Стохастичні / імовірнісні методи	Автоматизовані способи присвоєння частини мови слову на основі ймовірності того, яким може буде це слово зважаючи на послідовності наступних та/або попередніх слів. Ці методи є найкращими та найбільш використаними на сьогодні.
Методи глибинного навчання	Методи, що використовують прийоми глибинного навчання для визначення частини мови слова. На сьогодні ці методи не продемонстрували ніяких

	переваг перед імовірнісними – вони, здебільшого показують однаковий рівень точності, ціною складнішого та довшого навчання

Особливістю лематизації є її точність в порівнянні з стемером. Звісно використовуючи цей метод, слід розуміти про різницю в витраченому часі на реалізацію, особливо для не популярних мов. А також про час, витрачений алгоритмом на пошук необхідних слів в словнику. Саме через цей нюанс, метод лематизації найчастіше використовують, коли необхідне саме точне визначення слова, і перед застосунком не стоїть за мету отримання результату в найкоротші можливі терміни. Наприклад у алгоритмах Word Sense Disambiguation (відкрита проблема, пов'язана з визначенням того, яке значення слова вживається в реченні).

1.2.2. Стемінг

Стемінг – це процес знаходження основи слова для заданого вхідного токену, що базується на грубому евристичному підході. Стематизація відбувається на основі аналізу всіх граматичних форм слова, відсікаючи суфікси та закінчення. Слід зауважити, що результати виконання не можна вважати морфологічним коренем слова, незважаючи на те, що інколи отримана основа може збігатися з ним. Важливим є також те, що для використання даного алгоритму неважливий контекст[4].

Алгоритм стемінга – це конкретний спосіб вирішення задачі пошуку основ слів, а стемер – це конкретна реалізація. До стемера зазвичай на вхід подається список токенів, а на виході ми отримуємо список стем. Тобто послідовності символів, що залишилися після видалення частин рядків та яка виконує функцію ототожнення токенів[4].

Рис. 2 Основні типи алгоритмів стемінгу

Загалом, алгоритми стемінгу можна класифікувати на три групи: відсікаючі, статистичні та змішані (див. рис. 2). Кожна з цих груп має типовий спосіб пошуку основи слова [7]:

Відсікаючі – як можна зрозуміти з назви цей метод полягає у видаленні суфіксів чи префіксів слова (афіксів). До них належать [1]:

1) Ловінса – ідея полягає у видаленні найдовшого суфіксу, шукаючи збіг за таблицею, де зберігаються закінчення, умови та правила. Перевагами цього методу є те що він є швидким, оскільки виконується до першого відсікання, він коректно опрацьовує подвоєння та багато нестандартних множин. Але він є дуже залежним від словника, що використовується та часто не може знайти стему.
2) Портера – ідея полягає у тому, що більшість суфіксів складаються з комбінацій простіших суфіксів. Він складається з 5 кроків, на кожному з яких видаляється відповідний суфікс, якщо він був знайдений. Портером було створено детальний фреймворк для стемінгу, що має назву “Snowball”. До основних переваг цього методу належать те, що він генерує найкращий результат, порівняно з іншими та те що він є незалежним до мови. До недоліків можна віднести, що його результуючі стеми не завжди є реальними словами.

3) Пейса/Хуска – простий алгоритм, що перевіряє закінчення слова за 120 правилами та видаляє або заміняє його. Він є дуже простим в реалізації, але повільний та часто виникає проблема видалення зайвих частин основи.
4) Доусона – розширення методу Ловінса, особливістю якого є організація у вигляді дерева, для швидшого доступу. Він відпрацьовує коректніше через більшу кількість суфіксів аніж алгоритм Ловінса, але є дуже складним і не має стандарту імплементації.

Статистичні – стемери, що створюються на основі статистичного аналізу та технік. Більшість з них теж видаляють афікси, але вже після застосувань певних статистичних процедур. До них належать [7]:

1) На основі Н-грамм – ідея полягає у розбиті слів на н-грамми, тобто послідовність сусідніх символів довжиною н, тоді можна буде виконувати пошук подібності для слів, спираючись на подібності н-грам цих слів. Перевагою цього методу є те, що він не залежить від мови, а тому дуже корисний в багатьох додатках. Але він є не дуже практичним, оскільки потребує велику кількість пам’яті для зберігання н-грамм.
2) Прихована модель Маркова – стемер, що базується на використанні автоматів скінченних станів, де переходи між станами керуються функціями ймовірності. На кожному переході, кожний стан представляє собою символ з певною вірогідністю. Перевагою цього методу э те, що в його основі лежить навчання без нагляду, а тому він є незалежним від мови, але є дуже складним в розробці та інколи виникає проблема видалення зайвих символів.

Змішані – стемери, що поєднують в собі елементи різних методів. Здебільшого до них належать контекстно залежні алгоритми. Прикладом цієї категорії можна вважати [20]:

1) Корпусний – метод який відноситься до автоматичних модифікацій додаванням класів зав’язків – щоб слова, зводилися до однієї стеми, на

основі заданого корпусу тексту. Основна ідея полягає у тому, що словоформи які мають бути спільними для заданого корпусу будуть зустрічатися в даному документі декілька разів в тому самому корпусі. Цей метод дозволяє зменшити виникнення ситуацій утворення спільних стем для не суміжних слів, але для цього потрібно розробити статистичну міру для кожного з корпусу.

2) Флективні та похідні – цей підхід складається з двох етапів. Під час першого видаляються флективні суфікси, тобто ті шо створюють додаткові форми слова. Потім через перевірку в спеціальному словнику, спираючись на контекст(корпус) для обраного слова, відбуваються перетворення, необхідні для створення з отриманих стем реальних слів та виправлення винятків правопису. Завдяки цьому методу можна отримати морфологічно коректні стеми, але є неефективним для великих документів, не завжди дає якісний результат та може працювати лише з заздалегідь визначеним лексиконом.
3) Контекстно залежний – стемер, який використовується для запитів в пошукових системах. Ідея полягає у створенні морфологічних варіантів для слів зі вхідного запиту, які використовуються для подальшого пошуку, в системі. Основними етапами цього алгоритму є: створення стем для слів запиту за допомогою методу Портеру, визначення контексту для кожного слова в запиті, на основі статистики, а вже потім виконується пошук в документах сходження для всіх морфологічних форм слова в заданому корпусі(контексті). Цей метод підвищує точність в пошукових системах, але є складним в реалізації, та є досить вузько направленим.

На сьогодні найпопулярнішим є алгоритм Портера. Він став стандартом для англійської мови, і в свою чергу став основою для стемінгу інших мов. Для деяких з мов спільним залишилось лише використання малого суфіксального словника. Портером власноруч було створено покращену та адаптовану під інші

мови версію свого алгоритму, під назвою “Snowball”. Він був описаний на мові програмування високого рівня та містив однозначний опис правил [8].

Оскільки стемер зазвичай базується на евристиці, він має свої недоліки. Виділяють дві основні проблеми: надстемінг і недостемінг.

Надстемінг виникає при видаленні занадто великої кількості символів від слова. Це спричиняє утворення стем, що не мають ніякого сенсу, тобто весь сенс втрачається або змінюється. Також можливий варіант, коли декілька слів ототожнюється стемами (зводяться до однакових основ), хоча їх варто було б розділити на окремі. Наприклад, якщо взяти слова “вода”, “води”, “водій”, “водія” неякісний алгоритм стемінгу зведе всі ці чотири слова до однієї основи “вод”, що призведе до подальшого хибного використання. Для попередження такого трактування, кращим вирішенням може слугувати зведення “вода” і “води” до “вод”, а “водій”, “водія” до “воді”[4].

