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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методики вилучення смислової інформації з тексту на основі методів NLP»

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

_____ Владислав МУЗИКА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61

Владислав МУЗИКА

Керівник: Наталія ТРІНТІНА
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання
_____
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

Музиці Владиславу Павловичу

1. Тема кваліфікаційної роботи: Розробка методики вилучення смислової інформації з тексту на основі методів NLP

керівник кваліфікаційної роботи: Наталія ТРІНТІНА к.т.н., доц., доцент кафедри ІТ,

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

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

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, методики вилучення інформації з тексту, фреймворки для роботи з NLP.

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

1.Аналіз підходів до обробки тексту в NLP.
2. Існуючі математичні моделі, методи та алгоритми.
3. Розробка та аналіз практичних результатів.

5. Перелік графічного матеріалу: *презентація*

1. Титульний слайд.
2. Мета, об’єкта та предмет дослідження.
3. Порівняльний аналіз існуючих NLP бібліотек.
4. Алгоритм роботи програми.
5. Практичний результат.
6. Висновки.
7. Публікації та апробація роботи.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання на магістерську роботу	26.10.23	
2	Огляд предметної області	27.10.23 - 31.10.23	
3	Аналіз наявної науково-технічної літератури	01.11.23 - 12.11.23	
4	Дослідження методик вилучення смислової інформації	13.11.23 - 16.11.23	
5	Аналіз існуючих алгоритмів, методів та фреймворків	17.11.23 - 23.11.23	
6	Побудова підходу для поєднання двох систем аналізу настроїв тестових даних	24.11.23 - 28.11.23	
7	Розробка програмного забезпечення для реалізації компонованої методики вилучення тональності тексту	29.11.23 - 10.12.23	
8	Аналіз та тестування результатів виконаної роботи	11.12.23 - 14.12.23	
9	Оформлення роботи: вступ, висновки, реферат	15.12.23 - 17.12.23	

Здобувач вищої освіти _____ Владислав МУЗИКА
(підпис)

Керівник
кваліфікаційної роботи _____ Наталія ТРІНТІНА
(підпис)

РЕФЕРАТ

Текстова частина дипломної роботи 70 с., 64 формули, 14 рис., 4 табл., 26 джерел.

Об’єкт дослідження – вилучення смислової інформації з тексту.

Предмет дослідження – методи вилучення смислової інформації з тексту.

Метою роботи є покращення ефективності вилучення смислової інформації з тексту за рахунок створення методики на основі методів NLP.

Методи дослідження – емпіричні та теоретичні методи: проведення аналізу існуючих досліджень і публікацій, складення діаграм, методи проектування та розробки програмного забезпечення, технології об’єктно-орієнтованого програмування, проектування та розробка прототипів, експериментальні висновки.

У роботі проведено аналіз сучасного стану галузі обробки природньої мови, розглянуто методики, що застосовуються при його розробці. Проведено аналіз існуючих алгоритмів, моделей та фреймворків, що застосовуються при вирішенні завдань NLP, зокрема попередньої обробки та вилученні смислової інформації з тексту. Проаналізовано наукові статті нових розробок в цьому напрямку.

Проаналізовано можливість об’єднання методик та моделей для класифікації тональності та настрою вхідного тексту для підвищення точності при виконанні задач визначення настроїв вхідного тексту.

Створено програмний застосунок, що реалізує підходи об’єднання моделей для підвищення точності в класифікації настрою текстів на реальних корпусах даних.

Проведено порівняльний аналіз отриманих результатів при застосуванні програмного застосунку для визначення найбільш оптимального підходу, що має найбільший вплив на точність при об’єднанні таких моделей.

КЛЮЧОВІ СЛОВА: ОБРОБКА ПРИРОДНЬОЇ МОВИ, МЕТОДИ NLP, МОДЕЛІ NLP, АЛГОРИТМИ NLP, СМИСЛОВА ІНФОРМАЦІЯ ТЕКСТУ.

ABSTRACT

Text part of the master's qualification work:

70 pages, 64 formulas, 14 pictures, 4 table. 26 sources

The purpose of the work is to improve the efficiency of extracting meaningful information from the text by creating a methodology based on NLP methods.

Object of research – the extraction of semantic information from the text.

Subject of research – methods of extracting semantic information from the text.

Research methods – empirical and theoretical methods: analysis of existing research and publications, diagramming, methods of software design and development, object-oriented programming technologies, design and development of prototypes, experimental conclusions.

Summary of the work: The paper analyzes the current state of the field of natural language processing, examines the methods used in its development. An analysis of existing algorithms, models and frameworks used in solving NLP tasks, in particular pre-processing and extraction of semantic information from the text, was carried out. Scientific articles of new developments in this direction were analyzed.

The possibility of combining methods and models for classifying the tonality and mood of the input text in order to increase the accuracy when performing sentiment analysis of the input text.

A software application has been created that implements the approaches of combining models to increase the accuracy in the classification of the mood of texts on real data corpora.

A comparative analysis of the results obtained when using the software application was carried out to determine the most optimal approach that has the greatest impact on the accuracy when combining such models.

KEYWORDS: NATURAL LANGUAGE PROCESSING, NLP METHODS, NLP MODELS, NLP ALGORITHMS, MEANINGFUL TEXT INFORMATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NLP (Natural Language Processing) – Обробка природньої мови.

MLE (maximum likelihood estimation) - Оцінка максимальної правдоподібності

TF-IDF (Term Frequency - Inverse Document Frequency) – Частота термінів - зворотна частота документа

POS (Part-of-Speech) – Частина мови в частиномовному аналізі.

HMM (Hidden Markov Model) – Прихована модель Маркова

UD (Universal Dependency) – Фреймовок для узгодженої анотації граматики у різних людських мовах.

NER (Named Entity Recognition) - Розпізнавання іменованих сутностей

CRF (Conditional Random Fields) – Модель умовних випадкових полів.

RNN (Recurrent Neural Network) - Рекурентна нейронна мережа

BERT (Bidirectional Encoder Representations from Transformers) - Двоспрямовані кодувальні представлення з трансформерів

МП - Машинний переклад

IR (Information retrieval) - Інформаційний пошук

BM25 (Best Matching) - Функція ранжирування для оцінки релевантності документів певному пошуковому запиту.

QA (Question answering) – Системи для генерації автоматичних відповідей в галузі пошуку інформації та NLP.

LR (Logistic Regression) - Логістична регресія.

SVM (Support Vector Machine) - Опорно векторна машина.

NLTK (Natural Language Toolkit) - Набір інструментів природної мови.

GPL (GNU General Public License) - Загальна публічна ліцензія на комп’ютерне ПЗ.

JVM (Java Virtual Machine) – Віртуально машина Java.

JRE (Java Runtime Environment) - Середовище виконання Java.

JDK (Java Development Kit) - Набір для розробки Java

IDE (Integrated Development Environment) - Інтегроване середовище розробки.

CSV (Comma-Separated Values) – Формат файлу, значення якого розділені комами, для збереження даних.

ВСТУП

Створення методики обробки природньої мови на базі методів або алгоритмів NLP є важливим і актуальним напрямком в сучасних комп'ютерних науках. Це обумовлено рядом ключових факторів, які підкреслюють значення цієї теми.

Перш за все, зростання обсягів неструктурованих текстових даних в Інтернеті та інших цифрових медіа вимагає розвитку більш ефективних методів їх аналізу та обробки. Природна мова є основним способом комунікації людей, тому здатність комп'ютерних систем ефективно обробляти, розуміти та генерувати природну мову є ключовою для багатьох застосувань, включаючи автоматичний переклад, системи рекомендацій, пошукові системи, інтерактивні асистенти тощо.

Сучасні методи NLP значною мірою базуються на використанні нейронних мереж та глибокого навчання, що дозволяє досягти вражаючих результатів в розумінні та генерації природної мови. Однак це також ставить нові виклики, зокрема у сфері обробки великих обсягів даних, забезпечення точності та нейтральності результатів, а також захисту приватності та безпеки даних.

Людська мова надзвичайно різноманітна та динамічна, і методики NLP повинні вміти адаптуватися до різних мов, діалектів, контекстів та стилів спілкування. Це вимагає глибокого розуміння лінгвістичних особливостей та соціокультурних контекстів, а також розвитку більш гнучких та розширюваних моделей.

Збільшення інтеграції NLP-технологій у різноманітні області, від освіти до охорони здоров'я та бізнесу, підкреслює їх важливість та актуальність. Розвиток ефективних методів NLP може принести значні переваги у вигляді підвищення ефективності роботи, забезпечення кращого доступу до інформації, покращення якості послуг та багато іншого.

Отже, створення та впровадження нових методик при обробці природньої мови є важливим напрямком, що відкриває нові можливості для розвитку інформаційних технологій та їх застосування у різноманітних сферах життєдіяльності людини.

1 АНАЛІЗ ПІДХОДІВ ДО ОБРОБКИ ТЕКСТУ В NLP

NLP має досить багато аспектів, починаючи з попередньої обробки вхідного тексту, які зводять текстові дані до певних стандартизованих шаблонів інформацію, що надходить в запиті. До основних відносять розбиття тексту на менші одиниці, токени, спрощуючи подальший аналіз. Використовуючи для розбиття регулярні вирази, що додатково допомагають очистити текст від зайвих елементів, що не несуть семантичної інформації. Найпоширеніший вид розбиття це на слова, рідше для збереження семантики вираз ділиться на фрази та вирази та рідко поділ відбувається по-символьно. Додаткова нормалізація зводить текст до нижнього регістру та зменшує слова до їхніх базових форм (леми або корені) для уніфікації[3].

Після первинної обробки текст складений з токенів проходить до наступних етапів обробки синтаксису, визначаючи структуру речень та його компонентів для зв’язування слів у синтаксичну структуру. Це є основою у подальшому семантичному аналізі, а саме розуміння значення тексту, визначення зв'язків між словами та висловлюваннями. Отриманий текст також перевіряється на наявність важливих елементів для розуміння контексту (імена, дати, місця, тощо).

Для обробки додатково можуть використовувати досить багато алгоритмів, мовних моделей та інших підходів, таких як переклад на іншу мову або визначення емоційного тону тексту для імітації почуттів при генерації відповідей, якщо мова йде про створення текстового контенту.

У дослідженнях часто зустрічаються стандартні, аспекти обробки NLP та різного роду алгоритми, такі як контрольовані алгоритми класифікації для вирішення завдань класифікації, також комбіновані або гібридні моделі і алгоритми.

Кожна наскрізна NLP-система має свій визначений фундаментальний набір алгоритмічних інструментів, які складають сучасну нейромовну модель, це є її основою. Починаючи з токенізації, попередньої обробки тексту та корисних

алгоритмів, таких як обчислення відстані редагування, продовжуючи з більш складними задачами класифікації, логістичної регресії, нейронних мереж та завершуючи мережами прямого зв'язку, мережами зворотного зв'язку та трансформаторами.

1.1 Регулярні вирази

Одним з перших пунктів стандартизації та нормалізації є використання регулярних виразів, мови для визначення текстових пошукових рядків для пошуку тексту. Ця практична мова використовується у кожній комп'ютерній мові, текстовому процесорі, та інструментах обробки тексту. Представляючи собою звичайний алгебраїчний запис для опису набору рядків.

Регулярні вирази особливо корисні для пошуку регулярні вирази особливо корисні для пошуку в текстах, коли у нас є шаблон для пошуку шаблон для пошуку і корпус текстів для пошуку. Функція пошуку за регулярними виразами буде шукати в корпусі, повертаючи всі тексти, які відповідають шаблону. Корпус може бути як окремим документом, так і колекцією. Пошук можна спроектувати так, щоб він повертав кожен збіг у рядку, якщо їх більше ніж один, або тільки перший збіг.

1.2 Текстова нормалізація

Перед майже будь-якою обробкою тексту природною мовою, текст повинен бути нормалізований. У процесі нормалізації зазвичай вирішуються принаймні три завдання:

1. Токенізація (сегментація) слів
2. Нормалізація форматів слів
3. Сегментування речень

1.2.1 Токенізація

Початковим етапом є наївна версія токенізації та нормалізації, яку виконується лише для англійської мови за допомогою UNIX команд. Набір цих команд включає заміну деяких символів, що можуть заважати подальшій обробці, сортування на алфавітом та виключення однакових слів, що допомагає сформувати статистику при підрахунку однакових слів. Але для роботи з більшістю мов, окрім англійської, використовуються більш складні алгоритми токенізації.

Складніший алгоритм токенізації повинен брати до уваги деякі символьні комбінації зі словами для представлення точних дат, вказування валюти та точності числових даних у тексті або скорочення слів з використанням апострофа. Цей підхід повинен конфігуруватися більш детально в залежності від мови. Також цей етап тісно пов’язаний з розпізнаванням іменованих об'єктів, завданням виявлення імен, дат та організацій. Один з найпоширеніших стандартів токенізації відомий як Penn Treebank, випущений консиліумом лінгвістичних даних, який використовується для розпізнавання іменованих об'єктів. Виконує функцію зберігання слів без вилучення додаткових символів, дефізів чи пунктуації, зберігаючи пробіли та словосполучення у початковому смисловому вигляді.

Токенізація слів повинна відбуватись за мінімальний час виконання, тому для його оптимізації і використовуються детерміновані алгоритми на основі регулярних виразів, скомпільованих у дуже ефективні кінцеві автомати. Основні операції у регулярних виразах включають конкатенацію символів, диз'юнкція символів ([], |, і .), лічильники (*, +, і {n,m}), прив’язки (^, $) та оператори пріоритету ((,)). Токенізація та нормалізація слів зазвичай виконується за допомогою каскадів простих підстановок регулярних виразів або скінченних автоматів.

Третій варіант токенізації тексту замість визначення кожного слова як токену, використовуються попередньо сформовані дані, щоб визначити, що є токеном. Це корисно при роботі з невідомими словами, що є важливою проблемою при обробці мови. Для вирішення проблеми невідомих слів, модемні токенізатори часто автоматично індукують набори токенів, які включають менші за розміром

токени, що називаються підсловами. Підслова можуть бути довільними підрядками, або вони можуть бути одиницями, що несуть значення, такими як морфеми. У сучасних схемах токенізації більшість токенів - це слова, але деякі токени - це морфеми, що вільно зустрічаються.

Більшість схем токенізації складаються з двох частин: токену, що навчається, і токену-сегментатора. Токену, що навчається бере необроблений навчальний корпус (іноді грубо попередньо розділений на слова, наприклад, пробіли) і генерує словник з набору токенів. Сегментатор токенів бере сире тестове речення і сегментує його на лексеми у словнику.

Широко використовуються три алгоритми: кодування байт-пар, уніграмне моделювання мови, WordPiece та додатково бібліотека SentencePiece, яка включає в себе реалізацію перших двох .

1.2.2 Нормалізація слів

Нормалізація слів – це завдання приведення слів/токенів до стандартного формату, вибір єдиної нормальної форми для слів з декількома формами. Така стандартизація може бути цінною, незважаючи на те, що інформація про правопис яка втрачається в процесі нормалізації.