Недостемінг представляє собою повну протилежність до описаної вище проблеми. Вона виникає у випадку недостатнього видалення афіксів, що призводить до віднесення морфологічних форм одного слова до різних груп. [4]

Алгоритми стемінгу намагаються зменшити кількість відповідних помилок. Але це не є тривіальним завданням, оскільки вирішення однієї такої проблеми додаванням, зміною або видаленням правила, може призвести до появи двох інших того ж або іншого типу. Тому алгоритми зазвичай адаптують під певні предметні області та залежно від мети використання.

Розділ 2. Використання нормалізованих даних для аналізу тексту

2.1. Векторне представлення

Після того як текст було нормалізовано, з нього було видаленні зайві стоп-слова та пунктуація - він вже майже готовий для опрацювання алгоритмами машинного навчання. Останнім необхідним етапом є векторизація. Під векторизацією розуміють кодування текстових даних в вигляді чисел, які в подальшому зможуть бути використанні в алгоритмах [21]. Для такої конвертації використовують спеціальні моделі. Найбільш популярними є: мішок слів, Tf-Idf, N-грами та Words2Vec.

2.1.1. Мішок слів

Мішок слів (Bag of Words) – один з способів вилучення функцій з тексту для використання в моделюванні. Цей підхід є дуже простим та гнучким, тому його можна з легкістю змінювати та адаптувати при необхідності. Ідея цієї моделі полягає у репрезентації описуваного тексту у вигляді вектора входження кожного слова [22].

Слід зауважити, що будь-яка інформація про порядок або структуру слів ігнорується. Саме тому цей метод називають мішком слів, оскільки він лише намагається визначити чи зустрічається дане слово в документі, а не де саме воно розташовано в ньому.

Для використання цієї моделі необхідно :

- Визначений словник відомих токенів (слів);
- Міра (степінь) присутності відомих слів.

Наприклад, якщо взяти за основу 4 документи:

Таблиця 2.1

Приклад колекції документів

Документ №	Текст документу
1	“Андрій любить їсти рибу, а Юля любить їсти яблука.”
2	“Андрій не хотів би їсти яблука.”
3	“Юля любить рибу.”
4	“Аня любить огірки.”

Потім створимо словник слів, який складається з унікальних слів. В нашому випадку це: *“Андрій”, “любить”, “їсти”, “*рибу*”, “а”, “Юля”, “яблука”, “не”, “хотів”, “би”, “Аня”, “огірки”*

Словник складає 12 слів в той час як корпус документів має 20.

Наступним етапом є оцінка входження слів у кожному документі. Найпростішим варіантом реалізації є відмічання наявності слів, тобто ставити 1, якщо є слово і 0, якщо його немає.

Для першого документа отримуємо: *“Андрій” - 1, “любить” - 1, “їсти” - 1 , “*рибу*” - 1 , “а” - 1, “Юля” - 1, “яблука” - 1, “не” - 0, “хотів” - 0, “би” - 0, “*Аня*” - 0, “огірки” – 0*

Якщо зобразити у вигляді бінарного вектору, то для документу 1 буде відповідно: *[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0].*

Після застосування цього кроку для кожного з документів отримаємо:

Таблиця 2.2

Представлення кожного документу у вигляді бінарних векторів

	Андрій	*любить*	*їсти*	рибу	*а*	*Юля*	*яблука*	*не*	*хотів*	*би*	Аня	*огірки*
1	1	1	1	1	1	1	1	0	0	0	0	0
2	1	0	1	0	0	0	1	1	1	1	0	0
3	0	1	0	1	0	1	0	0	0	0	0	0

4	0	1	0	0	0	0	0	0	0	0	1	1

Для оцінки присутності слів в документі використовують не лише бінарний підхід (1 – присутнє слово, 0 – відсутнє). Існують декілька інших варіантів. Найпоширенішими є [21]:

1) За кількістю – підраховується скільки разів кожне слово зустрічається в документі. Для наведеного вище прикладу векторне представлення мало б наступний вигляд:

Таблиця 2.3

Представлення кожного документу у вигляді бінарних векторів

	Андрій	*любить*	*їсти*	рибу	*а*	*Юля*	*яблука*	*не*	*хотів*	*би*	Аня	*огірки*
1	1	2	2	1	1	1	1	0	0	0	0	0
2	1	0	1	0	0	0	1	1	1	1	0	0
3	0	1	0	1	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	1	1

2) За частотою – підраховується як часто кожне з слів зустрічається в тексті, по відношенню з загальною кількістю слів.

2.1.2. TF-IDF

Для описаного вище частотного варіанту реалізації мішка слів виникає проблема – найбільшу оцінку отримують слова з найбільшою частотою. Ці слова не завжди будуть нести так багато інформаційного виграшу для моделі, ніж менш використовувані. Для вирішення цієї проблеми використовують спеціальну статистичну міру для оцінки важливості слова в документі, що є частиною колекції чи ми корпуса - Term Frequency - Inverse Document Frequency (TF-IDF) [21].

Основна ідея полягає у тому, що якщо слово зустрічається часто в якомусь з документів, і при цьому зустрічається рідко в всіх інших документах, то це слово має велику значимість для обраного документа.

Term Frequency(TF) – частота терміну, яка вимірює наскільки часто зустрічається даний термін в обраному документі. Оскільки в великих документах термін буде зустрічатися більшу кількість раз ніж в маленьких, просто кількість знаходжень цього слова нам не вистачає. Тому використовують відносну частоту – відношення числа входження слова до загальної кількості слів в документі [23].

$$TF(\boldsymbol{term}) = \frac{Number\ of\ times\ \boldsymbol{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

Формула 1 – визначення Частоти слова

Якщо взяти за приклад документи з таблиці 4, то отримаємо наступний результат підрахунку частоти для кожного слова:

Таблиця 2.4

Term Frequency для кожного терма в документі (“0“ означає, що терм не зустрічається в цьому документі)

	Андрій	*любить*	*їсти*	рибу	*а*	*Юля*	*яблука*	*не*	*хотів*	*би*	Аня	*огірки*
1	0.111	0.222	0.222	0.111	0.111	0.111	0.111	0	0	0	0	0
2	0.167	0	0.167	0	0	0	0.167	0.167	0.167	0.167	0	0
3	0	0.333	0	0.333	0	0.333	0	0	0	0	0	0
4	0	0.333	0	0	0	0	0	0	0	0	0.333	0.333

Inverse Document Frequency (IDF) – зворотня частотність документа, що вимірює важливість терміна в конкретній колекції документі. Деякі слова, наприклад прийменники, зустрічаються в усіх документах дуже часто, хоча майже не мають впливу на сенс тексту. Оскільки під час обрахування частоти терміну, ми вважали кожен токен рівнозначним, нам потрібно зменшити оцінку

у словах, які присутні у всіх документах. Для цього і обраховують IDF. Його значення відповідає логарифму від відношення загальної кількості документів в колекції, до кількість документів, в яких присутній обраний термін [23].

$$IDF(\boldsymbol{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \boldsymbol{term}\ in\ it}\right)$$

Формула 2 – визначення зворотної частотності документа

Для документів з таблиці 4 отримуємо наступний результат зворотної частотності документа для кожного терма:

Таблиця 2.5

Inverse Document Frequency для кожного терма в документі (“0“ означає, що терм не зустрічається в цьому документі)

	Андрій	*любить*	*їсти*	рибу	*а*	*Юля*	*яблука*	*не*	*хотів*	*би*	Аня	*огірки*
1	0.693	0.287	0.693	0.693	1.386	0.693	0.693	0	0	0	0	0
2	0.693	0	0.693	0	0	0	0.693	1.386	1.386	1.386	0	0
3	0	0.287	0	0.693	0	0.693	0	0	0	0	0	0
4	0	0.287	0	0	0	0	0	0	0	0	1.386	1.386

В кінці відбувається множення частоти терміну на зворотну частотність документа і отримуємо остаточний результат.

$$TFIDF(\boldsymbol{term}) = TF(\boldsymbol{term}) * \mathrm{IDF}(\mathbf{term})$$

Формула 3 – визначення TF-IDF [23]

Для нашого прикладу маємо:

Таблиця 2.6

TF-IDF для кожного терма в документі

	Андрій	*любить*	*їсти*	рибу	*а*	*Юля*	*яблука*	*не*	*хотів*	*би*	Аня	*огірки*
1	0.077	0.063	0.154	0.077	0.154	0.077	0.077	0	0	0	0	0

2	0.115	0	0.115	0	0	0	0.115	0.231	0.231	0.231	0	0
3	0	0.095	0	0.231	0	0.231	0	0	0	0	0	0
4	0	0.095	0	0	0	0	0	0	0	0	0.462	0.462

2.1.3. N – грами

Як було зазначено раніше, мішок слів не зберігає ніякої інформації про контекст та структуру тексту. Для вирішення цієї проблеми можна використовувати більш витончений підхід який полягає у створенні словника з згрупованих слів – N-грамм.

N-грама – це комбінація з n послідовних елементів. Цими зазвичай виступають слова, але можуть використовуватися й інші об’єкти, такі як букви, числа, цифри тощо. Під час аналізу за “граму” вважають саме слово. Уніграма - це грама, що складається з одного слова, біграма – послідовність двох слів, триграма – трьох слів і так далі [22].

Основна ідея полягає у спрощенні розпізнавання змісту контексту, шляхом збереження суміжних послідовностей слів в тексті, довжиною n. Чим більше є число n, тобто чи довше є грама, ти більше у нас контексту. Адже очевидно, що уніграми зазвичай не несуть у собі багато інформації про контекст, порівняно з біграмами та триграмами.

Наприклад для речення “Андрій не хотів би їсти яблука”.

Були б отриманні наступні біграми: *“Андрій не”, “не хотів”, “хотів би”, “би їсти”, “їсти яблука”.*

Або триграми: *“Андрій не хотів”, “не хотів би”, “хотів би їсти”, “би їсти яблука”.*

2.1.4. Word2Vec

Word2Vec – це один з варіантів реалізацій вкладання слів (word embedding), для розрахунку векторного уявлення слів залежно від контексту. Основна ідея

полягає у знаходженні зв’язків між словами, залежно від контексту, керуючись гіпотезою, що слова в схожих контекстах мають тенденцію бути семантично близькими один одному, тобто мати схожий сенс [10].

Формально це звучить як пошук косинусної близькості між векторами слів, які знаходяться в одному контексті (поряд один з одним), та мінімізація косинусної близькості між векторами слів, що не з’являються поряд один з одним.

Основний принцип роботи цього алгоритму має наступний вигляд [10]:

1) Спочатку відбувається підрахунок кількості скільки разів кожне слово в корпусі (колекції документів) зустрічається
2) Список слів сортується за частотою в корпусі. Відбувається видалення гапаксів, тобто слів, які зустрічаються невелику кількість, задля значного зменшення використання програмою пам’яті.
3) З метою зменшення обчислювальної складності алгоритму та кількості затрачуваного часу, використовується бінарне дерево Хаффмана (Huffman Binary Tree) для зберігання словника.
4) З корпусу витягується певний токен, зазвичай для цього використовують речення, але це може бути інший базовий елемент корпусу, наприклад абзац або розділ, тощо. Для обраного речення виконується етап сабсемплування (Subsampling) – процес видалення з аналізу слів з найбільшою частотою, з метою прискорення процесу навчання алгоритму та значному збільшенню якості отриманої моделі.
5) Далі до отриманого речення застосовують розбиття на n-грами. Цей етап є схожим на описаному в попередньому підході розбиттю, де токенами виступають слова обраного речення. За замовчуванням число n дорівнює 5, але оптимальним вважають значення – 10.
6) До отриманих даних найчастіше застосовується нейронна мережа зворотнього зв’язку (Feedforward Neural Network) з можливим використанням ієрархічного софтмаксу (Hierarchical Softmax), або

негативного семплування (Negative Sampling). Перший варіант краще працює з не дуже частотними словами, а другий(семплування) показує кращі результати при частотних словах та векторах невеликої розмірності.

Очевидним є той факт, що для досягнення моделю гарної якості, її необхідно навчати на дуже великих корпусах даних.

Існують дві основні архітектури реалізації Word2Vec: Continuous Bag of Words (CBOW) та Skip-gram [12].

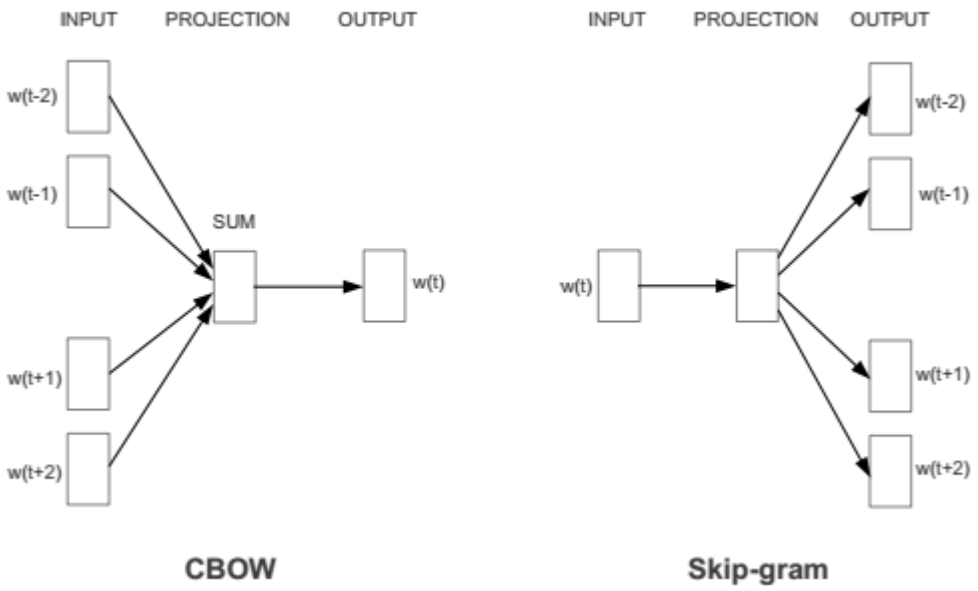

Рис. 3 Види архітектур реалізації Word2Vec[12]

CBOW та Skip-gram – це нейромережеві архітектури, що визначають як саме модель “навчається” на вхідних даних та як вона “запам’ятовує” інформацію про слова. Вони мають кардинально різні підходи [12]:

- CBOW – намагається передбачити слова при заданому контексті. Цей метод уявляє собою звичайну модель мішка слів з урахуванням чотирьох сусідніх термінів (двох перед та двох за).
- Skip-gram – навпаки передбачає контекст для заданого слова. Ця модель, як зрозуміло з назви, аналізує послідовність довжиною n грам,

де елементи розташовані на k один від одного. Зазвичай називають k-skip-n-gram. Наприклад для речення “Я люблю їсти смажену картоплю” – 1skip-2gram буде виглядати як набір біграм (“Я люблю”, “люблю їсти”, “їсти смажену”, “смажену картоплю”). Тоді пропускаємо k(1, в нашому прикладі) токенів, а з слів, які залишаються зліва та справа від пропущеного – формуємо біграми. Отримуємо: (“Я їсти”, “люблю смажену”, “їсти картоплю”).