Зведення всього вхідного тексту до нижнього регістру, що є досить корисним для узагальнення у багатьох задачах, таких як пошук інформації або розпізнавання мови. Для аналізу настроїв та інших завдань класифікації тексту, вилучення інформації та машинного перекладу, навпаки, регістр може бути дуже корисним, то згортання регістрів, як правило, не виконується. Це пов'язано з тим, що збереження різниці між, наприклад, між країною США та займенником us може переважити перевагу в узагальнення, яку дало б відмінювання для інших слів.

У багатьох ситуаціях обробки природної мови ми також хочемо, щоб дві морфологічно різні форми слова поводилися однаково. Так наприклад можуть грати роль закінчення деяких слів або форми відмінювання, що можуть грати роль при запитах пошуку.

Лематизація – визначення того, що два слова мають один і той самий корінь, незважаючи на їх поверхневі відмінності. Найскладніші методи лематизації передбачають повний морфологічний розбір слова. Можна виділити два великі класи морфем: основа – центральна морфема слова фема слова, що забезпечує основне значення, і афікси, що додають додаткові значення різного роду.

Алгоритми лематизації можуть бути складними, але основним та більш простим, але грубим методом, який в основному полягає у відсіканні кінцевих афіксів слів кінцевих афіксів є наївна версія морфологічного аналізу під назвою стеммінг. Алгоритм базується на серії правил перезапису, що виконуються послідовно: вихід кожного проходу подається на вхід наступного проходу. Декілька з цих правил для прикладу:

ATIONAL → ATE (наприклад, relational + relate)

ING → $\in$, якщо основа містить голосну (наприклад, motoring > motor)

SSES → SS (наприклад, grasses + grass)

Прості стеммери можуть бути корисними у випадках, коли нам потрібно зіставити різні варіанти однієї і тієї ж леми. Тим не менш, вони схильні до помилок як надмірного, так і недостатнього узагальнення, наприклад для помилок пропуску та комісії:

Organization → Organ, doing → doe, numerical → numerous, policy → police

European → Europe, analyzes → analysis, noisy → noise, sparsity → sparse

Сегментація речень – ще один важливий крок в обробці тексту. Найбільш корисними сигналами для сегментації тексту є розділові знаки, такі як крапки, питання та знаки оклику. Знаки питання та знаки оклику є відносно однозначними маркерами меж речень. Крапки, з іншого боку, є більш неоднозначні. Символ крапки "." є неоднозначним між маркером межі речення та маркером межі речення і маркером абревіатур. З цієї причини токенізацію речення і токенізацію слова можна розглядати разом.

Загалом, методи токенізації речень працюють, спочатку вирішуючи (на основі правил або на основі машинного навчання), чи є крапка частиною слова або маркером межі речення. Словник абревіатур може допомогти визначити, чи є

крапка частиною загальновживаної абревіатури. У Стенфордському CoreNLP, наприклад, розбиття речень відбувається на основі правил, адетермінований наслідок токенізації; речення закінчується, коли в кінці речення розділові знаки (., !, або ?) ще не згруповані з іншими символами в токен (наприклад, абревіатура або число), за бажанням супроводжується додатковими фінальними лапками або дужки.

1.2.3 Мінімальна відстань редагування

Значна частина обробки природної мови пов'язана з вимірюванням того, наскільки схожими є два рядки схожі між собою. Це особливо корисно під час виправлення орфографії користувач при помилковому вводі.

Схожість рядків може означати не лише помилку, але й вказувати на те, що ці рядки можуть бути кореферентними. Відстань редагування дає нам спосіб кількісно оцінити обидві ці інтуїції щодо схожості рядків. Більш формально, мінімальна відстань редагування між двома рядками визначається як мінімальна кількість операцій редагування (таких як вставка, видалення, заміна), необхідних для перетворення одного рядка в інший.

Відстань Левенштейна між двома послідовностями є найпростішим ваговим коефіцієнтом, де кожна з трьох операцій має вартість 1, припускає, що заміна літери на саму себе, наприклад, t на t, має нульову вартість. Відстань Левенштейна між наміром і виконанням дорівнює 5. Левенштейн також запропонував альтернативну версію своєї метрики, в якій кожна вставка або видалення має вартість 1, а заміни заборонені. (Це еквівалентно дозволу на заміну, але кожна заміна має вартість 2, оскільки будь-яка заміна може бути представлена однією вставкою і одним видаленням). У цій версії відстань Левенштейна між наміром і виконанням дорівнює 8.

1.3 N-грами

Одним з найпоширеніших інструментів в обробці мовних даних є n-грами та мовне моделювання. Мовні моделі дають змогу призначити ймовірність реченню

або іншій реченню або іншій послідовності слів, а також передбачити слово на основі попередніх слів. Цей інструмент допомагає знаходити та вирішувати проблеми пов’язані з орфографією та граматичними помилками, аналізуючи та відсікаючи найменш ймовірні варіанти слів при підборі. Також може значно впливати при створенні багатомовних систем методом перекладу, де отриманий текст аналізується на ймовірність підібраних при перекладі слів.

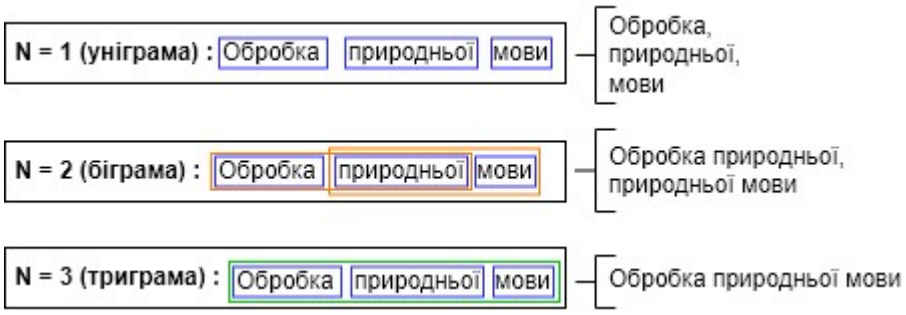


Рисунок 1.3.1 – Візуальне представлення N-грами

N-грами - це Марківські моделі, які оцінюють слова з фіксованого вікна попередніх слів. Ймовірності n-грами можуть бути оцінені шляхом підрахунку в корпусі та нормалізацією (оцінка максимальної правдоподібності).

Моделі n-грамної мови оцінюються ззовні в деякій задачі, або внутрішньо, за допомогою внутрішньо, використовуючи складність. Складність тестового набору відповідно до мовної моделі є середнім геометричним значення зворотної ймовірності тестового набору, обчисленої моделлю.

Ймовірність вираховується методом підрахунку відносної частоти зустрічі тих чи інших слів після аналізу великого об’єму текстової інформації, корпусу. Але це не є гарантією хорошої оцінки таких ймовірностей, тому що мова може змінюватись та є дуже гнучкою. Першим етапом при обчисленні ймовірності цілих послідовностей, таких як $P(w_1, w_2, \ldots, w_n)$, це розкласти цю ймовірність за допомогою ланцюгового правила обчислення ймовірностей:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \quad (1.1)$$

Такі розрахунки будуть досить трудомісткими, тому для цього впроваджують більш оптимальні формули, що визначають ймовірність використання тих чи інших слів у послідовності, Так для прикладу біграмна модель бере до уваги лише попереднє слово в такій послідовності та складає ймовірність його використання, це має назву припущення Маркова. Таким чином нам не потрібно аналізувати все речення в якому зустрічається бажане слово. Така модель може бути розширена до триграми, що аналізує вже два попередні слова або більше, складаючи n-граму, де кількість слів для аналізу є n -1 від бажаного.

Загальне рівняння для наближення n-грами до умовної ймовірності наступного слова в послідовності буде виглядати як:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}) , \tag{1.2}$$

де N – розмір n-грам, враховуючи біграмне припущення щодо ймовірності окремого слова, ми можемо обчислити ймовірність повної послідовності слів:

$$P(w_{1:n}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{1.3}$$

Оцінка цих біграмних або n-грамної ймовірності відбувається за допомогою інтуїтивного способу оцінки ймовірностей називається оцінкою максимальної правдоподібності MLE(maximum likelihood estimation). Ця оцінка отримується для параметрів n-грамної моделі, отримавши відділки з корпусу та нормалізуючи їх таким чином, щоб вони лежали між 0 та 1.

Наприклад, щоб обчислити певну ймовірність біграми слова w_n, заданого попереднім словом w_{n-1} , обчислюється лічильник біграми $C(w_{n-1}w_n)$ і пронормується його на сума всіх біграм, що мають однакове перше слово w_{n-1}, оскільки сума всіх лічильників біграм, які починаються з заданого слова w_{n-1}, повинна дорівнювати кількості унігram для цього слова w_{n-1}, отримуємо формулу:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{1.4}$$

Для прикладу можна взяти мінімальний корпус з декількох речень. Щоб обмежити речення та вказати на початкове та кінцеве слово в кожній послідовності

потрібно спочатку доповнити кожне речення спеціальним символом <s> на початку речення речення, щоб отримати біграмний контекст першого слова та спеціальний символ кінця </s>.

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

Розрахунки для деяких ймовірностей біграм з цього корпусу:

$P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}|\texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>}|\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}|\texttt{I}) = \frac{1}{3} = .33$

Рівняння для загального випадку оцінювання параметрів n-грами MLE:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}\ w_n)}{C(w_{n-N+1:n-1})} \tag{1.5}$$

Рівняння (1.5) оцінює ймовірність n-грами шляхом ділення спостережуваної частоти певної послідовності на спостережувану частоту префікса. Це відношення називається відносною частотою. В MLE результуючий набір параметрів максимізує ймовірність навчальної вибірки T для моделі M (тобто P(T|M)).

1.3.1 Оцінка мовних моделей

Для оцінки роботи побудованих моделей та n-грам, потрібні великі корпуси з текстовою інформацією, найкращим випробуванням таких систем вважається тестування їх у реальних зовнішніх застосунках, де масштаб даних є максимальним в умовах роботи, але такий підхід є надзвичайно дорогим та не дає наглядних метрик приросту в ефективності та точності нових моделей у порівнянні з іншими. Тому для тестування таких систем використовують стандартизовані набори тестових корпусів. Так при тестуванні двох систем можна виявити слабкі сторони кожної в порівнянні результатів на однакових даних, окрім того такі корпуси, як правило добре проаналізовані часом та показують сучасний стан та проблеми, що потрібно вирішити у першу чергу. Цей підхід скорочує витрати та зусилля на тестування нових моделей.

Для навчання таких моделей використовують інші набори даних, що не пов’язані з тестовими корпусами. Якщо при навчанні таких моделей тестові дані потрапляють у навчальну програму, то це може спричинити не точність у розумінні при використанні у зовнішньому середовищі. Метрики ймовірності при навчанні на таких корпусах можуть бути не реалістичними, а зазвичай завищеними, тому що бере до уваги структуру обмеженої кількості даних, що може негативно вплинути на реальну роботу такої моделі.

Різний об’єм даних по різному впливає на процес навчання, менший об’єм дозволяє не використовувати потужності на початкових етапах розробки моделей, що допомагаю на початкових етапах виявляти недоліки. В свою чергу більший об’єм потенційно збільшує початкову точність в навчанні, але займає більше часу та більш трудомістким. На практиці поділ при розробці такого роду систем поділяють відсотково на 80% основного витраченого часу на навчання з корпусами даних, та по 10% залишають на розробку і тестування. Хорошою практикою є не поєднувати тестові та навчальні корпуси та при недостатніх ключових даних в тестових, додавати неохоплені частини даних під час навчання.

1.3.1.1 Розгубленість

На практиці не використовують сиру ймовірність як метрику для оцінювання мовних модифікацій, є інший варіант, який називається розгубленість. Розгубленість мовної моделі на тестовому наборі - це обернена ймовірність тестового набору, нормалізована за кількістю слів.

Для тестового набору $W = (\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_N)^{-\frac{1}{N}}$:

$$\textbf{розгубленість}\,(W) \;=\; P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \;=\; \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \tag{1.6}$$

Для розширення ймовірності W використовують ланцюгове правило обчислення ймовірностей:

$$\textbf{розгубленість}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}} \tag{1.7}$$

Складність тестового набору W залежить від того, яку мовну модель ми використовуємо. Розгубленість W з одноградною мовною моделлю (просто середнє геометричне від ймовірностей уніграми):

$$\textbf{розгубленість}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}} \tag{1.8}$$

Розгубленість W, обчислена за допомогою біграмної мовної моделі, також є середнім геометричним, але тепер біграмних ймовірностей:

$$\textbf{розгубленість}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{1.9}$$

Потрібно розуміти, чим вища умовна ймовірність послідовності слів, тим менша розгубленість. Таким чином, мінімізація розгубленості еквівалентно максимізації ймовірності тестового набору відповідно до мовної моделі. Зазвичай під послідовністю слів ми розуміємо всю послідовність слів у деякому тестовому наборі. Оскільки ця послідовність буде перетинати багато речень, потрібно включити маркери початку і кінця речення <s> і </s> до обчислення ймовірності. Нам також потрібно включити маркер кінця речення </s> (але не маркер початку речення <s>) у загальну кількість токенів N.

1.3.2 Невідомі слова

Ще одна проблема, що трапляється дуже часто, невідомі слова в тексті при застосуванні мовних моделей. Такого роду проблема може виникнути в період їх тестування, коли тестовий набір не містить слів, що були присутні під час навчання, а також при зовнішньому застосуванні. Для навчання моделей використовуються

словники, які повинні містити якомога більшу кількість зрозумілих слів. Але це можливо лише, якщо тестовий корпус містить лише слова, що присутні у навчальному. Однак, в реальних застосунках такі ситуації несумнівно трапляться, тому потрібно брати їх до уваги та обробляти також. Відсоток невідомих слів у таких наборах тексту називають часткою невідомих слів.

Одним з варіантів обробки таких випадків є створення відкритої словникової системи, шляхом додавання таких слів з маркером невідомого <UNK>. Існує два поширених способи дізнатися ймовірність невідомого слова. Першим є модель <UNK> та полягає у перетворенні проблеми у задачу з закритим словником, де фіксований словник обирається заздалегідь. На кроці нормалізації тексту кожне слово, що не входить до навчальної множини замінюють на слово-токен <UNK>. Далі відбувається оцінка ймовірності його виникнення, як для звичайного слова у навчальній множині.

Альтернативою, у випадку відсутності попередньо створеного словника, є створення неявного словника, зі схожою заміну на слово-токен <UNK>, на основі їхньої частоти. Тобто тренування моделі відбувається так само, як і для слів, що містить словник, вираховуючи ймовірність виникнення такого слова. Вибір такого <UNK> слова також впливає на метрики заплутаності, при невеликому словнику та зміні частоти виникнення невідомого слова можна зменшити рівень заплутаності, Такі моделі можуть бути порівняні тільки з використанням однакових словників.

1.3.3 Алгоритми згладжування

Наступна проблема це випадки зі словами, які є в нашому словнику, але з'являються в тестовому наборі в невидимому контексті, наприклад, вони з'являються після слова, після якого вони ніколи не з'являлися в навчанні. Щоб мовна модель не приписувала нульову ймовірність цим невидимим подіям, потрібно відкинути частину ймовірнісної від деяких більш частих подій і віддати їх тим, яких ми ніколи не бачили. Ця модифікація називається згладжуванням або дисконтуванням.

Алгоритми згладжування надають більш досконалий спосіб оцінити ймовірність здатність n-грам. Найпоширеніші алгоритми згладжування для n-грам базуються на відліки n-грам нижчого порядку за допомогою відступу або інтерполяції. Як відставання, так і інтерполяція вимагають дисконтування для створення розподілу ймовірностей. Існують різні способи згладжування, а саме згладжування за Лапласом, згладжування з додаванням k, безглуздий відступ та згладжування Кнезера-Нея, що використовує ймовірність того, що слово є новим продовженням, поєднуючи дисконтовану ймовірність з ймовірністю продовження нижчого порядку.

Згладжування Лапласа

Найпростіший спосіб згладжування - додати одиницю до всіх n-грамових відліків, перш ніж до того, як ми нормалізуємо їх у ймовірності. Всі відділки, які раніше дорівнювали нулю, тепер матимуть значення 1, значення 1 - 2, і так далі. Цей алгоритм називається згладжуванням Лапласа. Згладжування Лапласа не є достатньо ефективним для використання у сучасних n-грамних моделях, але воно корисно вводить багато понять, які ми бачимо в інших алгоритмах згладжування, дає корисну базову лінію, а також є практичним алгоритмом згладжування для інших задач, таких як класифікація текстів.

Застосування згладжування Лапласа до уніграми ймовірностей, якщо незгладжена оцінка максимальної правдоподібності унігрaмної ймовірності слова w_y є його кількість c_i, нормована на загальну кількість лексем слова N:

$$P(w_i) = \frac{c_i}{N} \tag{1.10}$$

Згладжування Лапласа просто додає одиницю до кожного підрахунку (звідси його альтернативна назва - згладжування з додаванням одиниці). Оскільки в словнику є V слів і кожне з них було збільшено, також потрібно скоригувати знаменник, щоб врахувати додаткові V спостережень.

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V} \tag{1.11}$$

Замість того, щоб змінювати і чисельник, і знаменник, зручно описати, як алгоритм згладжування впливає на чисельник, визначивши скориговане число c^*. Цей скоригований підрахунок легше порівнювати безпосередньо з підрахунками MLE і можна перетворити на ймовірність, як і MLE, шляхом нормалізації на N. Для визначення цього числа, оскільки ми змінюємо лише чисельник на додаток до додавання одиниці, потрібно помножити на коефіцієнт нормалізації $\frac{N}{N+V}$:

$$c_i^* = (c_i + 1)\frac{N}{N + V} \tag{1.12}$$

Далі слідує перетворення c_i^* у ймовірність P_i^* шляхом нормалізації на N. Схожий спосіб для розгляду згладжування є дисконтування (зменшення) деяких ненульових для того, щоб отримати масу ймовірності, яка буде присвоєна нульовим відліком. Таким чином, замість того, щоб посилатися на дисконтовані відділки c^*, ми можемо описати алгоритм згладжування в термінах відносної знижки d_c, відношення дисконтованих відліків до початкових відліків:

$$d_c = \frac{c^*}{c} \tag{1.13}$$

Згладжування з додаванням k

Альтернативою згладжуванню Лапласа є перенесення трохи меншої частини маси ймовірностей з видимих подій на невидимі. Замість того, щоб додавати 1 до кожного підрахунку, ми додаємо дробовий підрахунок k(.5? .05? .01?). Тому цей алгоритм називається згладжуванням з додаванням k.

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \tag{1.14}$$

Згладжування з додаванням k вимагає, щоб ми мали метод вибору k, це можна зробити, наприклад, шляхом оптимізації на розробці. Хоча add-k є корисним для деяких завдань (зокрема класифікації текстів), виявляється, що він все ще не дуже добре працює для моделювання мови, генеруючи підрахунки з поганою дисперсією і часто недоречними знижками.

Великі мовні моделі та безглуздий відступ

Використовуючи текст з Інтернету або інших величезних колекцій, можна будувати надзвичайно великі мовні моделі. Корпус Web 1 Trillion 5-грамм, випущений Google, включає різні великі набори n-грамм, в тому числі від 1-грамм до 5-грамм з усіх послідовностей п'яти слів, які з'являються щонайменше в 40 різних книгах з 1,024,908,267,229 слів тексту з загальнодоступних веб-сторінок англійською мовою.

Google також випустив корпус Google Books Ngrams з n-грамами, взятими зі своїх книжкових колекцій, включаючи ще 800 мільярдів токенів n-грам з китайської, англійської, французької, німецької, івриту, італійської, російської та іспанської мов [21]. Менші, але більш ретельно куровані корпуси n-грам для англійської мови включають мільйон найчастотніших n-грам, взятих з COCA (Corpus of Contemporary American English) - корпусу американської англійської мови на мільярд слів [22]. COCA - це збалансований корпус, тобто він містить приблизно рівну кількість слів з різних жанрів: веб, газет, транскриптів розмовної мови, художньої літератури тощо, взятих за період 1990-2019 років, а також контекст кожної n-грами та контекст кожної n-грами, а також позначки жанру та походження.

Міркування ефективності є важливими при побудові мовних моделей, які використовують такі великі набори n-грам. Замість того, щоб зберігати кожне слово як рядок, його зазвичай представляють у пам'яті як 64-бітне хеш-число, а самі слова зберігають на диску.

Модель n-грамної мови можна також зменшити за допомогою відсікання, наприклад, зберігаючи лише ті n-грами, кількість яких перевищує певний поріг

(наприклад, поріг 40, використаний у версії Google n-gram), або використовуючи ентропію для відсікання менш важливих n-грам. Іншим варіантом є побудова наближених мовних моделей за допомогою таких методів, як фільтри Блума.

Ефективні інструменти для побудови мовних моделей, такі як KenLM, використовують відсортовані масиви, ефективно об'єднують ймовірності та відступи в одне значення, а також використовують сортування злиттям для ефективної побудови таблиць ймовірностей за мінімальну кількість проходів через великий корпус.

Хоча за допомогою цих інструментів можна будувати мовні моделі веб-масштабу за допомогою просунутих алгоритмів згладжування, таких як алгоритм Кнезера-Нея, показують, що для дуже великих мовних моделей набагато простіший алгоритм може бути достатнім. Цей алгоритм називається безглуздим відступом. Цей алгоритм відмовляється від ідеї намагатися зробити мовну модель істинним розподілом ймовірностей. Не відбувається дисконтування ймовірностей вищих порядків. Якщо графа вищого порядку має нульовий підрахунок, відбувається відступ до n-графа нижчого порядку, зваженого фіксованою (не залежною від контексту) вагою. Цей алгоритм не створює розподіл ймовірностей, його можна представити як:

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \textbf{якщо} \ \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \textbf{інакше} \end{cases} \tag{1.15}$$

1.4 Класифікація та категоризація тексту

Класифікація є основою інтелекту, як людського так і штучного. Можливість розпізнавати та вирізняти категорії речей, явищ та почуттів, значно полегшує розуміння навколишнього світу. Мовні NLP моделі також використовують такий підхід для структуризації власних набутих знань. Окрім основного завдання загальної категоризації тексту розглядається саме двійкова класифікація, вона є базовою та може поділяти отримані дані, наприклад на категорії настроїв.

Вилучення та аналіз настроїв вхідного тексту допомагає визначати емоційне забарвлення, тобто чи є воно позитивним або негативним. Це є досить корисною галуззю, що має комерційне застосування у маркетингу та політиці, виконуючи завдання аналізу настроїв користувачів таких систем.

Тренувальними корпусами для навчання такого роду моделей можуть виступати будь-якого роду огляди продукції, фільмів, книг та іншого, що виражають ставлення людини до того чи іншого аспекту. Відгуки в інтернеті, які містять фрази, що включають маркери настрою, наприклад: “чудово”, “гарно”, “жахливо” та інші, значно допомагають навчатися таким моделям.

Такі моделі здатні застосовувати задачі двійкової класифікації для виявлення спаму, щоб в подальшому відсіювати електроні листи на пошті до класу спаму або не спаму. Для такої класифікації можуть використовуватися багато різних лексичних та інших ознак. Так підозрілими для такої системи можна вважати листи що містять в собі фрази: “безкоштовно”, “шановний переможцю” та інші. Для наукової роботи, цифрових гуманітарних, соціальних наук, судової лінгвістики, визначення авторства тексту, присвоєння тексту бібліотечної предметної категорії чи позначок теми, допомагаючи при дослідницькій діяльності.

Класифікація тексту повинна включати в себе також розуміння якою мовою він написаний, для випадків комбінованого використання різних мов, тому що для кожної мови потрібно здійснювати окрему обробку. Це є одним з перших кроків у більшості конвеєрів обробки мови.

Мета класифікації полягає в тому, щоб взяти окреме спостереження, виділити в ньому деякі корисні ознаки і таким чином класифікувати це спостереження в один з наборів дискретних класів.

Більшість випадків класифікації в мовній обробці натомість виконується за допомогою керованого машинного навчання, де у контрольованому навчанні ми маємо набір даних вхідних спостережень, кожне з яких пов’язано з певним правильним результатом. Мета алгоритму – навчитися відображати нове спостереження на правильний результат. Формально, завдання контрольованої класифікації полягає в тому, щоб взяти вхідний параметр x і фіксований набір

вихідних класів $Y = \{y_1, y_2, \ldots, y_M\}$ і повертає прогнозований клас $y \in Y$. У підконтрольній ситуації ми маємо навчальний набір з N документів, кожен з яких вручну позначено класом: $\{(d_1, c_1), \ldots, (d_N, c_N)\}$, де c – клас, а d - документ

Метою є навчання класифікатору, здатного відображати новий документ d у його правильний клас $c \in C$, де C – деякий набір корисних класів документів.

Багато типів алгоритмів машинного навчання використовуються для створення класифікаторів. Прикладами таких способів класифікації є алгоритми наївного Байєсу та логістична регресія. Генеративні класифікатори, такі як наївний Байєс, будують модель того, як клас може генерувати деякі вхідні дані, проаналізувавши спостереження, повертаючи клас, який скоріш за все згенерував це спостереження. Дискримінаційні класифікатори, такі як логістична регресія в свою чергу які ознаки вхідних даних є найбільш корисними для розрізнення можливих класів, такі системи є більш точними.

1.5 Лексична семантика

Семантика, як наука ставить перед NLP досить багато завдань для їх вирішення. Так при обробці мови виникає низка проблем, таких як значення окремих слів, леми, що до них відносяться, синоніми в різних контекстах, подібність і спорідненість слів, фреймові структури та конотації.

Слова, які у словнику можуть бути визначені по різному, однакові за написанням але зовсім різні за значенням, слово “миша” для прикладу це або гризунів або пристрій для керування курсором. Форма миші є лемою, яка також називається формою цитування. Форма миша також буде лемою для слова миші; словники не мають окремих визначень для відмінюваних форм, як миші. У багатьох мовах форма інфінітива використовується як лема для дієслова. Як показує приклад, кожна лема може мати кілька значень, що може ускладнити тлумачення, наприклад при пошуку інформації про мишу, така система повинна вирішувати про що саме йде мова. Саме це є аспектом сенсу слова, що вирішує проблему багатозначності та розуміє значення зсилаючись на контекст.

Кожне слово має своє значення, але існує поняття синонімії, коли значення слова майже ідентичне до зовсім іншого. Тобто одне слово може замінити інше без суттєвих змін істинності речення, в такому випадку ці слова мають однакове пропозиційне значення. Проблема виникає при синонімах, що не досить тісно взаємо замінюють один одного, так для прикладу слово “вода” може бути замінено на H_2O. В цьому прикладі зберігається правдивість того, що обидва цих слова зберігають сенс, але не є повністю ідентичними за значенням. Це є принципом контрасту, який стверджує, що різниця у формі завжди пов’язана з різницею у значенні. Так H_2O має більше сенсу при його використанні у науковому контексті, ніж у путівнику для походів, тому на практиці синоніми використовуються для приблизного зв’язування.

Подібні слова, які не являються синонімами також створюють проблеми для NLP моделей. Знання того, наскільки схожі два слова допомагає підрахувати, наскільки значення двох фраз або речень схожі між собою, що є досить корисним компонентом при вирішенні семантичних завдань, таких як відповіді на питання, перефразування або резюмування.

Значення двох слів може бути пов’язане іншими способами, окрім подібності. Одним з таких класів зв’язків називається спорідненістю слів або асоціацією слів. Так для прикладу можна взяти слова “чашка” та “чай”. Ці слова практично не мають спільних рис, чай – напій, а чашка – предмет, вмісткість. Але вони пов’язані між собою асоціаціями, так як часто вживаються в одному контексті. Одним з видів спорідненості між словами є їх належність до одного семантичного поля, тобто набір слів, що охоплює певний семантичний домен і мають структуровані відносини між собою. Це може бути певне семантичне поле, наприклад море, яке включає в себе пов’язані асоціаціями слова, на кшталт риб, водоростей, устриць, тощо. Такі підходи до компонування слів у тематики є корисними при створенні тематичних моделей, що здатні виявляти тематичні структури в документах.

Тісно пов’язаними з семантичними полями є семантичні фрейми, набори слів, що позначають учасників певних заходів чи подій. Для прикладу можна взяти

комерційну операцію, слова в ній можуть бути закодовані лексично, тобто людина в такій операції може мати назву суб’єкт, при передачі товару або грошей іншій, так виконується послуга покупки чи продажу. Так формується фрейм, тобто група слів, що відносяться скоріш за все до сукупності виду операцій. Знання такого роду, допомагає зрозуміти системі, що речення типу “Олег купив книгу у Василя” є рівнозначним “ Василь продав книгу Олегу”, так як вони відіграють роль покупця та продавця у цьому фреймі. Здатність розпізнавати такі парафрази є важливою для може значно вплинути на машинний переклад.

Слова мають афективні значення або іншими словами канотації. Канотація виступає для позначення аспекту значення слова, пов’язаного з емоціями, настоями думками, тощо. Слова можуть мати позитивне або негативне забарвлення, де подібні слова можуть відрізнятися за значенням та відіграє велику роль при завданнях аналізу настроїв.

1.6 Векторна семантика

Векторна семантика – це стандартний спосіб представлення значення слова в NLP, що допомагає моделювати багато аспектів значення, що відносяться до лексичної семантики. Ідея векторної семантики полягає в тому, щоб представити слово як точку у багатовимірному семантичному просторі, який виходить з розподілу сусідніх слів. Вектори для представлення слів називаються вбудуваннями. Вбудуваннями у математичному значенні, як побудова одного простору чи структури в інше. Візуально це можна представити, як площину на якій розміщені слова, що складаються в групи по схожості за категоріями або пов’язані між собою.

Дрібнозерниста модель подібності слів векторної семантики є надзвичайно потужною для програм NLP. Розглянуті класифікатори безпосередньо залежать від такого набору даних при їх навчанні та тестуванні. Є дві найпоширеніші моделі, які використовують векторну семантику, а саме tf-idf та word2vec.