2.2. Приклади практичного застосування

Обробка текстів природньої мови широко використовується в багатьох застосунках, де на вхід отримується неструктурований текст, в тому чи іншому форматі. Деякі з них використовують лише окремі етапи, описанні раніше, але вони є необхідними для коректної роботи програм. Наведемо приклади найпопулярніших з таких задач.

2.2.1. Машинний переклад

В машинному перекладі головним завданням є переклад тексту з однієї мови на іншу, цільову мову. Можна виділити декілька складних аспектів для цієї задачі [24]:

- Існує велика різноманітність мов, алфавітів та граматик
- Перекладання цілого речення є набагато складнішою задачею, аніж одного конкретного слова
- Можуть виникнути ситуації, де немає коректного перекладу, або є декілька варіантів.

Можна виділити три основні підходи:

1) Машинний переклад на основі правил – вимагає знань експертів про мови з якої та на яку перекладають, а також семантичних та морфологічних правил для цього. Цей метод уявляє собою

послідовність завдань з обробки природньої мови, включаючи токенізацію, визначення частин мови та нормалізацію. Для цього методу необхідно мати гарні словники для кожної з мов, а також правила прописуються вручну, що звичайно дає можливість легко розширювати систему, але з збільшенням кількості правил стає складніше її підтримувати.[24]

2) Статичний машинний переклад – як зрозуміло з назви, цей підхід використовує статистичні моделі, засновані на аналізі двомовних текстових корпусів. Основна ідея полягає у тому, що для кожної пари речень можна поставити у відповідність коефіцієнт вірогідність перекладу між ними. Тоді, враховуючи, що значення такої вірогідності буде дуже малим для некоректних випадків, ми обираємо найбільш вірогідну пару для конкретного речення за теоремою Байєса та отримуємо переклад.[14]
3) Нейронний машинний переклад – підхід, що використовує нейронні мережі для отримання машинного перекладу. Основна ідея полягає у прогнозуванні ймовірності послідовності слів, зазвичай за допомогою моделювання цілих речень в одній інтегрованій моделі. Цей метод машинного перекладу потребує менше пам’яті ніж статистичний, вони показують кращі результати в випадках зміни порядку слів в реченнях, але все ж таки в них можуть виникати проблеми у випадках коли навчальні дані є незбалансованими, та деякі слова зустрічаються лише декілька разів.[13]

2.2.2. Узагальнення тексту

Узагальнення тексту (Text summarization) – завдання стиснення фрагменту тексту до меншої версії, тобто зменшення розміру тексту і при цьому збережені ключових елементів, що несуть найбільше сенсу та загалом значення тексту. Автоматизація цього завдання набуває все більшої популярності в останній час,

оскільки ручне узагальнення тексту це дуже трудомісткий процес, що вимагає багато часу[25].

Існує багато різних завдань де використовується узагальнення текстів. До їх числа належать: класифікація текстів, відповіді на запитання, узагальнення різних типі документів, новин, генерація заголовків, тощо. Також генерація короткого змісту тексту, може бути інтегрована в системи та використана як проміжний етап, з метою зменшення обсягу документів.

Автоматичне узагальнення тексту – це завдання створення короткого змісту тексту, зберігаючи сенс оригінального документу, без будь-якої допомоги людини. Це дуже складана та нетривіальна здача, оскільки людина зазвичай перед створенням резюме для тексту, повністю читає його для розуміння його суті, а вже потім описує короткий зміст основних ідей [15].

Для цього завдання були запропоновані різні моделі, засновані на машинному навчанні. Загалом існує два різні підходи до автоматичного узагальнення текстів: екстрактивний та абстрактний [25].

Екстрактивний – основна ідея полягає у підбиранні речень безпосередньо з самого документу, створюючи цілісний короткий зміст. Цей метод намагається виявити важливі секції тексту, які потім вирізаються та доповнюються іншими порціями контенту, з метою отримати стиснуту версію. Таким чином вони залежать лише від вилучення речень з оригінального документу [25].

Абстрактний – спрямовані на створення резюме шляхом інтерпретації тексту за допомогою використання потужних методів природньої мови. Основною метою є створення коротшого тексту, що передає основний зміст, частина речень якого можуть не зустрічатися в оригінальному документі. Цей підхід створює резюме шляхом перефразування та включання частини інформації з початкового текст, який буде схожий на написаний людиною. Таким чином цей метод не обмежується простим підбором та перестановкою уривків [15].

2.2.3. Голосові помічники

Голосовий помічник (voice assistant) – це боти, що працюють на основі штучного інтелекту, розпізнавання голосу та обробки природньої мови щоб мати змогу відповідати на запитання та вести розмову з користувачем. Головна відмінність від текстових інтерфейсів, що вимагають від машин обробки тексту, його аналізу та складання відповіді, голосові помічники роблять все це на слух. Тобто вам не потрібно вводити своє запитання, а достатньо лише запитати його в голос. Ця технологія є досить складною і порівняно з текстовими інтерфейсами – новою.

Можна виділити такі основні кроки, необхідні голосовому помічникові, щоб отримати бажаний нам результат[16]:

1) Більшість з ботів використовують пасивне прослуховування. Це означає, що асистент постійно стежить за оточенням на наявність слів-ініціаторів. Як тільки слово-тригер буде вимовлено з достатньою, для бота, гучністю, він почне слухати запит користувача. В голосових асистентах типу Siri або Google Assistant, користувач має можливість вимкнути цей функціонал, віддаючи перевагу активації бота натисканням певної клавіши з метою збереження конфіденційності.
2) Наступним кроком є розпізнавання голосу. Цей етап починається як тільки бота було активовано. За допомогою підмножині штучного інтелекту та глибинному навчанню, відбувається перетворення звукових хвиль на структуровані, зрозумілі для обробки машиною, дані. Під час цього процесу враховується все, починаючи від тону, висоти, гучності та точності мови.
3) Потім за допомогою методів обробки природньої мови відбувається опрацювання складніших нюансів людської мови. Сюди належать такі речі, як контекст, сленг, акценти, тон користувача або інші неформальні аспекти. Це допомагає машині зрозуміти сенс питання, що вона почула.

4) Після обробки запиту, за допомогою розпізнавання голосу та NLP, голосовому помічникові потрібно отримати інформацію, пов’язану з запитанням. Це відбувається за допомогою пошуку інформації в певній базі знань, глибина якої залежить від помічника.
5) Завершальний етап – виведення відповідної інформації користувачу.

Розділ 3. Опис практичної частини

3.1. Аналіз технічного завдання

В межах даної роботи завданням було реалізація етапів попередньої обробки неструктурованих тексту для української мови. Необхідно було реалізувати наступний функціонал:

- Зчитування тексту

На вхід ми отримаємо шлях до файлу збереженого на комп'ютері, та отримуємо стрічку з текстом з файлу. Має бути реалізовано підтримка файлів з форматами PDF та txt.