1.6.1 TF-IDF

TF-IDF (Term Frequency - Inverse Document Frequency) – частота термінів - зворотна частота документа, широко використовуваний статистичний метод у обробці природної мови та пошуку інформації. Він вимірює, наскільки важливим є термін у документі відносно корпусу. Слова в текстовому документі перетворюються на номери важливості за допомогою процесу векторизації тексту. TF-IDF оцінює слово шляхом множення термінової частоти слова на зворотну частоту документа.

Першим фактором є частота термінів, це те наскільки часто слово t зустрічається у документі d. При необробленій кількості такої частоти, можна вивести формулу:

$$\mathrm{tf}_{t,d} = \mathrm{count}(t,d) \tag{1.16}$$

Але частіше ця частота трохи стискається та використовується log_{10} від частоти. Інтуіція полягає в тому, що слово, яке з'являється в документі 100 разів, не робить це слово в 100 разів більш релевантним значенню документа. Оскільки не можна взяти логарифм 0, зазвичай до лічильника додають одиницю та формула для частоти термінів виглядає, як:

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1) \tag{1.17}$$

Так при використанні логарифмічного зважування, терміни, що зустрічалися наприклад, 0 разів у документів, матимуть значення $tf = log_{10}(1) = 0$, для 10 разів $tf = log_{10}(11) = 1.04$, для 100 – $tf = log_{10}(101) = 2.004$ і так далі.

Другий фактор у tf-idf використовується для надання більшої ваги словам, які зустрічаються лише у кількох документах. Терміни, обмежені кількома документами, корисні для виділення цих документів з решти колекцій, а ті що зустрічаються по всій колекції є менш корисними. Частота терміну t у документі df_t є кількістю документів у яких він зустрічається.

Дискримінаційні слова, що зустрічаються рідко, підкреслюються за допомогою зворотної частоти документа або idf та визначається за допомогою

виразу $\frac{N}{df_t}$, де N – загальна кількість документів у колекції, а df_t – кількість документів, у яких зустрічається термін t. Вага терміну зростає відносно кількості появ у всіх документах, так найменша вага для терміну, що зустрічається у всіх документах становить 1, та зростає в залежності від унікальності. Через велику кількість документів у корпусах чи колекціях, цей показник також обмежується логарифмічною функцією. Значення зворотної частоти документа обчислюється наступною формулою:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right) \tag{1.18}$$

Таким чином, зважене значення tf-idf, $w_{t,d}$ для слова *t* у документі *d* поєднує частоту членів $df_{t,d}$ з idf за формулою:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{1.19}$$

TF-IDF є способом зважування матриць спільного входження в пошук інформації, а також відіграє у багатьох інших аспектах NLP.

1.6.2 Word2vec

Група пов’язаних моделей Word2vec, або неглибока двошарова нейронна мережа, що визначає асоціації між словами під час навчання на великих корпусах тексту. В цьому алгоритмі кожне слово представлено як числовий вектор, що допомагає визначати семантичні та синтаксичні якості слів та за допомогою простих математичних функцій, таких як косинус подібності, може показувати рівень подібності за семантичними ознаками. Також набір моделей Word2vec може бути корисним при виявленні синонімів та здатний пропонувати додаткові слова для генерації тексту.

Підхід Word2vec полягає в навчанні класифікатора на задачах двійкового передбачення, де в результаті беруться результати ваги класифікатора, як показник вбудування слів. Ідея полягає в використанні тексту як неявно контрольовані навчальні дані для класифікатора. Тобто, слово *c*, яке зустрічається біля цільового

слова, виступає як еталон ймовірності появи слова *c* перед цільовим словом. Так формується метод самонагляду в нейронних мережах, де нейронна модель, що може використовувати наступне слово в поточному тексті як контрольний сигнал аналізує це слово та визначає вбудоване представлення для кожного наступного слова. Вбудування Word2vec є статичними, тобто метод визначає лише одне фіксоване вбудування для кожного слова у словнику.

Word2vec замінює передбачення слів на двійкову класифікацію спрощуючи завдання, а також полегшує навчання класифікатора логістичної регресії, замінюючи багатошарові нейронні мережі з прихованими шарами, які потребують більш складних алгоритмів навчання

1.7 Аналіз настроїв

Аналіз настроїв або сентимент-аналіз є однією з складових аспектів обробки природньої мови, що покликаний для класифікації настрою, визначення емоційного тону та аналізу глибини почуттів у тексті. Аналіз настроїв допомагає системам краще розуміти відношення людей до певних сутностей та взаємодію між ними.

Категоризація відносить текст до трьох основних класів: негативного, позитивного та нейтрального, але кількість цих класів може розширюватись в залежності від потреб замовників та областей впровадження таких систем.

Сентимент-аналіз може застосовуватись у різних галузях, зокрема в комерційних цілях для підвищення ефективності при побудові маркетингових планів, пропонування клієнтам продуктів чи послуг. Його успішно застосовують в соціальних мережах для виявлення емоцій користувачів, при аналізі оцінок на певні послуги чи товари для визначення задоволеності клієнтів, аналізу новинних статей, тощо.

Для реалізації аналізу настроїв можуть використовуватися різноманітні методи, включаючи машинне та глибоке навчання, системи на основі правил та експертної оцінки та інші. При такому аналізі враховується контекст та мовні

варіанти такі, як різні діалекти, мовні рівні, соціолекти та інші форми мови. Для кожної мови при такому аналізі враховуються географічні, соціокультурні та інші фактори, так як кожна мова відрізняється одна від одної за граматикою, лексикою, вимовою, тощо.

Для вирішення завдань цієї класифікації можуть застосовуватись стандартні наївні класифікатори Байєса, але їх ефективність зазвичай покращується впровадженням невеликих змін у підході. Так для класифікації настроїв і ряду інших завдань класифікації тексту, визначення чи зустрічається слово чи ні має більше значення ніж частота його виникнення. Якщо брати до уваги цей фактор, та використовувати обрізання кількості слів у кожному з документів, то це часто покращує продуктивність аналізу при його застосуванні.

Цей підхід називається бінарним мультиноміальним наївним Байєсом. Для навчання застосовується той самий класифікатор Байєса, за винятком того, що для кожного документу видаляються всі повторювані слова перед об’єднанням їх в один великий документ під час навчання, а також дублікати видаляються в тестових документах.

Важливу роль у цій класифікації є обробка заперечень, що виражається приставкою “не” повністю змінює висновки з результуючого предикату. Так заперечення негативного тону може використовуватись для отримання позитивного сентименту.

Базовим підходом при аналізі настроїв для вирішення проблеми заперечення зазвичай під час нормалізації додавати префікс “не” до кожного токену логічного заперечення (не, ні, ніколи) до наступного розділового знаку, для прикладу фраза:

«Мені не сподобався цей фільм, але я…» перетворюється на «Мені не не_сподобався не_цей не_фільм, але я…».

Таким чином новоутворені слова з часткою не, частіше зустрічатимуться в негативному документі та виступатимуть сигналом для негативних настроїв, тоді як слова з цією часткою, але з початковим негативним тоном набудуть позитивних асоціацій.

В деяких ситуаціях може бути недостатньо навчальних даних для навчання точних наївних класифікаторів Байєса, використовуючи всі слова в навчальному наборі для отримання позитивних чи негативних настроїв. У таких випадках характеристики таких слів можливо отримати з лексиконів настроїв, тобто списку слів, що попередньо анотовані позитивними чи негативними настроями.

Поширеним способом лексиконів у наведеному класифікаторі є додавання функції, яка вираховується щоразу, коли зустрічається слово з цього лексикону. Таким чином при появі слова з цього словника, що має наприклад позитивне значення, функція буде підраховувати ці слова окремо від основного класифікатору.

Окрім наївного класифікатора Байєса існують й інші підходи та методи, які використовуються при аналізі настроїв. Впроваджуються методи опорних векторів (SVM) та нейронні мережі, такі як рекурентні нейронні мережі (RNN) чи трансформатори. Враховуються додаткові мовні особливості, такі як підсилювачі форм слів або символи пунктуації чи емодзі, а також впроваджуються комбіновані моделі для об’єднання результатів.

1.8 Частиномовний аналіз

Частиномовний аналіз (POS) є процесом маркування частин мови для кожного слова в тексті. Віднесення слова до частини мови мови може грунтуватися на його визначенні в контексті, а також зв’язках із сусідніми та пов’язаними словами у словосполученнях, реченнях чи абзацах. POS-тегування в основному поділяється на дві групи: засновані на правилах і стохастичні.

Багато програм для обробки природної мови використовують стохастичні методи для визначення частини мови. Привабливість стохастичних методів перед традиційними методами, що базуються на правилах заснованими на правилах, полягає в простоті автоматизованого збору необхідної статистики. Крім того, програми, що базуються на правилах, часто складні в реалізації і не такі надійні.

Стохастичні визначники досягли високого ступеня точності, не покладаючись на чистий синтаксичний аналіз вхідних даних. Ці POS-тегери покладаються на приховану Маркова модель (HMM), яка фіксує лексичну та контекстну інформацію. Параметри цієї моделі, моделі можуть бути оцінені на основі тегованого або нетегованого тексту. Після того, як параметри оцінені, вхідним даним автоматично призначається найбільша ймовірність послідовності тегів на основі моделі. Продуктивність таких моделей покращується за рахунок більш високого рівня методів попередньої обробки або шляхом ручного налаштування.

Хоча стохастичні тегери містять вищий ступінь точності, існує велика надмірність при генерації перестановок для послідовності тегів. Тегери на основі правил зменшують таку надмірність за допомогою невеликого набору значущих правил на противагу великим таблицям статистики, необхідних для стохастичної моделі. Ці тегери базуються на формальному синтаксисі мови.

На практиці для створення більш надійних моделей та для зменшення рівню надмірності стохастичних моделей може використовуватись комбінація двох ймовірнісних моделей до застосування моделі на основі правил. Точність таких алгоритмів є високою та може досягати 97% для понад 15 мов, як було виявлено в одному з досліджень з використанням дерев Universal Dependency (UD).

Мови розглядалися як соціологічні об'єкти як кластери властивостей, що поділяються групою мовців і об'єднані разом як природні мови. Цими властивостями були списки звуків, слів та морфем. Будь-які інші властивості розглядалися як універсальна логіка або пов'язані з індивідуальними звичками.

Як соціологічні об'єкти, природні мови постійно змінюються або через фігури мови, або через контекст слів, або під впливом людської природи. Слова можуть мати одне фіксоване значення у словнику. Однак, при вживанні у реченні, його значення може змінюватися залежно від контексту або зв'язку з іншими словами. Як наслідок, визначення контексту слова в тексті за своєю суттю ускладняється.

Формальна мова є частиною природної мови. Вона існує лише у добре сформованих реченнях, оскільки певні правила можуть легко визначити структуру формального речення. Однак, вона не може визначити семантику через свою залежність від фіксованих правил або коли структура перестає бути формальною.

Кожне речення має структуру, що визначена окремо для кожної мови, її можна представити у вигляді зв’язного дерева, де кожне слово розділено на окрему частку мови та розміщено у відповідному порядку.

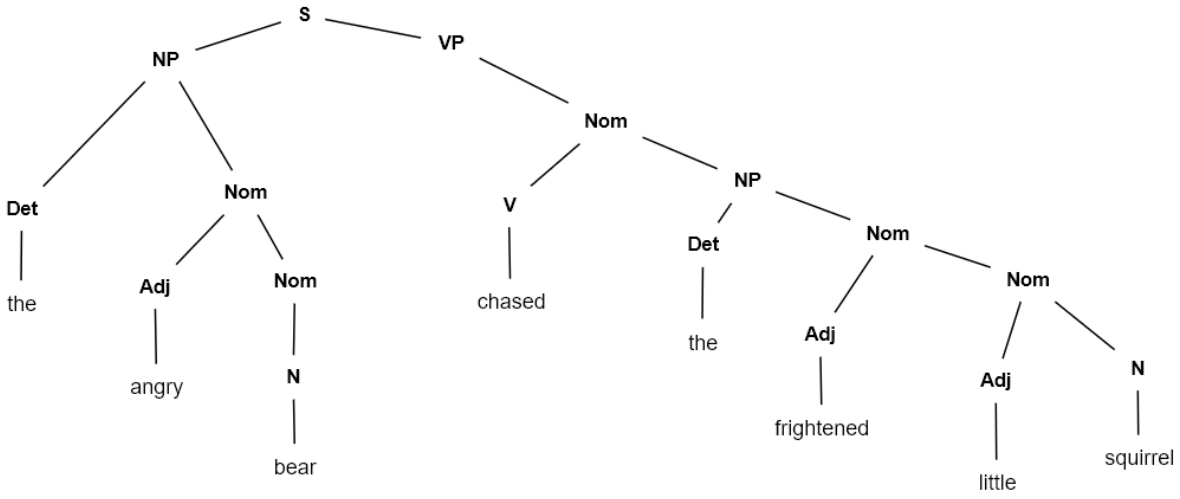


Рисунок 1.8.1 – Частиномовний аналіз речення

1.9 Розпізнавання іменованих сутностей

Розпізнавання іменованих сутностей (NER) – це завдання обробки природної мови, що спрямоване на виявлення фрагментів тексту, що містять власні назви в тексті та їх класифікацію до визначених тегів,

Найпоширенішими з таких тегів є PER (особа), LOC (місце розташування), ORG (організація) та GPE (геополітична сутність). Однак цей список може розширюватись та включати окрім осіб, місць, організації також дати, час, числа та інші важливі елементи. Застосування NER є корисним в багатьох аспектах обробки природньої мови при вилученні смислової інформації, аналізі настрою відносно певної сутності, побудові баз знань та іншого. Іменовані сутності є першим етапом

при перевірці пошукових запитів, зв’язуванні тексту з інформацією у структурованих структурах та при відповідях на запитання.

На відміну від частиномовного аналізу, в якому немає проблем при сегментації, оскільки кожне слово аналізується окремо на отримує власний тег, завдання розпізнавання іменованих сутностей також включає в себе пошук та позначення проміжків тексту або знаків пунктуації, що можуть містити такі об’єкти, тому завдання такого аналізу ускладняється через неоднозначність сегментації.

Для вирішення завдання NER використовуються різні методи та техніки машинного навчання. Одними з популярних алгоритмів є умовні випадкові поля (CRF), приховані марківські моделі (HMM), а також глибокі нейронні мережі, такі як рекурентні нейронні мережі (RNN) та трансформатори, такі як BERT. Окрім алгоритмів можуть бути використані великі NLP моделі, щ обули навчені на великих обсягах інформаційних текстів та мають представлення про іменовані сутності.

1.9.1 Прихована марківська модель

Прихована марківська модель (HMM) – ймовірнісна модель послідовності, що обчислює розподіл ймовірностей за можливими послідовностями тегів та обирає найкращу з них. HMM заснована на розширенні ланцюга Маркова, моделі що відображає ймовірність послідовностей випадкових величин, станів, кожен з яких може приймати значення з деякого набору. Цей набір може представляти послідовність слів, тегів, чи символів, що представляють будь-що.

В ланцюгу Маркова припущення для передбачення майбутнього в послідовності йде лише з урахуванням поточного стану, де всі стани до поточного не впливають на припущення майбутнього.

Припущення Маркова для послідовності значень $q_1, q_2, \dots, q_i$ виглядає наступним чином:

$$P(q_i = a|q_1 \ldots q_{i-1}) = P(q_i = a|q_{i-1}) \tag{1.20}$$

На Рисунок 1.9.1 – Ланцюг Маркова на прикладі гри з перегортанням монет показано простий ланцюг Маркова для призначення ймовірності послідовності перегортання монети для значень “Орел” та ”Решка”. Стани представлені як вузли на графі, а переходи з ймовірностями – як ребра. Переходи є ймовірностями , значення дуг що виходять із заданого стану – повинні дорівнювати 1.

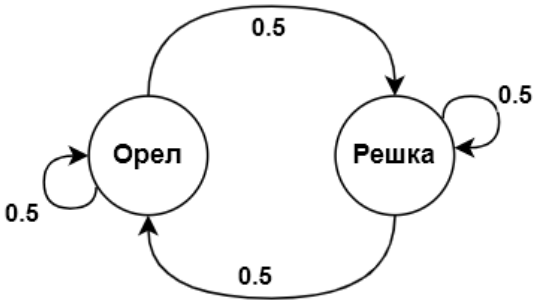


Рисунок 1.9.1 – Ланцюг Маркова на прикладі гри з перегортанням монет

Ланцюг Маркова задається наступними компонентами:

$Q = q_1, q_2, \ldots, q_N$ – набір з N станів;

$A = a_{11}, a_{12}, \ldots, a_{N1}, \ldots, a_{NN}$ – матриця ймовірності переходу A, кожен a_{ij} – представляє ймовірність переходу від стану *i* до стану *j*, так що $\sum_{j=1}^{n} a_{\text{ij}} = 1 \; \forall_{\text{i}}$;

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ – початковий розподіл ймовірності за станами, π_{i} ймовірність початку ланцюга в стані *i*. Деякі стани *j* можуть мати значення $\pi_{\text{j}} = 0$, тобто вони не можуть бути початковими станами, так $\sum_{i=1}^{n} \pi_{\text{i}} = 1$.

Ланцюг Маркова корисний при обчисленні ймовірності для послідовностей спостережуваних подій, але не у випадках коли такі події приховані. Наприклад при аналізі частин мови в тексті аналізується саме послідовність слів, з яких потім класифікуються теги.

Для спостереження за прихованими подіями використовується прихована модель Маркова, визначається вона наступними компонентами:

$Q = q_1, q_2, \ldots, q_N$ – набір з N станів;

$A = a_{11}, a_{12}, \ldots, a_{N1}, \ldots, a_{NN}$ – матриця ймовірності переходу A, кожен a_{ij} – представляє ймовірність переходу від стану *i* до стану *j*, так що $\sum_{j=1}^{n} a_{\text{ij}} = 1 \; \forall_{\text{i}}$;

$O = o_1, o_2, \ldots, o_T$ – послідовність спостережень T, кожне з яких взято зі словника $V = v_1, v_2, \ldots, v_V$;

$B = b_i(o_{\text{t}})$ – послідовність ймовірностей спостережень, кожна з яких виражає ймовірність спостереження або генерується зі стану q_i;

$\pi = \pi_1, \pi_2, \ldots, \pi_N$ – початковий розподіл ймовірності за станами, π_{i} ймовірність початку ланцюга в стані *i*. Деякі стани *j* можуть мати значення $\pi_{\text{j}} = 0$, тобто вони не можуть бути початковими станами, так $\sum_{i=1}^{n} \pi_{\text{i}} = 1$.

Прихована модель Маркова першого порядку створює два спрощених припущення. По-перше, як і у випадку з ланцюгом, ймовірність конкретного стану залежить лише від попереднього стану:

$$P(q_i|q_1, \ldots, q_{i-1}) = P(q_i|q_{i-1}) \tag{1.21}$$

По-друге, ймовірність вихідного спостереження залежить лише від стану, який створив спостереження q_i, а не від будь-якого іншого стану чи спостереження:

$$P(o_i|q_1, \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \tag{1.22}$$

1.9.2 Умовні випадкові поля

Незважаючи на ефективність розглянутої моделі HMM, вона потребує ряду доповнень для досягнення високої точності. Підвищення випадків зустрічі невідомих слів, які з'являються частіше через швидкі зміни в мові, створюються нові власні назви або абревіатури, тощо.

Для вирішення цих проблем була створена дискримінаційна модель послідовності, заснована на логарифмічно-лінійних моделях – CRF (Conditional Random Fields) умовні випадкові поля.

Припускається, що для послідовності вхідних слів $X = x_1, \dots, x_n$ обчислюється послідовність тегів $Y = y_1, \dots, y_n$. У HMM для обчислення найкращої послідовності тегів, яке максимізує $P(Y|X)$, використовується правило Байєса та ймовірність $P(Y|X)$:

$$\begin{aligned} \hat{Y} &= \underset{Y}{\operatorname{argmax}}\, p(Y|X) \\ &= \underset{Y}{\operatorname{argmax}}\, p(X|Y)p(Y) \\ &= \underset{Y}{\operatorname{argmax}} \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1}) \end{aligned} \tag{1.23}$$

У CRF навпаки, обчислюється безпосередньо задній $p(Y|X)$, навчаючи CRF для розрізнення можливих послідовностей тегів:

$$\hat{Y} = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} P(Y|X) \tag{1.24}$$

Однак CRF не обчислює ймовірність для кожного тегу на кожному кроці часу, а обчислює логарифмічно-лінійні функції над набором релевантних ознак і ці локальні особливості агрегуються та нормалізуються для створення глобальної ймовірності для всієї послідовності.

Для більш формального представлення CRF використовуються X та Y як вхідні та вихідні послідовності. CRF, як логарифмічно-лінійна модель призначає ймовірність усій вихідній послідовності Y з усіх можливих послідовностей γ, враховуючи всю вхідну послідовність X.

У CRF функція F відображає всю вхідну послідовність X і всю вихідну послідовність Y та вектор ознак. Припускається, що є K ознак з вагою для кожної функції F:

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)} \tag{1.25}$$

Зазвичай також це рівняння описується, витягаючи знаменник у функцію *Z(X)*:

$$p(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right) \tag{1.26}$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right) \tag{1.27}$$

Ці K функції $F_k(X,Y)$ називається глобальною властивістю, оскільки кожна з них є властивістю всієї вхідної послідовності X і вихідної послідовності Y. Вони обчислюються шляхом розкладання на суму локальних ознак для кожної позиції i в Y:

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i) \tag{1.28}$$

Кожна з цих локальних функцій f_k у лінійному ланцюгу CRF може використовувати поточний вихідний маркер y_i, попередній вихідний маркер y_{i-1}, весь вхідний рядок X і поточна позицію i. Це обмеження залежить лише від поточного та попереднього вихідних токенів y_i та y_{i-1}, що характеризують лінійний ланцюг CRF.

1.10 Машинний переклад

Машинний переклад (МП) при застосуванні його у галузі NLP, покликаний зменшити цифровий розрив у доступі до інформації, шляхом перекладу цієї інформації з однієї мови на іншу. Машинний переклад зосереджується на ряді практичних завдань, а саме поширення доступності інформації різними мовами, що відноситься до різних профілів інтересів.

Пошук інформації з використанням не достатньо поширеної мови з обмеженим ресурсом інформації може давати набагато меншу кількість ресурсів

для користувача ніж, наприклад, при використанні англійської мови. Так високоякісний переклад може допомогти у вирішенні таких проблем.

Іншим поширеним використанням машинного перекладу є допомога людям-перекладачам, які також працюють у цьому напрямку для адаптації вмісту інформації до певної мовної спільноти.

Одним з найновітніших застосувань машинного перекладу це переклад мовлення в режимі реально часу для усунення труднощів під час коммунікації, а також його застосування в оптичному розпізнаванні та перекладі тексту на зображенні.

При МП потрібно брати до уваги різну структуру речень у різних мовах під час перекладу, враховувати всі складності для окремо взятих мов, таку як різна кількість слів у вхідному та вихідному тексті, відсутність деяких елементів, порядок частин мови, тощо.

Для вирішення цих завдань використовуються мережі-декодери, або моделі послідовності до послідовності. Стандартним алгоритмом для МП є рекурентні нейронні мережі.

1.10.1 Рекурентна нейронна мережа

Рекурентна нейронна мережа (RNN) – штучна нейронна мережа, яка містить цикл у своїх мережевих з’єднаннях, де значення деякої одиниці прямо чи опосередковано залежить від її власних попередніх виходів як вхідних даних. Такі мережі використовуються для завдань, коли потрібно зіставити вхідну послідовність з вихідною, яка є складною функцією всієї вхідної послідовності.

На відміну від односпрямованих нейронних мереж із прямим передаванням даних, ця штучна нейронна мережа є двоспрямованою, тобто вихід одного вузла впливає на наступні входи того самого вузла.

Здатність використовувати внутрішні стани для опрацювання довільних послідовностей вхідних даних може бути застосована до таких завдань, як розпізнавання несегментованого рукописного тексту і розпізнавання мови.

1.11 Генерація тексту

За створення тексту, вираження думок чи ідей NLP системами відповідає генерація тексту. Результатом роботи моделей для текстової генерації є створення автоматичних відповідей, що відповідають запиту користувача. Основним з його застосувань є генерація відповідей для чат-ботів, написання статей, пошуку різного роду інформації, тощо.

Використання таких моделей спрощує взаємодію користувачів з системами та автоматизує деякі аспекти життєдіяльності, виступаючи асистентом, що значно допомагає зекономити час витрачений на роботу.

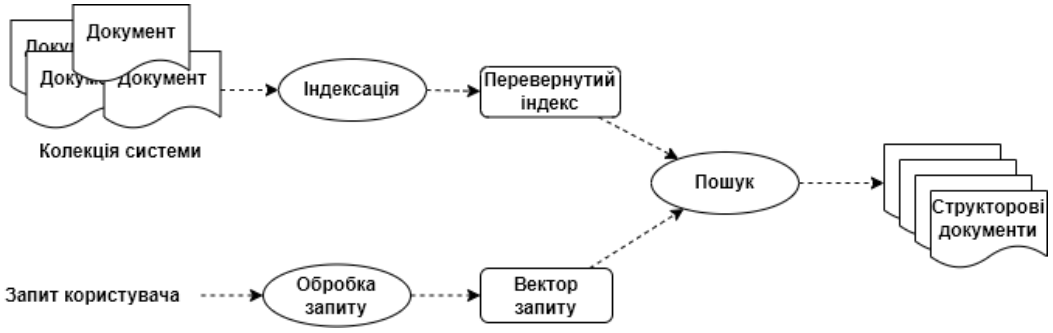
Генерація тексту включає в себе два аспекти, а саме інформаційний пошук та відповідь на запитання.

1.11.1 Інформаційний пошук

Інформаційний пошук (IR) – назва галузі, яка охоплює пошук всіх видів медіа файлів та інформації на основі потреб користувача. Цю систему також називають пошуковою, її застосування відіграє важливу роль при генерації тексту при відповіді на запитання.

Завдання IR-системи називається спеціальним пошуком, в якому користувач створює запит до пошукової системи, яка повертає впорядкований набір документів з деякої колекції. Документом може бути будь-який текст, що був проіндексований системою, тобто веб-сторінки, статті, новини, тощо.

Інформація, що надходить до користувача після запиту структурується у вигляді набору термінів, процес та архітектуру представлені нижче:



(1.29)

Рисунок 1.11.1 – Архітектура IR системи

Зважування термінів і оцінка документів

В IR системі обчислюється вага терміну для кожного слова в окремо взятому документі. Двома поширеними системами зважування термінів є розглянута модель TF-IDF та BM25.

Оцінка документу d здійснюється за косинусом його вектору d із вектором запиту q:

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|} \tag{1.30}$$

Також можна розглядати обчислення косинуса як скалярного добутку одиничних векторів, де спочатку нормалізується вектор запиту та документа до одиничних векторів, шляхом ділення на їх довжини, з подальшим вилучення скалярного добутку:

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|}\cdot\frac{\mathbf{d}}{|\mathbf{d}|} \tag{1.31}$$

Цю формулу можна сформулювати використовуючи значення TF-IDF і описати скалярний добуток як суму добутків:

$$\text{score}(q,d) = \sum_{t\in\mathbf{q}} \frac{\text{tf-idf}(t,q)}{\sqrt{\sum_{q_i\in q}\text{tf-idf}^2(q_i,q)}}\cdot\frac{\text{tf-idf}(t,d)}{\sqrt{\sum_{d_i\in d}\text{tf-idf}^2(d_i,d)}} \tag{1.32}$$

На практиці прийнято апроксимувати це рівняння, спрощуючи обробку запитів. Запити користувачів зазвичай досить короткі, тому кожне слово в запиті, ймовірно буде траплятися лише раз. Нормалізація косинуса для запиту ділення на $|q|$ буде однаковою для всіх документів , тому не змінюватиме рейтинг між будь-якими двома документами D_i та D_j. Отже зазвичай використовується спрощена оцінка для документа d з запитом q:

$$\text{score}(q,d) = \sum_{t\in q}\frac{\text{tf-idf}(t,d)}{|d|} \tag{1.33}$$

Більш складною системою зважування є схема BM25, вона додає два параметри: регулятор k, який регулює баланс між частотою термінів і IDF та параметр b, який контролює важливість нормалізації довжини документа. Оцінка з використанням схеми BM25 для документа d з запитом q:

$$\sum_{t \in q} \overbrace{\log\left(\frac{N}{df_t}\right)}^{\text{IDF}} \overbrace{\frac{tf_{t,d}}{k\left(1-b+b\left(\frac{|d|}{|d_{\text{avg}}|}\right)\right)+tf_{t,d}}}^{\text{weighted tf}}, \tag{1.34}$$

де $|d_{avg}|$ – довжина середнього документу. Якщо k дорівнює нулю, BM25 не використовує частоту термінів, а лише двійковий набір термінів у запиті плюс ідентифікатор idf. Велике значення параметру k призводить до необробленої частоти терміну плюс ідентифікатор idf. Значення b змінюється в діапазоні від 0 до 1, де 0 – відсутність масштабування довжини, а 1 – масштабування за довжиною документа.

Для обчислення оцінки, потрібно ефективно знаходити документи, які містять слова у запиті. Основною проблемою пошуку в IR є знаходження всіх документів $d \in C$, що містять термін $q \in Q$. Структурою даних для цього завдання є перевернутий індекс, який використовується для ефективного пошуку, а також для зручності зберігання корисної інформації, такої як частота документу та кількість кожного терміну в кожному документі. Перевернутий індекс із заданим терміном запиту дає список документів, які містять цей термін.

Він складається з двох частин: словника та дописів. Словник – це список термінів, кожен з яких вказує на список публікацій для терміну. Дописи – список ідентифікаторів документів, пов’язаних з кожним терміном, який може містити інформацію, наприклад частоту термінів або точні позиції термінів у документі. Наявність списку термінів у запиті, допомагає ефективно отримувати список всіх документів-кандидатів разом з інформацією, необхідною для обчислення потрібних оцінок tf-idf.