- Токенізація

Реалізація розділення речення на необхідні токени. На вхід приходить стрічка з текстом – на вихід ми отримуємо масив токенів. Необхідним є реалізація токенізації на речення та на слова.

- Видалення стоп-слів

На вхід отримується масив токенів. Потрібно реалізація видалення стоп-слів, а також відкидання цифр, за потреби.

- Стемінг

Реалізація методу для отримання стем слова. За для зручності використання необхідно створити методи для масиву токенів та для одного конкретного токену. Адаптувати під потреби підручників з задачами по геометрії.

- Векторизація

Створення векторного уявлення. Реалізація методів мішку слів, зокрема бінарного та частотного, також створення TF-IDF представлення для колекції документів.

3.2. Обґрунтування алгоритму й структури програми

Для реалізації описаних в технічному завданні етапів, було створено наступну структуру програми (див рисунок 4.а). На етапі токенізації було використано спосіб розбивання тексту за допомогою регулярних виразів. Для реалізації алгоритму стемінгу, було обрано за основу алгоритм Портера, та його реалізацію під російську мову. Після аналізу відмінностей між мовами в суфіксах та закінченнях для різних частин мови, був реалізований наступний алгоритм: видаляється знак апострофа, при наявності постфікси, потім відбувається пошук та видалення закінчень різних частин мов в наступному порядку: дієприслівник, прикметник, дієприкметник(якщо це прикметник), дієслово, іменник. Видалення подвоєнь та м’який знак наприкінці(див. рисунок 4.б).

Слід зауважити, що вибір суфіксів для слів було адаптовано під потребу роботи програми з підручниками з геометрії. Були прибрані певні суфікси та додані інші з метою оптимізації результатів саме для таких документів.

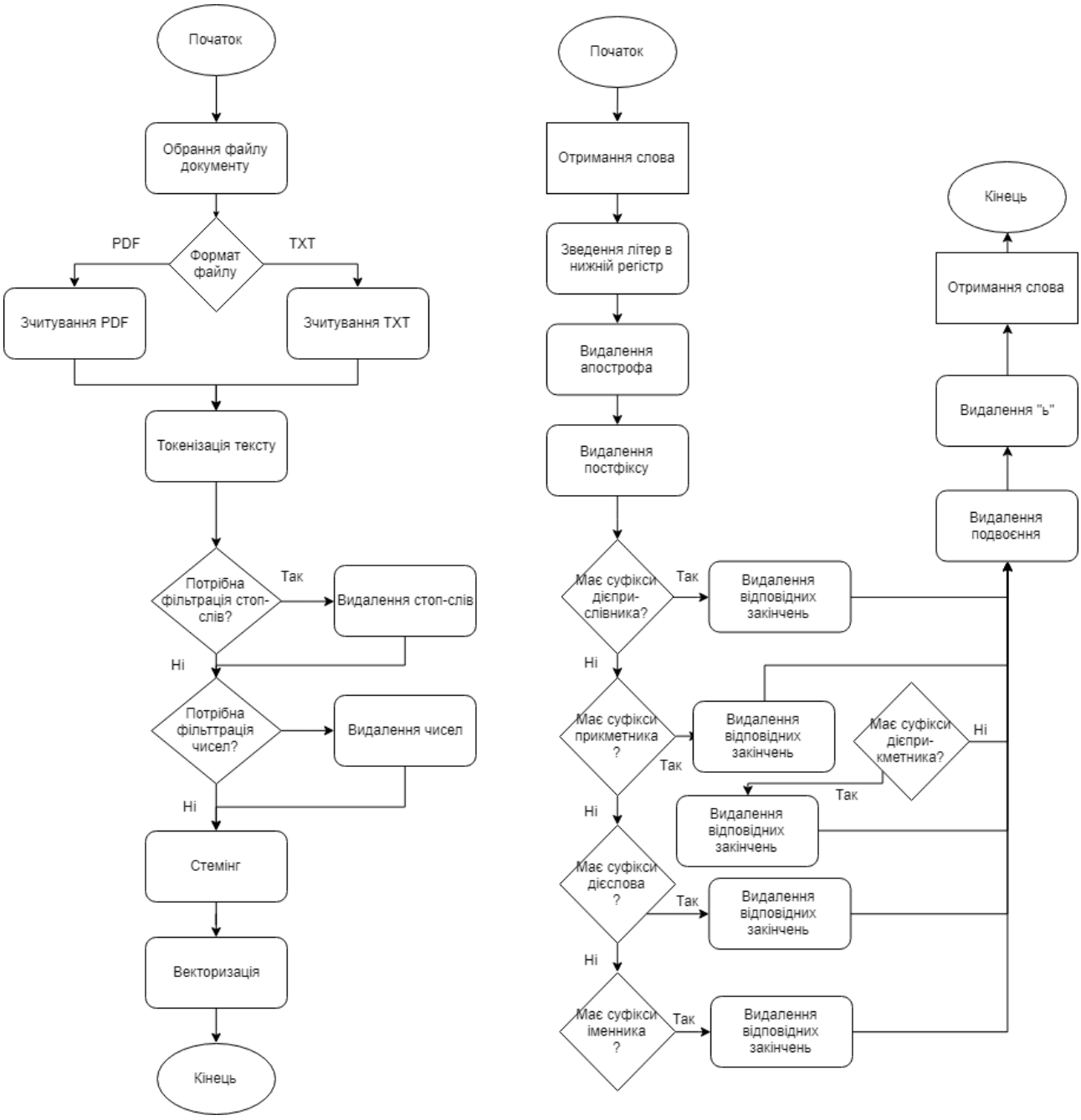

Рисунок 4. (а – структура програми, б – схема алгоритму стемінгу для української мови)

3.3. Опис розробки програми

При розробці даної програми використовувалась мова програмування Java версії 8. Для реалізації було створено чотири інтерфейси – FileReader, Stemmer, Tokenizer, Vectorizer, що зберігають інформацію про методи, та забезпечують

модульність та слабкої зв’язності. Та відповідні реалізації – FileReaderImpl, UkrainianStemmer, UkrainianTokenizer, VectorizerImpl та Main.(див рисунок 5)