1.11.2 Відповідь на питання на основі системи інформаційного пошуку

На основі вибраних та оброблених документів система формує саму відповідь. Це може включати в себе створення короткого резюме, виділення ключових фраз або використання інших методів для підготовки інформації для користувача.

Для створення відповіді в системах базованих на інформаційному пошуку є отримання релевантних документів та їх зчитування з виділенням діапазону відповіді в обраному уривку, що містить відповідь.

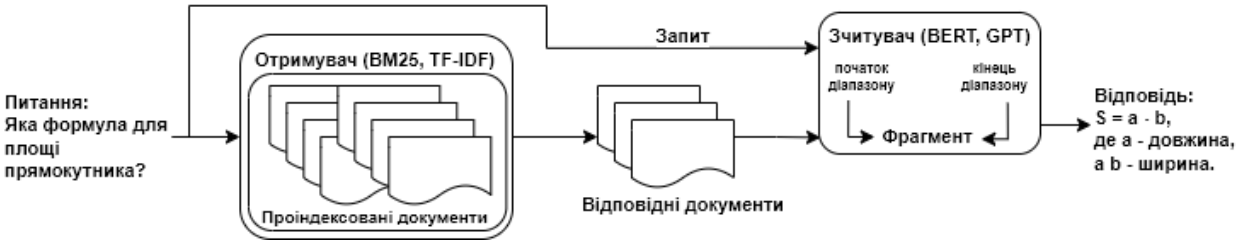


Рисунок 1.11.2 – Генерація відповіді на основі інформаційного пошуку

Інші системи відповідей на запитання вирішують завдання пошуку та зчитування разом, вони отримують запит та велику колекцію документів або отримують доступ до сканування інтернету і повертають фрагмент тексту витягнутого з документа в якості відповіді.

2 ІСНУЮЧІ МАТЕМАТИЧНІ МОДЕЛІ, МЕТОДИ ТА АЛГОРИТМИ

2.1 Наївний Байєс

Мультиноміальний наївний класифікатор Байєса – класифікатор, що робить спрощене припущення щодо взаємодії функцій. Тобто інтуїтивно розділяючи весь набір даних на не впорядкований, не зберігаючи позицію, а лише частоту кожного слова в даних.

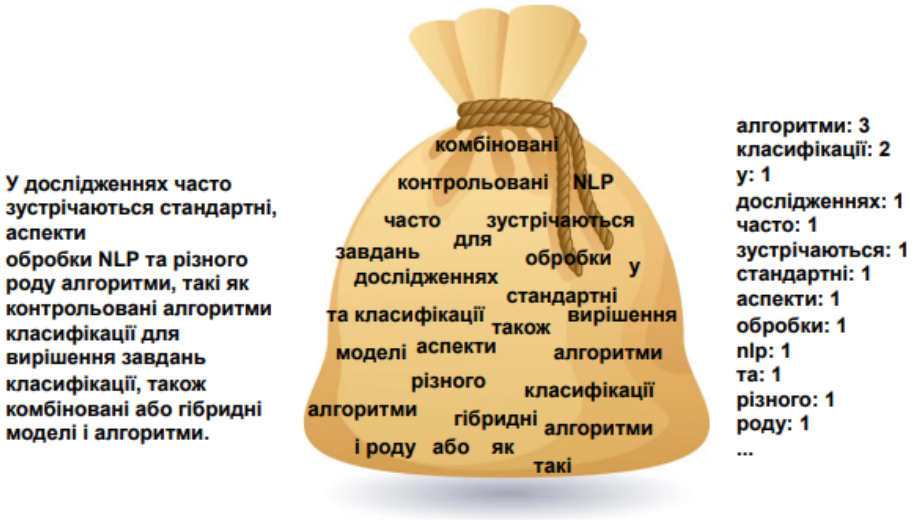


Рисунок 2.1.1 – Мішок слів у класифікаторах Байєса

Алгоритм заснований на теоремі ймовірності Байєса:

$$\mathbf{P}(\mathbf{y} \mid \boldsymbol{x}) = \frac{\mathbf{P}(\mathbf{y}) * \mathbf{P}(\mathbf{x} \mid \mathbf{y})}{\mathbf{P}(\mathbf{x})}, \tag{2.1}$$

де y – клас, x – вхідні дані, $P(y \mid x)$ це апостеріорна ймовірність, тобто ймовірність того, що y відбудеться, якщо відбудеться x, $P(x \mid y)$ є функцією визначеності, а P (x) і P (y) є ймовірностями того, що x і y відбудуться.

Рішення про класифікацію приймається на основі максимального значення апостеріорної ймовірності. На основі вхідних даних розраховується ймовірність для кожного з можливих класів і вибирається максимальне значення.

Першим припущенням є інтуїтивне, тобто про суму слів, де припускається, що слова, які можуть зустрічатися на будь якій позиції відіграють однакову роль у

класифікації. Таким чином, припускається, що ознаки $f_1, f_2, \ldots, f_n$ кодують лише ідентичні слова, а не позицію.

Префікс «наївний» походить від другого припущення, що атрибути умовно незалежні один від одного і всі атрибути однаково важливі. Тобто ймовірності $P(f_i \mid c)$ є незалежними з урахуванням класу c, а отже можуть бути помножені наступним чином:

$$P(f_1, f_2, \ldots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots \cdot P(f_n|c) \tag{2.2}$$

Остаточне рівняння для класу, обраного наївним класифікатором Байєса виглядає:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c) \tag{2.3}$$

Для застосування цього класифікатору до тексту, потрібно врахувати позицію слів, пройшовши за індексом через кожну позицію слова у тексті:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in позиція} P(w_i|c), \tag{2.4}$$

де *позиція* – це перелік всіх індексів слів у тексті.

Для пришвидшення розрахунків цих обчислень, як і для обчислень мовних моделей, використовуються логарифмічні функції, тому зазвичай такі розрахунки мають вигляд:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in позиція} \log P(w_i|c) \tag{2.5}$$

Логарифмічна функція обчислює прогнозований клас як лінійну комбінацію вхідних характеристик для прийняття класифікаційного рішення, від цього класифікатори, подібні до наївного Байєса, а також логістичні регресії, що використовують подібні функції, називаються лінійними класифікаторами.

Зокрема наївний Байєс має подібність до мовного моделювання, тому його можна розглядати як набір специфічних для класу моделей уніграмної мови, в яких кожен клас виступає як екземпляр уніграмної мовної моделі. Оскільки ознаки правдоподібності цієї моделі призначають ймовірність кожному слову $P(w \mid c)$, модель також призначає ймовірність кожному реченню:

$$P(s|c) = \prod_{i \in позиція} P(w_i|c) \tag{2.6}$$

Таким чином, наївна модель Байєса за допомогою двійкової класифікації з визначенням позитивної та негативної ймовірності може призначити ймовірність виникнення речення, наприклад:

P(“I love this fun film” | +) = 0.1×0.1×0.01×0.05×0.1 = 0.0000005, для позитивної ймовірності.

P(“I love this fun film” | −) = 0.2×0.001×0.01×0.005×0.1 = .0000000010, для негативної ймовірності.

Можна зауважити, що для позитивної моделі ймовірність є вищою.

2.2 Логістична регресія

У NLP, логістична регресія – це базовий контрольований алгоритм машинного навчання для класифікації, що також тісно пов’язаний з нейронними мережами. Ця модель виконує завдання спостереження для класифікації в один з двох або більше класів. На відміну від наївної Байєвської моделі, яка являється генеративним класифікатором, логістична регресія є дискримінаційним.

Відмінністю є те, що генеративна модель обирає серед згенерованих варіантів результат та вирішує який з варіантів краще підходить до опису, обираючи його як мітку для вірності. В свою чергу дискримінаційна вивчає результат на наявність певних характеристик і робить висновки, додаючи спостереження до бази знань. Іншими словами, генеративні моделі використовують термін правдоподібності, що пояснює, які потрібні характеристики для генерації даних, якщо зрозуміло що він належить класу *c*.

Дискримінаційна модель – намагається безпосередньо обичислити ймовірність $P(c \mid d)$, надаючи вагу отриманим характеристикам, що покращує його здатність розрізняти можливі класи.

Модель логістичної регресії належить до імовірнісного класифікатора. Гіпотеза логістичної регресії:

$$\mathbf{h}(\mathbf{x}) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-(\omega_0 + \omega_1 x_1 + \ldots + \omega_n x_n)}}, \tag{2.7}$$

де n – кількість ознак, що використовуються в моделі. Нові дані додаються до класу, який для нього більш вірогідний. Для $h(x) > 0{,}5$ дані класифікуються в класі $y = 1$, а для $h(x) < 0{,}5$ вони належать до класу $y = 0$.

Введенням фіктивної ознаки $x_0 = 1$ гіпотеза перетворюється у такий вигляд:

$$\mathbf{h}(\mathbf{x}) = \frac{e^{W^T X}}{e^{W^T X} + \mathbf{1}}, \tag{2.8}$$

де W представляє вектор усіх вагових параметрів, X представляє вектор усіх значень атрибутів, а $W^T X$ є їхнім скалярним добутком. Тоді для гіперплощини поділу класів $h(x) = 0{,}5$ застосовується наступне:

$$e^{W^T X} = \mathbf{1} \tag{2.9}$$

$$W^T X = \sum_{i=0}^{n} \omega_i x_i = \mathbf{0} \tag{2.10}$$

Під час навчання моделі визначаються оптимальні значення параметрів моделі, щоб $h(x)$ правильно визначав клас y для вхідних параметрів x . Функція втрат $L(h(x), y)$ визначає міру відхилення значення гіпотези від точного значення для окремої частини даних. Функція помилки є середнім значенням функції втрат для всіх даних із досліджуваного набору:

$$J(\omega) = \frac{1}{m} \sum_{i=1}^{m} L\left(h\left(x^{(i)}\right), y^{(i)}\right) \tag{2.11}$$

У лінійній регресії функція помилки використовує середнє квадратичне відхилення $h(x)$ від y ; однак, через природу логістичного класифікатора та

потребу, щоб функції помилок і втрат були опуклими, це не підходить функція втрат для логістичної регресії. Функція втрат крос-ентропії відповідає таким вимогам до функції втрат логістичної регресії:

$$\boldsymbol{L\left(h\left(x\right),y\right) = -y\,ln\,h\left(x\right) - \left(1-y\right)ln\,1 - h\left(x\right)} \quad (2.12)$$

2.3 Опорна векторна машина

Подібно до логістичної регресії (LR), з опорною векторною машиною (SVM) необхідно знайти гіперплощину, яка розділяє дані, що належать до різних класів. На відміну від LR, SVM має лише класифікаційне рішення як результат, і він представляє неімовірнісний класифікатор.

Якщо подивитися на приклад бінарного класифікатора, і якщо дані можуть бути розділені лінійно, це означає, що можна побудувати дві паралельні гіперплощини, які розділяють дані різних класів. Область простору між класами називається запасом, і його значення має бути максимальним, щоб помилка класифікації нових даних була мінімальною. Якщо n – кількість рішень, що використовуються в моделі, то гіпотеза має вигляд:

$$\boldsymbol{h\left(x\right) = \omega_0 + \omega_1 x_1 + +\omega_2 x_2\ \ldots + \omega_n x_n = W^T X + \omega_0} \quad (2.13)$$

і рівняння розділової гіперплощини:

$$\boldsymbol{h\left(x\right) = W^T X + \omega_0 = 0} \quad (2.14)$$

Фактори $\boldsymbol{W}$ і $\boldsymbol{\omega_0}$ можна вибрати довільно, але умовно вибирають таке значення, щоб до опорних векторів $\boldsymbol{X^{(sv)}}$ застосовувалося наступне :

$$\boldsymbol{y^{(s\,v)}(W^T X^{(s\,v)} + \omega_0) = 1} \quad (2.15)$$

SVM має дуже хорошу продуктивність у широкому діапазоні проблем і набагато меншу тенденцію до пере підбору, ніж інші методи. На жаль, результат не є імовірнісним, і залежно від кількості функцій і обсягу даних він може бути набагато повільнішим, ніж інші моделі.

2.4 Багатокласова класифікація

Наївна байєсовська модель безпосередньо застосовується до багато класової класифікації, оскільки при розробці моделі нічого не передбачається щодо кількості вихідних значень. Логістичну регресію та метод опорного вектора можна застосувати до багато класової класифікації шляхом об'єднання результатів ряду бінарних класифікаторів одним із таких способів:

- Підхід «Один проти всіх» (OvA) або «Один проти решти» (OvR).
- Підхід один проти одного (OvO).

За допомогою підходу OvA будується k бінарних класифікаторів для k класів. Кожен із класифікаторів отримує один клас і обробляє всі інші класи разом як інший клас. Проблема цього принципу полягає в дисбалансі кількості прикладів в окремих класифікаторах, оскільки кількість прикладів у другому класі набагато більша, ніж кількість прикладів у першому. Нові дані класифікуються в клас, бінарний класифікатор якого створює найвищу ймовірність належності даних до спостережуваного класу.

Для підходу OvO, $\frac{k*(k-1)}{2}$ будуються двійкові класифікатори – по одному для кожної пари класів. Кількість класифікаторів набагато більша, ніж у підході OvA, але навчальний набір даних для кожного бінарного класифікатора менший. Нові дані класифікуються в клас, вибраний більшістю бінарних класифікаторів.

Мультиноміальна логістична регресія є природним розширенням логістичної регресії для роботи з кількома класами. Імовірність належності до класу отримується за допомогою функції:

$$P(y = t \mid x) = e^{\sum_{i=0}^{n} \omega_i^{[t]} x_i} \sum_{j=1}^{k} e^{\sum_i^{[j]} x_i} = \frac{e^{(W^{[t]})^T x}}{\sum_{j=1}^{k} e^{(W^{[j]})^T x}}, \qquad (2.16)$$

де k – кількість класів, n – кількість функцій, $\omega_i^{[t]}$ є ваговим параметром i-ї ознаки для t класу, а X є вектором значення ознаки.

2.5 Гібридна модель

Гібрид наївного методу опорних векторів Байєса (SVM). Ця модель заснована на поєднанні лінійної моделі з байєсівською моделлю та заміні атрибутів частоти слова їхнім вектором співвідношення підрахунку NB позитивних і негативних класів. Основною моделлю став лінійний класифікатор:

$$\boldsymbol{y(i) = sign(W^T x_{(i)} + \in)} \tag{2.17}$$

Якщо $\boldsymbol{f^{(i)}}$ є вектором атрибутів, а вихідним значенням є$\boldsymbol{y_i}$, V – множина атрибутів і $\boldsymbol{f_j}^{(i)}$ кількість входжень атрибута $\boldsymbol{V_j}$ у вхідний текст $\boldsymbol{i}$. Рахункові вектори додатного та від'ємного класу визначаються як:

$$\boldsymbol{p = \alpha + \sum_{i:y^{(i)}=1} f^{(i)}} \tag{2.18}$$

$$\boldsymbol{q = \alpha + \sum_{i:y^{(i)}=-1} f^{(i)}} \tag{2.19}$$

Вектор відношення позитивного до негативного класу визначається як:

$$\boldsymbol{r = log\left(\frac{p/\,||p||_1}{q/\,||q||_1}\right)} \tag{2.20}$$

Щоб поєднати наведені вище рівняння, виконується елементарне множення вектора SVM атрибутів ($\boldsymbol{f}$) і вектора відношення результатів NB підрахунку позитивних і негативних класів ($\boldsymbol{r}$):

$$\boldsymbol{\bar{f}^{(k)} = r * f^{(k)}} \tag{2.21}$$

Отриманий вектор використовується як вхідні дані для стандартного класифікатора SVM.