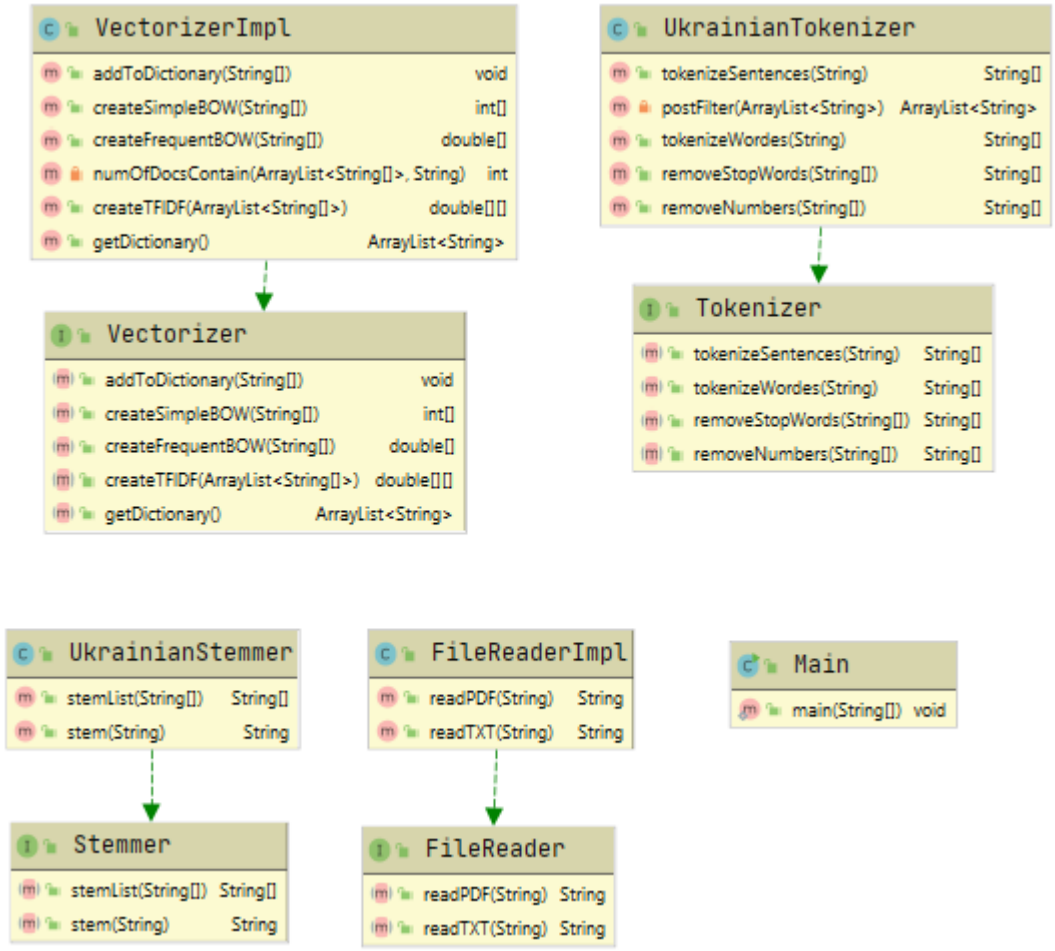

Рисунок 5 UML – діаграма класів застосунку

В класі FileReader реалізовано два метода readPDF та readTXT, що реалізують зчитування PDF та txt формати відповідно та повертають String, який зберігає контент документу.

```java
public class FileReaderImpl implements interfaces.FileReader {

    @Override
    public String readPDF(String fileLocation) throws IOException {
        PDDocument document = PDDocument.load(new File(fileLocation));
        String text= "";
        if (!document.isEncrypted()) {
            PDFTextStripper stripper = new PDFTextStripper();
            text=stripper.getText(document);
            document.close();
        }
        return text;
    }

    @Override
    public String readTXT(String fileLocation) throws IOException {
        File file = new File(fileLocation);
        BufferedReader br = new BufferedReader(new FileReader(file));
        StringBuilder text = new StringBuilder();
        String s;
        while ((s = br.readLine()) != null)
            text.append(s);
        return text.toString();
    }

}
```

Клас UkrainianTokenizer реалізує методи токенізації на слова та на речення, а також два окремі методи для видалення стоп слів та чисел з масиву токенів. Для токенізації слів також реалізований додатковий приватний метод postfilter, що допомагає прибрати пусті токени, а також реалізувати функціонал переносу слова.

```java
    public String[] tokenizeWordes(String text){
        ArrayList<String> arrayList = new ArrayList<>(Arrays.asList(text.split( regex: "[^A-Za-zА-Яа-яІіЇїҐґЄє’-]+")));
        postFilter(arrayList);
        return arrayList.toArray(new String[0]);
    }
    @Override
    public String[] tokenizeSentences(String text) {
        text = " " + text + "  ";
        text = text.replaceAll( regex: "[.]+.", replacement: ".");
        text = text.replaceAll( regex: "\n", replacement: " ");
        text = text.replaceAll( regex: "\r", replacement: " ");
        text = text.replaceAll(prefixes, replacement: "$0<prd>");
        text = text.replaceAll(number, replacement: "$0<prd>");
        text = text.replaceAll(websites, replacement: "<prd>$0");
        text = text.replaceAll( regex: "\\.”", replacement: "”.");
        text = text.replaceAll( regex: "\\.\"", replacement: "\".");
        text = text.replaceAll( regex: "!\"", replacement: "\"!");
        text = text.replaceAll( regex: "\\?\"", replacement: "\"?");
        text = text.replaceAll( regex: "\\.(?!<prd>)", replacement: ".<stop>");
        text = text.replaceAll( regex: "\\?", replacement: "?<stop>");
        text = text.replaceAll( regex: "!", replacement: "!<stop>");
        text = text.replaceAll( regex: "<prd>", replacement: "");
        ArrayList<String> arrayList = new ArrayList<>(Arrays.asList(text.split( regex: "<stop>")));
        postFilter(arrayList);
        return arrayList.toArray(new String[0]);
    }
    @Override
    public String[] removeStopWords(String[] arr) {
        return Stream.of(arr).filter(str -> !stopWords.contains(str))
                .collect(Collectors.toSet()).toArray(new String[0]);
    }
    @Override
    public String[] removeNumbers(String[] arr) {
        return Stream.of(arr).filter(str -> !isNumeric(str))
                .collect(Collectors.toSet()).toArray(new String[0]);
    }
}
```

Клас UkrainianStemmer зберігає реалізацію методу стемінгу Портера для української мови. Метод stem приймає на вхід токен, виконує та видалення регулярними виразами відповідно до прописаних для кожної з частин мови суфіксів. Метод stemList виконує стемінг для кожного з списку заданих токенів

```
public String[] stemList(String[] arr) {
    String[] copy = new String[arr.length];
    for (int i = 0; i < arr.length; i++) {
        copy[i] = stem(arr[i]);
    }
    return copy;
}
public String stem(String stem){
    stem = stem.toLowerCase();
    stem = stem.trim();
    stem = stem.replaceAll( regex: "'", replacement: "");
    //Постфікс
    String postfix = "(сять|ість|ся|іст)$";
    stem = stem.replaceAll(postfix, replacement: "");
    String perfectiveground = "(ив|ившись|ивши)$";
    String all = "[А-Za-zА-Яа-яІіЇїҐґЄє]*";
    String noun = "(ом|о|у|ах|иях|ями|ами|еи|и|ей|ой|ий|й|иям|ям|ию|ью|ия|ья|я|і|ові|ї|ею|єю|ою|є|а|ев|ов|е|ам|еві|ем|єм|ів|їв|ю|ием|ях|ы|ь)$";
    String verb = "(ять|у|вши|яти|є|ю|ав|сь|ся|ши|е|ме|ати|си|ив|ать|али|учи|ячи)$";
    String adj = "(а|е|ів|є|ій|у|ю|ова|ове|ою|ею|йми|ого|єє|ее|я|ем|им|ім|іми|ому|ими|ій|ий|их|іх|ої)$";
    //Дієприслівник
    if (stem.matches( regex: all + perfectiveground)){
        stem = stem.replaceAll(perfectiveground, replacement: "");
    }//Прикметник
    else if (stem.matches( regex: all + adj)){
        stem = stem.replaceAll(adj, replacement: "");
        // Дієприкметник
        String participle = "(ий|ою|і|их|ій|йми|ого|им|ім|а|у|ому)$";
        if (stem.matches( regex: all + participle)){
            stem = stem.replaceAll(participle, replacement: "");
        }
    }//Дієслово
    else if (stem.matches( regex: all + verb)){
        stem = stem.replaceAll(verb, replacement: "");
    }//Іменник
    else if (stem.matches( regex: all + noun)){
        stem = stem.replaceAll(noun, replacement: "");
    }
    stem = stem.replaceAll( regex: "и$", replacement: "");
    stem = stem.replaceAll( regex: "ь$", replacement: "");
    stem = stem.replaceAll( regex: "нн$", replacement: "н");
    return stem;
}
```

Клас VectorizerImpl реалізовує методи створення бінарного мішка слів за допомогою createSimpleBow, частотного бінарного мішка – createFrequentBOW для конкретного документу на основі словника, та createTFIDF для колекції документів. Словник зберігається в вигляді списку, та доповнюється списками токенів, за допомогою допоміжної функції addToDictionary. Слід зауважити, що оскільки tf-idf підраховується для всієї колекції, то і повертає значення для кожного слова кожного документа в колекції.

```java
public void addToDictionary(String[] tokens){
    Set<String> dict = new HashSet<>(dictionary);
    dictionary.clear();
    dict.addAll(Arrays.asList(tokens));
    dictionary.addAll(dict);
}
public int[] createSimpleBOW(String[] tokens){
    int[] BOWvector = new int[dictionary.size()];
    Arrays.fill(BOWvector, val: 0);
    for (String s:tokens) {
        BOWvector[dictionary.indexOf(s)]=1;
    }
    return BOWvector;
}
public double[] createFrequentBOW(String[] tokens){
    double size = tokens.length;
    double[] BOWvector = new double[dictionary.size()];
    Arrays.fill(BOWvector, val: 0);
    for (String s:tokens) {
        BOWvector[dictionary.indexOf(s)]+=(1.0/size);
    }
    return BOWvector;
}
public double[][] createTFIDF(ArrayList<String[]> collection){
    double[][] TFcollection=new double[collection.size()][];
    // Calculate term frequency for every term and document in collection
    for (int i = 0; i < collection.size(); i++) {
        TFcollection[i] = createFrequentBOW(collection.get(i));
    }
    // Calculate inverse document frequency for each term
    for (int i = 0; i < collection.size(); i++) {
        for (int j = 0; j < TFcollection[i].length; j++) {
            double IDF = Math.log(((double)collection.size())/(double)numOfDocsContain(collection,collection.get(i)[j]));
            TFcollection[i][j] = TFcollection[i][j] * IDF;
        }
    }
    return TFcollection;
}
```

3.4. Тестування програми і результати її виконання

Токенізація на речення було протестовано на прикладі уривку:

«Нехай про деяку лінію відомо лише те, що вона проходить через точки A і B. Для того щоб скласти уявлення про цю фігуру, такої інформації явно бракує.
Адже через точки A і B можна провести багато різних ліній (рис. 14).
Пряма ж задається цими точками однозначно. У цьому й полягає суть основної властивості прямої.»

Після токенізації зчитування з файлу та токенізації на речення отримуємо результат: [“Нехай про деяку лінію відомо лише те, що вона проходить через точки A і B.”, “Для того щоб скласти уявлення про цюфігуру, такої інформації явно бракує.”, “Адже через точки A і B можна провести багато різних ліній (рис.14).”, “Пряма ж задається цими точками однозначно.”, “У цьому й полягає суть основної властивості прямої.”].

На даному прикладі можна побачити коректність виконання токенізації для тексту, а особливо застосування винятку для “(рис.14)” – не розділяється на 2 різні речення.

Для тестування токенізації на слова та підрахунків векторів для документів були використанні приклади з таблиці 2.1, з метою контролю коректності підрахунків.

Було створено 4 документа, які були зчитані за допомогою readtxt, та розбиті на токени за допомогою методу tokenizeWordes, отримані токени були додані до словника, та відповідні мішки слів біли створенні, що відповідають до значень в табл 2.2:

```
Документ 1:Андрій любить їсти рибу, а Юля любить їсти яблука.
Документ 2:Андрій не хотів би їсти яблука.
Документ 3:Юля любить рибу.
Документ 4:Аня любить огірки.
Word simple BOW  а   test1: 1  test2: 0  test3: 0  test4: 0
Word simple BOW  Юля   test1: 1  test2: 0  test3: 1  test4: 0
Word simple BOW  рибу   test1: 1  test2: 0  test3: 1  test4: 0
Word simple BOW  любить   test1: 1  test2: 0  test3: 1  test4: 1
Word simple BOW  би   test1: 0  test2: 1  test3: 0  test4: 0
Word simple BOW  не   test1: 0  test2: 1  test3: 0  test4: 0
Word simple BOW  їсти   test1: 1  test2: 1  test3: 0  test4: 0
Word simple BOW  огірки   test1: 0  test2: 0  test3: 0  test4: 1
Word simple BOW  хотів   test1: 0  test2: 1  test3: 0  test4: 0
Word simple BOW  Аня   test1: 0  test2: 0  test3: 0  test4: 1
Word simple BOW  яблука   test1: 1  test2: 1  test3: 0  test4: 0
Word simple BOW  Андрій   test1: 1  test2: 1  test3: 0  test4: 0
```

При формувані частотного мішка слів для тих самих документів, отримаємо результати відповідні до табл 2.3

```
Документ 1:Андрій любить їсти рибу, а Юля любить їсти яблука.
Документ 2:Андрій не хотів би їсти яблука.
Документ 3:Юля любить рибу.
Документ 4:Аня любить огірки.
Word simple FREQ  а   test1: 0.11111  test2: 0  test3: 0  test4: 0
Word simple FREQ  Юля   test1: 0.11111  test2: 0  test3: 0.33333  test4: 0
Word simple FREQ  рибу   test1: 0.11111  test2: 0  test3: 0.33333  test4: 0
Word simple FREQ  любить   test1: 0.22222  test2: 0  test3: 0.33333  test4: 0.33333
Word simple FREQ  би   test1: 0  test2: 0.16667  test3: 0  test4: 0
Word simple FREQ  не   test1: 0  test2: 0.16667  test3: 0  test4: 0
Word simple FREQ  їсти   test1: 0.22222  test2: 0.16667  test3: 0  test4: 0
Word simple FREQ  огірки   test1: 0  test2: 0  test3: 0  test4: 0.33333
Word simple FREQ  хотів   test1: 0  test2: 0.16667  test3: 0  test4: 0
Word simple FREQ  Аня   test1: 0  test2: 0  test3: 0  test4: 0.33333
Word simple FREQ  яблука   test1: 0.11111  test2: 0.16667  test3: 0  test4: 0
Word simple FREQ  Андрій   test1: 0.11111  test2: 0.16667  test3: 0  test4: 0
```

Після обрахунку Tf-Idf для створенної з цих документів колекції, були отримані наступні результати:

```
Документ 1:Андрій любить їсти рибу, а Юля любить їсти яблука.
Документ 2:Андрій не хотів би їсти яблука.
Документ 3:Юля любить рибу.
Документ 4:Аня любить огірки.
Word  TFIDF а   test1: 0.15403  test2: 0  test3: 0  test4: 0
Word  TFIDF Юля   test1: 0.07702  test2: 0  test3: 0.23105  test4: 0
Word  TFIDF рибу   test1: 0.07702  test2: 0  test3: 0.23105  test4: 0
Word  TFIDF любить   test1: 0.06393  test2: 0  test3: 0.09589  test4: 0.09589
Word  TFIDF би   test1: 0  test2: 0.23105  test3: 0  test4: 0
Word  TFIDF не   test1: 0  test2: 0.23105  test3: 0  test4: 0
Word  TFIDF їсти   test1: 0.15403  test2: 0.11552  test3: 0  test4: 0
Word  TFIDF огірки   test1: 0  test2: 0  test3: 0  test4: 0.4621
Word  TFIDF хотів   test1: 0  test2: 0.23105  test3: 0  test4: 0
Word  TFIDF Аня   test1: 0  test2: 0  test3: 0  test4: 0.4621
Word  TFIDF яблука   test1: 0.07702  test2: 0.11552  test3: 0  test4: 0
Word  TFIDF Андрій   test1: 0.07702  test2: 0.11552  test3: 0  test4: 0
```