В дослідженнях [2] наводяться оцінок описаних моделей, що тренерувалися на різних наборах даних.

2.6 Архітектурна модель ScanTextGAN

Основним завданням для побудування цієї моделі є генерація траєкторії сканування, щоб згенерувати послідовність $\boldsymbol{S(T)}$, яка представляє сканування над текстом $\boldsymbol{T = \{\omega_1, \omega_2, \dots, \omega_n\}}$, що складається з послідовності слів, можна визначити наступним чином наступним чином:

$$\boldsymbol{S(T) = \left\{ \dots, (\omega_a^i, t^i), \dots, \left(\omega_b^j, t^j\right), \dots, (\omega_c^k, t^k) \right\}}, \quad (2.22)$$

де $\boldsymbol{t^i}$ - тривалість фіксації над словом $\boldsymbol{\omega_a}$, яке зустрічається у позиції $\boldsymbol{i}$. Не обов'язково, щоб $\mathbf{a < b}$ (слова читаються у лінійному порядку) або щоб $\mathbf{k = n}$ (кількість фіксацій дорівнює кількості слів). Внаслідок регресії, тобто зворотних саккад до попередніх слова, слова також переглядаються. Таким чином, одне й те саме слово може з'являтися в послідовності кілька разів.

Модель ScanTextGAN складається з двох конкуруючих агентів. Умовний генератор, який генерує шляхи сканування на основі текстові підказки та дискримінаторна мережа, яка розрізняє яка відрізняє реальні шляхи сканування людини від згенерованих. Модель ScanTextGAN навчається за допомогою комбінації втрат вмісту тексту: втрат контенту на шляху та втрат від конкурента.

Втрата вмісту на шляху сканування вимірює різницю між прогнозованою траєкторією сканування і відповідною істинною траєкторією сканування. Втрата вмісту тексту реконструює вхідний текст, а втрати конкурента залежать від реального/синтетичного передбачення конкурента над згенерованою траєкторією сканування.

Генератор ScanTextGAN являє собою трансформаторний кодер-декодер на основі трансформатора. Кодер працює на основі текстових вбудувань на основі BERT, які об'єднуються з шумом, щоб зробити вихід генератора недетермінованим. Вихід трансформаторного кодера подається на декодер, який складається з мереж прямого розповсюдження, орієнтованих на конкретне завдання. Першим завданням є генерація шляху сканування однією з гілок, другим завданням іншої гілки є реконструкція 768-вимірного вбудовування токенів CLS у речення. Шлях

сканування виводиться у вигляді часової послідовності ідентифікаторів слів (точок фіксації) $\boldsymbol{\omega_a^i}$, тривалості фіксації $\boldsymbol{t^i}$ та ймовірності кінця послідовності $\boldsymbol{EOS^i}$. На момент виведення довжина $\boldsymbol{L(G)}$ згенерованої траєкторії сканування $\boldsymbol{G}$ визначається наступним чином:

$$\boldsymbol{L(G) = \begin{cases} \min 1 \le k \le M(k) & if\ EOS^k > \tau \\ M & otherwise \end{cases}}, \quad (2.23)$$

де $\boldsymbol{M}$ - максимальна довжина траєкторії сканування, а $\boldsymbol{\tau \in (0,1)}$ - поріг ймовірності. Ми використовуємо $\boldsymbol{\tau = 0.5}$.

Втрати вмісту розгортки намагаються мінімізувати відхилення згенерованих розгорток $\boldsymbol{G(T,N)}$ від істинних розгорток $\boldsymbol{R(T,h)}$ над текстом $\boldsymbol{T}$, де істинні контури сканування записані від людини $\boldsymbol{h}$, а $\boldsymbol{N}$ позначає гауссівський шум $\boldsymbol{N(0,1)}$. Функція втрат $\boldsymbol{L_s}$ має вигляд:

$$\boldsymbol{L_s = \big(G(T,N), R(T,h)\big) = \frac{1}{k}\sum_{i=0}^{k} (\alpha\big(id_g^i - id_r^i\big)^2 + \beta\big(t_g^i - t_r^i\big)^2 + \gamma\big(E_g^i - E_r^i\big)^2)}, \quad (2.24)$$

яка є зваженою сумою трьох доданків.

Перший доданок вимірює похибку між реальними та прогнозованими точками фіксації, яка визначається як середнє квадратичне різниці між згенерованими та реальними ідентифікаторами слів $\boldsymbol{\big(id_g^i - id_r^i\big)}$. Другий член вимірює різницю в тривалості фіксації, яка визначається як середньоквадратична різниця між згенерованою та реальною тривалістю фіксації $\boldsymbol{\big(t_g^i - t_r^i\big)}$. Тривалість фіксації імітує людську увагу до слів у вхідному тексті, що перебувають у центрі уваги. Таким чином, слово з більшою тривалістю фіксації зазвичай є синонімом більшої важливості, ніж інші слова у вхідному тексті. Цей термін помилки доповнює здатність генератора вивчати патерни людської уваги до вхідного тексту. Третій член вимірює середньоквадратичну похибку між прогнозом ймовірності кінця послідовності за реальним та згенерованим розподілами $\boldsymbol{\big(E_g^i - E_r^i\big)}$. Вони зважуються за допомогою параметрів **α, β** і **γ**. Попередні експерименти показали, що оптимізація середньоквадратичної похибки призводить до кращої

продуктивності порівняно з втратами перехресної ентропії для оптимізації виходу ймовірності **EOS**.

Шляхи сканування сильно залежать від лінгвістичних властивостей вхідного тексту. Тому, щоб спрямувати генератор до ймовірної множини реальних даних, приймається реконструкція вбудовування токенів CLS у вхідний текст генератором як допоміжну задачу, оскільки вбудовування токенів CLS кодує глобальне представлення вхідного тексту. Втрати при відновленні змісту тексту $\boldsymbol{L_r}$ задано як:

$$\boldsymbol{L_s = \big(G(T,N), R(T,h)\big) = (BERT(\omega_i^g, \omega_j^g, \dots, \omega_k^g - BERT(\omega_a^r, \omega_b^r, \dots, \omega_n^r))^2}, \quad (2.25)$$

де $\boldsymbol{BERT(\omega_a^r, \omega_b^r, \dots, \omega_n^r)}$ та $\boldsymbol{BERT(\omega_i^g, \omega_j^g, \dots, \omega_k^g)}$ позначають векторні представлення CLS реального та згенерованого тексту відповідно.

Дискримінатор розрізняє реальні та синтетичні шляхи сканування, що надходять до нього. Подібно до генератора, для розрізнення реальних і згенерованих шляхів сканування йому потрібні текстові представлення. Зокрема, дискримінатор складається з двох блоків BiLSTM, які виконують послідовне моделювання над траєкторіями сканування та вбудовуваннями BERT. Виходи двох гілок об'єднуються і передаються на модуль злиття уваги з чотирма головками, за яким слідує ще одна мережа BiLSTM. Приховані стани останнього шару BiLSTM як у прямому, так і в зворотному напрямках об'єднуються і подаються на мережу прямого поширення. Сигмоїдна функція активує вихід мережі прямого поширення. Таким чином, дискримінатор класифікує вхідні траєкторії сканування як справжні або фальшиві.

Мережі генератора та дискримінатора навчаються в режимі гри з нульовою сумою для двох гравців. Втрати задаються за допомогою:

$$\boldsymbol{L_a = \min_G \max_D \begin{matrix} E_{x\sim p_{data}(x)}[\log D(x|T,h)] + \\ E_{z\sim p_z(z)}[1 - \log D(G(z|T,N))] \end{matrix}} \quad (2.26)$$

Таким чином, чисті втрати генератора стають

$$\boldsymbol{L_g = L_s + L_r + E_{z\sim p_z(z)}[1 - \log D(G(z|T,N))]} \quad (2.27)$$

2.7 Мультисемантичний фреймворк для напівконтрольованої класифікації довгих текстів

Класифікація текстів, яка є важливим завданням в області обробки природної мови, швидко зростає з розвитком методів глибокого навчання. Однак, існуючі методи глибокого навчання стикаються з довгими вхідними текстами. Багато досліджень з класифікації довгих текстів використовують усічення тексту або прості методи вилучення ключових слів, що призводить до втрати багатої семантичної та структурної інформації. Крім того, відсутність маркованих навчальних даних і безперервна генерація довгих текстів різних стилів накладають значні вимоги на навчання напівкерованої класифікації довгих текстів.

Для вирішення цієї проблеми, було запропоновано метод гетерогенної мережі уваги, що базується на багатосемантичному проходженні фреймворку мультисемантичного переходу [5]. Було почато розробку гнучкого гетерогенного інформаційного графу для моделювання довгих текстів шляхом вилучення інформації, включаючи ключові слова, сутності, заголовки та їхні багатозначні взаємозв'язки, що може ефективно інтегрувати семантичні зв'язки та конденсувати глобальну інформацію, для збереження важливої семантичної та структурної інформації. Також, в активну розробку було поставлено мультисемантичний фреймворк, здатний витягувати семантичну та структурну інформацію в побудованому гетерогенному інформаційному графі за семантичним ступенем семантичної значущості певних структур. Досліджено експериментальні роботи на чотирьох реальних наборах даних, таких як ThuCNews, SougouNews, 20NG та Ohsumed, що дали значні результати.

2.8 Аналіз сучасних засобів програмної інженерії для вилучення смислової інформації з тексту

Сучасні засоби для вилучення смислової інформації з тексту включають в себе широкий спектр технологій та інструментів, які використовуються в обробці

природної мови (NLP), машинному навчанні та інших суміжних областях. Ці інструменти можуть використовуватися для аналізу тексту, вилучення смислової інформації, розпізнавання іменованих сутностей та вирішення інших завдань у галузі обробки природної мови.

2.8.1 SpaCy

SpaCy – це безкоштовна бібліотека з відкритим вихідним кодом, розроблена компанією Explosion AI та мові програмування Python. Використовується для розширеної обробки природної мови, надає інструменти для токенізації, частиномовного аналізу, вилучення іменованих сутностей, а також інші NLP-задачі.

Застосовується для аналізу тексту, розпізнавання іменованих сутностей, вилучення ключових слів. Розроблена спеціально для виробничого застосування і допомагає створювати додатки, які обробляють і "розуміють" великі обсяги текстів. Її використовують для створення систем вилучення інформації або систем розуміння природної мови та для попередньої обробки тексту при глибокому навчанні.

2.8.2 Natural Language Toolkit (NLTK)

NLTK – це провідна платформа для створення програм на мові Python для роботи з даними людської мови, це безкоштовний проект з відкритим вихідним кодом, керований спільнотою. Вона надає прості у використанні інтерфейси до понад 50 корпусів і лексичних ресурсів, таких як WordNet, а також набір бібліотек для обробки тексту для класифікації, токенізації, стеммінгу, тегування, синтаксичного аналізу та семантичного міркування, обгортки для бібліотек NLP промислового рівня, а також активний форум для обговорень. Дана платформа має практичний посібник, що містить основи програмування та теми з комп'ютерної лінгвістики, а також вичерпну документацію з API, NLTK підходить для лінгвістів, інженерів, студентів, викладачів, науковців та промислових користувачів. NLTK доступний для Windows, Mac OS X та Linux.

2.8.3 AllenNLP

AllenNLP – платформа для вирішення завдань обробки природної мови в PyTorch. Надає широкий набір існуючих реалізацій моделей, які добре задокументовані та розроблені за високими стандартами, що робить їх чудовою основою для подальших досліджень.

AllenNLP пропонує мову конфігурації високого рівня для реалізації багатьох поширених підходів у NLP, таких як експерименти з трансформаторами, багатозадачне навчання, завдання мовного бачення, справедливість та інтерпретація. Це дозволяє експериментувати з широким спектром завдань виключно через налаштування, допомагаючи зосередитись на важливих питаннях у своєму дослідженні.

Бібліотека AllenNLP мала великий успіх у AI2 і за її межами, і її використовували в усій сфері NLP для найсучасніших досліджень.

2.8.4 Stanza

Stanza – це пакет аналізу природної мови Python, подібний до Spacy. Він містить інструменти, які можна використовувати в конвеєрі, щоб перетворити рядок, що містить текст людською мовою, у списки речень і слів, щоб створити базові форми цих слів, їхніх частин мови та морфологічних особливостей, щоб надати аналіз залежностей синтаксичної структури і розпізнавати іменовані сутності, включає токенізацію, розширення багатослівних маркерів (MWT), лематизацію, теги частин мови (POS) і морфологічних ознак, синтаксичний аналіз залежностей і розпізнавання іменованих об’єктів. Набір інструментів розроблено для паралельного використання понад 70 мов із використанням формалізму універсальних залежностей. Також одною з додаткових переваг є підтримка української мови.

Stanza побудовано з високоточними компонентами нейронної мережі, які також забезпечують ефективне навчання та оцінку за допомогою ваших власних анотованих даних. Модулі побудовано на основі бібліотеки PyTorch. Крім того,

Stanza включає інтерфейс Python до пакета Java CoreNLP і успадковує звідти додаткову функціональність, таку як синтаксичний аналіз групи, роздільна здатність кореференції та зіставлення лінгвістичного шаблону.

2.8.5 Flair

Flair – один з найпопулярніших фреймворків глибокого навчання для NLP з відкритим вихідним кодом, створена на основі досліджень машинного навчання Берлінським університетом Гумбольдта. Бібліотека знаходиться у вільному доступі та має активний застосунок у промислових програмах та у нових дослідницьких проектах. Flair є офіційною частиною екосистеми PyTorch, що полегшує навчання власних моделей і експериментує з новими підходами за допомогою вбудовування та класів [24].

Flair дозволяє вам застосовувати до тексту наші найсучасніші моделі обробки природної мови (NLP), такі як розпізнавання іменованих об’єктів (NER), аналіз настроїв, теги частини мови (PoS), спеціальна підтримка біомедичних дані , усунення неоднозначності та класифікація з підтримкою швидко зростаючої кількості мов. Також Flair містить бібліотеку для вбудування тексту, що дозволяє використовувати та комбінувати різні вбудовані слова та документи, включаючи вбудовані та різноманітні трансформери.

2.8.6 Stanford CoreNLP

Stanford CoreNLP – це комплекс інструментів аналізу природної мови, активно розвивається та підтримується Стенфордським університетом, що забезпечує регулярні оновлення та вдосконалення для користувачів. Він є кросплатформним, тобто може бути використаний як бібліотека на різних мовних платформах (Java, Python тощо). Він надає широкий спектр функціональностей для аналізу текстів, включаючи визначення частин мови, лематизацію, розпізнавання іменованих сутностей, аналіз залежностей, розпізнавання відносин між сутностями та інше. Окрім англійської мови, надає різні рівні підтримки для (сучасного

стандарту), арабської, (материкової) китайської, французької, німецької, угорської, італійської та іспанської мов.