Якість стемінгу можна побачити на прикладах:

Початкове слово	Результат стемінгу
Предметний	предметн
тіл	тіл
зрізаної	зрізан
обчислення	обчислен
дотикається	дотикає т
многогранного	многогран
прямокутний	прямокутн
параграфі	параграф

Результати стемінгу було протестовано на колекції з 2 документів: Підручник з геометрії для 7 та 11 класів[17][18], які були подані в PDF форматі. В підручнику для 7 класу після токенізації за словами виявлено 39264 токени, в той час як в підручнику для 11 класу – 42217. Після стемінгу та видалення однакових стем для першого документу(7 класу) залишилось – 2296 унікальних токенів, для другого (11 класу) залишилося 1812. Словник для всієї колекції склав - 3141. Після аналізу отриманих стем власноруч, та за допомогою перевірки отриманих частотних коефіцієнтів для наведених документів, можна зробити висновок, що система показує досить якісні результати, та отримані вектори можна використовувати в подальшому аналізі текстів.

Висновки

Під час виконання даної роботи були детально проаналізовані етапи попередньої обробки неструктурованих текстів написаних природньою мовою. Найважливішими можна виділити два основні: токенізація та нормалізація.

Були досліджені такі варіанти токенізації як: токенізація за реченнями та за словами. А також вивчене питання видалення стоп-слів. Після аналізу можна зробити висновок, що цей етап є дуже важливим для подальшого аналізу даних, оскільки саме цей етап є першим в аналізі та саме від нього залежить якість отриманих токенів.

Були розглянути різні варіанти алгоритмів нормалізації – лематизація та стемінг. Проаналізовані відмінності їх реалізації та використання. Як підсумок лематизація дає більш точні результати, але вимагає більше часу, що часто є недопустимим у системах, що використовують NLP, тому часто віддають перевагу саме стемінгу.

Для лематизації були розглянуті необхідні для якісного результату етапи реалізації, та можливі підходи для цих етапів.

Для стемінгу були досліджені більш детально варіанти алгоритмів, основні принципи роботи кожного з них, а також переваги та недоліки. Після аналізу існуючих підходів до реалізації даного методу, можна зробити висновок, що алгоритм Портера є оптимальним, якщо розглядати з якості результату, складності імплементації та необхідних машинних ресурсів. Також були розглянуті основні недоліки алгоритмів стемінгу: надстемінг і недостемінг.

Для етапу векторизації було проаналізовані наступні підходи:

- Мішок слів
- TF-IDF
- N – грами
- Word2Vec

Кожний з цих методів має свої переваги, мішок слів – дуже простий в реалізації, TF-IDF допомагає краще розраховувати показник частотності для колекцій документів, N – грами, допомагає зберегти більше інформації про контекст слова для подальшого аналізу, а Word2Vec дає можливість розуміти схожість між векторами слів.

Список використаної літератури

1. Manning C. An Introduction to Information Retrieval / C. Manning, P. Raghavan, H. Schütze., 2009. – 544 с. – (Cambridge UP).
2. Grefensette G. What is word, What is sentence? Problems of Tokenization / G. Grefensette, P. Tapanainem. // Grenoble Laborotory.
3. Жердева М. В. Стемминг и лемматизации в lucene.net / М. В. Жердева, В. М. Артошенко. // Лесной Вестник. – 2016. – №3. – С. 131–134.
4. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ТЕКСТОВЫХ ДАННЫХ НА БАЗЕ ПАКЕТА NLTK PYTHON / Н. Ф.Хайрова, О. Ж. Мамырбаев, С. В. Петрасова, К. Ж. Мухсина. – Харків: НТУ «ХПИ», 2020. – 134 с.
5. Plisson J. A Rule based Approach to Word Lemmatization / J. Plisson, N. Lavrac, D. Mladenic.
6. Hindi Part-of-Speech Tagging and Chunking : A Maximum EntropyApproach / A.Dalal, K. Nagaraj, U. Sawant, S. Shelke. – 2006.
7. Jivani A. A Comparative Study of Stemming Algorithms / Anjali Ganesh Jivani. // IJCTA. – 2011.
8. Willet P. The Porter stemming algorithm: Then and now / Willet. // University of Sheffield. – 2006. – С. 219–223
9. Jurafsky D. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition / D. Jurafsky, J. Martin., 2008. – 608 с.
10. Distributed Representations of Words and Phrases and their Compositionality / [T. Mikolov, I. Sutskever, K. Chen та ін.].
11. A Neural Probabilistic Language Model / Y.Bengio, R. Ducharme, P. Vincent, C. Jauvin. // Journal of Machine Learning Research. – 2003. – №3. – С. 1137–1155.

12. Efficient Estimation of Word Representations in Vector Space / T.Mikolov, K. Chen, G. Corrado, J. Dean
13. Wu Y. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation / Y. Wu, M. Schuster, Z. Chen.
14. A STATISTICAL APPROACH TO MACHINE TRANSLATION / [P. Brown, J. Cocke, S. Pietra та ін.].
15. Dineshnath G. Abstractive Text Summarization of Research Articles Based on Word Associations / G. Dineshnath, S. Saraswath. // Journal of Computational and Theoretical Nanoscience. – 2019. – №16. – С. 1508–1511
16. Kumar L. Desktop Voice Assistant Using Natural Language Processing (NLP) / Lalit Kumar. // International Journal for Modern Trends in Science and Technology. – 2020. – №6. – С. 332–335.
17. Мерзляк А. Г. Геометрія: Підручник для 7 класу загальноосвітніх навчальних закладів / А. Г. Мерзляк, В. Б. Полонський, М. С. Якір. – Харків: Гімназія, 2015.
18. Геометрія : проф. рівень : підруч. для 11 кл. закладів загальної середньої освіти / А. Г. Мерзляк, Д. А. Номіровський, В. Б. Полонський та ін. — Х. : Гімназія, 2019. — 204 с.
19. Stopword Lists [Електронний ресурс] – Режим доступу до ресурсу: https://www.ranks.nl/stopwords.
20. Comparative Study of Truncating and Statistical Stemming Algorithms / [S. Memon, K. Memon, F. Dehraj та ін.]. // International Journal of Advanced Computer Science and Applications. – 2020.
21. Singh A. Vectorization of Text Documents for Identifying Unifiable News Articles / A. Singh, M. Shashi. // International Journal of Advanced Computer Science and Applications. – 2019. – С. 305–310.
22. Grzegorczyk K. Vector representations of text data in deep learning : дис. докт. техн. наук / Grzegorczyk Karol – Краків, 2018.

23. Shahzad Q. Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents / Q. Shahzad, A. Ramsha. // International Journal of Computer Applications. – 2018. – №181. – С. 25–29.
24. Sreelekha S. Statistical Vs Rule Based Machine Translation; A Case Study on Indian Language Perspective / S. Sreelekha.
25. Review of automatic text summarization techniques & methods / [A. Widyassari, S. Rustad, G. Shidik та ін.]. // Journal of King Saud University. – 2020.