Stanford CoreNLP – це інтегрований фреймворк, що поширюється під загальною публічною ліцензією GNU (GPL), що робить його відкритим для розширення та модифікацій згідно з вимогами ліцензії, дозволяючи застосувати низку інструментів аналізу мови до фрагмента тексту. Представлений набір аналізу тексту забезпечує базові можливості для програм розуміння тексту вищого рівня та предметно-спеціальних програм. Інструменти по-різному використовують компоненти на основі правил, імовірнісного машинного навчання та глибокого навчання. Цей інструмент широко використовується в дослідженнях та в індустрії для різноманітних завдань аналізу текстів та розуміння природної мови.

2.8.7 VADER

VADER (Valence Aware Dictionary and Sentiment Reasoner) – інструмент аналізу настроїв на основі правил з відкритим кодом, розроблений компанією Hutto і Gilbert. Призначений для визначення емоційного забарвлення текстових даних.

Використовуючи заздалегідь складений словник, що містить лексичні одиниці, що оцінені за тональністю. Словник включає в себе слова та вирази, які часто використовуються у тексті для підкреслення емоційного забарвлення, враховуючи різноманітність мовних особливостей і виразів для більш точного аналізу тексту.

Слова у словнику VADER розбивають текст на окремі токени і присвоюють кожному ваговий коефіцієнт, який визначає його вплив на оцінку тональності тексту, так кожне слово може по різному впливати на оцінку при такому аналізі.

Оцінка настрою обчислюється за допомогою трьох головних критеріїв, тобто кожне слово містить критерій позитивності, нейтральності або негативну, вони складають співвідношення пропорцій тексту та грають роль у обчисленні нормалізованого, зваженого балу, що обчислюється шляхом підсумовування балів валентності кожного слова в лексиконі, скоригованих відповідно до правил, а потім нормалізується до значення між -1 (найбільш негативний) і +1 (найбільш

позитивний). Цей показник виступає результатом одновимірної міри настрою для певного тексту. Ці критерії представляють необроблену категоризацію кожного лексичного елемента (слів, емодзі та інше) на позитивні, негативні чи нейтральні класи. Цей інструмент вдосконалює цю категорізацію, шляхом впровадження чутливості до порядку слів для багатослівних фраз, використанням підсилювачів форм слів та пунктуації, а також перемикачів полярності заперечень та чутливості контрастних сполучників.

2.8.8 Transformers (Hugging Face)

Transformers – це популярна бібліотека з відкритим вихідним кодом, розроблена компанією Hugging Face, що спеціалізується на обробці природної мови та надає модельний центр, де користувачі можуть легко знаходити, ділитися та використовувати попередньо навчені моделі. Бібліотека має динамічну та активну спільноту з постійними оновленнями, внесками та підтримкою дослідників, розробників і практиків у сфері NLP. Надає можливість створювати, навчати та використовувати моделі обробки природної мови, зокрема моделі на основі трансформаторів.

Transformers підтримує різноманітні попередньо навчені моделі, починаючи від маленьких моделей, придатних для мобільних додатків, і закінчуючи великими моделями, які досягають найсучаснішої продуктивності в різних завданнях NLP. Відомі моделі включають BERT, GPT-2, RoBERTa, T5 та багато інших.

Вона легко інтегрується з популярними фреймворками глибокого навчання, такими як PyTorch і TensorFlow, забезпечуючи гнучкість для користувачів, які віддають перевагу будь-якому з цих фреймворків.

2.8.9 Порівняльна характеристика функціональності

Таблиця 1

Порівняльна таблиця функціональності бібліотек для роботи з NLP

Функція	Назва бібліотеки						
	SpaCy	NLTK	AllenNLP	Stanza	Flair	Transformers (Hugging Face)	Stanford NLP
Кількість мов	50+	10+ (залежить від реалізації)	1 (англійська)	80+ (включаючи українську)	270+ (включаючи українську)	100+ (включаючи українську)	50+ (включаючи українську)
Токенізація	✓	✓	✓	✓	✓	✓	✓
Визначення частин мови (POS)	✓	✓	✓	✓	✓	✓	✓
Визначення іменованих сутностей	✓	✓	✓	✓	✓	✓	✓
Синтаксичний аналіз залежностей	✓	✓	✓	✓	✓	✓	✓
Сегментація речень	✓	✓	✓	✓	✓	✓	✓
Векторне представлення слів	✓	✗	✓	✗	✓	✓	✗
Визначення іменованих сутностей	✓	Обмежено	✓	Обмежено	✓	✓	✗
Правила відповідності	✓	✓	✗	✗	✓	✓	✗
Машинне навчання	✓ (опціонально)	✓	✓	✓	✓	✓	✓
Попередньо навчені моделі	✓	✓	✓	✓	✓	✓	✓
Підтримка багатомовності	✓	✓	Обмежено	✓	✓	✓	✓
Аналіз настрою	✓	✓	✓	✗	✓	✓	✓
Вирішення кореференції	✓	✓	Обмежено	✓	✗	✓	✓
Сумаризація тексту	Обмежено	✓	✗	✗	✗	✓	✗
Синтаксичні дерева	✓	✓	✓	✓	✗	✓	✓
Власні конвеєри	✓	✓	✓	✓	✓	✓	✗
Підтримка активного навчання	✓	✗	✗	✗	✗	✓	✗

Вище представлена порівняльна характеристика функціональності кожної з розглянутих бібліотек для роботи з NLP (Таблиця 1).

Таблиця включає в себе перелік функцій, що присутні в кожній з представлених бібліотек, де наявність конкретної функції відмічена позначкою «✓», а її відсутність «✗», інші позначки представлені у вигляді тексту.

У роботі при практичному розгляді, було використано бібліотеку Stanford NLP, через її мультиплатформенність та можливість застосування у середовищі Java. А також портований на мову програмування Java модуль бібліотеки NLTK, що відповідає за аналіз тональності тексту.

3 РОЗРОБКА ТА АНАЛІЗ ПРАКТИЧНИХ РЕЗУЛЬТАТІВ

Під час дослідження було розглянуто моделі та методи класифікації текстових даних за значною кількістю семантичних ознак та практично оцінено дві з таких моделей при впровадженні їх у програмний застосунок створений на мові програмування Java.

Практичним результатом роботи є об’єднання двох підходів для аналізу настрою, а саме моделі на основі правил та моделі на основі машинного навчання з використанням дерева рішень. Було застосовано декілька підходів при поєднанні різних систем аналізу настроїв, для тестування кожної було використано набір з чотирьох різних корпусів текстів. До них належать корпуси випадкових повідомлень з Твітера, корпус відгуків про товари магазину, корпус відгуків авіакомпаній та корпус оглядів фільмів. В кожному з повідомлень у цих корпусах є попередньо визначена тональність для порівняння результатів систем.

Модель аналізу настрою на основі правил - це метод визначення настрою в тексті на основі заздалегідь визначених правил. Цей підхід включає в себе створення конкретних лінгвістичних правил та шаблонів для визначення класифікації настрою тексту як позитивного, негативного чи нейтрального. Такі правила створюються на основі патернів, ключових слів чи фраз, що свідчать про певний настрій. Тональність тексту може визначатися вмістом в ньому позитивних слів, наприклад слова: “чудовий”, “привітний”, відносяться до позитивних та підвищують рівень позитивного настою тексту, в свою чергу негативні слова: “жахливо”, “ненависний” впливають на негативний настрій. Обидва цих фактори відіграють роль при обчисленні кінцевої оцінки при класифікації настрою тексту.

В моделях класифікації настрою кожне слово або словосполучення має своє математичне представлення у вигляді значення за певними метриками. Так у використаному при практичних дослідженнях інструменті для аналізу настроїв VADER, що побудований на основі правил, як вже було розглянуто, метрика представляє собою діапазон значень від -1 до 1, де значення, що є меншими ніж 0

впливають на негативний настрій в тексті, а значення до одиниці навпаки, для визначення нейтрального тону використовуються значення наближені до 0, тобто діапазон від -0.05 до 0.05.

В іншому підході до побудови моделей на основі машинного навчання для визначення тональності тексту можуть бути використані різні алгоритми під час навчання. Такі моделі включають нейронні мережі, дерева рішень, машинні опорні вектори, лінійні моделі. Деякі з них більш детально розглянуті в роботі під час дослідження. У використаній при практичному аналізі інструменту для аналізу настроїв від Stanford CoreNLP, використовується підхід машинного навчання, модель якого працює з використанням дерева рішень, що аналізують текст та настрої закладені в ньому, а також з використанням рекурсивної тензорної нейронної мережі. Ця модель була навчена на 215 154 фразах у деревах синтаксичного аналізу, що складалися з 11 855 речень. Метриками в цьому підході виступають також 3 критерії класифікації настроїв тексту як позитивного, негативного чи нейтрального. Де кожен з класів представлений як відносне відсоткове значення, що в сумі складає одиницю, найвищий результат аналізу в одній з категорій визначає найімовірніший результат визначення тональності тексту.

3.1 Опис використаних програмних засобів

Практичним результатом роботи є програмна реалізація поєднання систем аналізу тональності тексту у застосунку написаному на мові програмування Java з використанням інтегрованого середовища розробки IntelliJ IDEA.

Програмний продукт розроблявся та тестувався з використанням ноутбуку MSI GF63 8RD з наступними характеристиками:

- процесор Intel Core i5-8300H (2.3 - 4 ГГц);
- ОЗП 8.00 ГБ (DDR4 2666 МГц);
- графічний адаптер Nvidia GeForce GTX 1050Ti (дискретна).

Продукт було розроблено під операційною системою Windows 10, він функціонує на персональних комп'ютерах з операційною системою Microsoft Windows 10 та вище.

3.1.1 Java

Рисунок 1. – Логотип мови програмування Java

Java – це об'єктно-орієнтована мова програмування, яка широко використовується у веб-консолях, графічних інтерфейсах користувача, мобільних додатках, розробці ігор, вбудованих системах і настільних програмах. Окрім цього, Java також використовується для розробки програмного забезпечення для пристроїв. Її використання не обмежене застосуванням у розробці комп'ютерних чи мобільних програм, вона також застосовується при створенні програмного забезпечення в електронних пристроях, таких як телевізори, кондиціонери, пральні машини, тощо.

Це мова високого рівня, яку також легко читати та розуміти. В її основу покладено принцип «написати один раз, запустити будь-де», тобто скомпільований код Java працюватиме на всіх Java-сумісних платформах без необхідності повторної компіляції.

Java є С-подібною мовою, тобто базується на мовах С і С++. Перший компілятор Java був розроблений Sun Microsystems і був написаний на С з використанням деяких бібліотек з С++. Файли Java перетворюються у формат бітового коду за допомогою компілятора, який потім виконує інтерпретатор Java. Код Java виконується на віртуальній машині Java (JVM) – середовищі виконання.

У Java є три випуски. Програмісти можуть вивчити будь-яке з цих видань на основі програми, яку вони хочуть створити.

– Стандартна версія Java - містить основні бібліотеки, такі як java.lang, java.util тощо.

– Java Enterprise Edition - включає Java API , такі як JMS, EJB, JSP/сервлети тощо.

– Java Micro Edition – це видання використовується для програмування Java у мобільних телефонах, приставках, кишенькових комп'ютерах тощо.

Найбільш поширеною версією Java є Java SE (Standard Edition). Java SE містить основи Java – для більшості програм потрібна стандартна версія.

Існує три основні компоненти мови програмування Java, зокрема:

JVM (Java Virtual Machine) – це незалежний від платформи компонент Java, який забезпечує середовище для виконання програм Java. Отже, JVM завантажує код, перевіряє код, виконує код і забезпечує середовище виконання.

JRE (Java Runtime Environment) – створює середовище, у якому можна запускати та виконувати файли Java. Це програмний пакет, який містить JVM разом із бібліотеками класів Java і завантажувачем класів Java.

JDK (Java Development Kit) – представляє собою об'єднання JRE та середовища розробки програмного забезпечення. JDK містить приватну JVM і кілька інших ресурсів для розробки програми Java.

Основними поняттями Java є її функції об'єктно-орієнтованого програмування (ООП). ООП спрощує розробку програмного забезпечення та додатків, а також обслуговування, надаючи деякі концепції, такі як:

Об'єкт – це елемент або сутність, яка має стан і поведінку.

Клас – це сукупність кількох об'єктів. Це деякий шаблон, що може бути використаний для створення необмеженої кількості об'єктів та для організації коду.

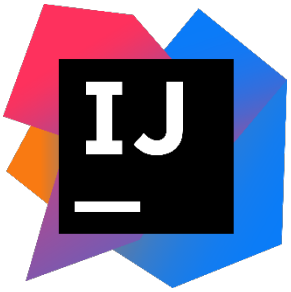
Успадкування – це концепція, в основу якої покладена передача класу властивостей та поведінки його батьківського класу.

Поліморфізм – це процес виконання одного завдання різними способами.

Абстракція – використовується, щоб приховати внутрішні деталі та показати лише важливі деталі.

Інкапсуляція – зв’язування або обгортання коду та даних в один блок. Клас в Java є прикладом інкапсуляції.

3.1.2 IntelliJ IDEA



(Рисунок 1. – Логотип інтегрованого середовища розробки IntelliJ IDEA)

IntelliJ IDEA – це інтегроване середовище розробки (IDE) для Java і Kotlin, призначене для максимального підвищення продуктивності розробника. Це середовище є кросплатформним, тобто забезпечує стабільну роботу в Windows, macOS і Linux.

Цей інструмент полегшує виконання рутинних та повторюваних завдань, допомагаючи в генерації такого коду, також забезпечує розумне контекстно-залежне завершення коду, статичний аналіз коду та рефакторинг та швидку навігацію по всьому проекту та у файлах вихідного коду.

Розробка сучасних програм передбачає використання кількох мов, інструментів, фреймворків і технологій. IntelliJ IDEA розроблено як IDE для мов JVM, а саме: Java, Kotlin, Scala та Groovy, але численні плагіни можуть розширити цей список та додати такі мови як: Python, Ruby, PHP, SQL та інші.

Це середовище розробки випускається у двох версіях:

– IntelliJ IDEA Community Edition – безкоштовна IDE, створена на основі коду з відкритим вихідним кодом, яка надає основні функції для ентузіастів Java і Kotlin.

– IntelliJ IDEA Ultimate – повнофункціональна Java та Kotlin IDE. Ця версія містить усі функції видання Community , а також додає підтримку мов , на яких зосереджені інші IDE на основі платформи IntelliJ, а також підтримку

різноманітних серверних і зовнішніх фреймворків, серверів додатків , інтеграцію з базою даних і профілювання інструменти та інше.

3.1.3 Опис програмного застосунку

Програмний застосунок представляє собою програму, що здійснює аналіз тональності тексту, що надходить у вигляді файлу формату CSV. Блок схема алгоритму роботи програми наведена нижче:

Рисунок 3.1.1 – Алгоритм роботи програми

Приклад вхідних даних, що використовуються для аналізу виглядає наступним чином:

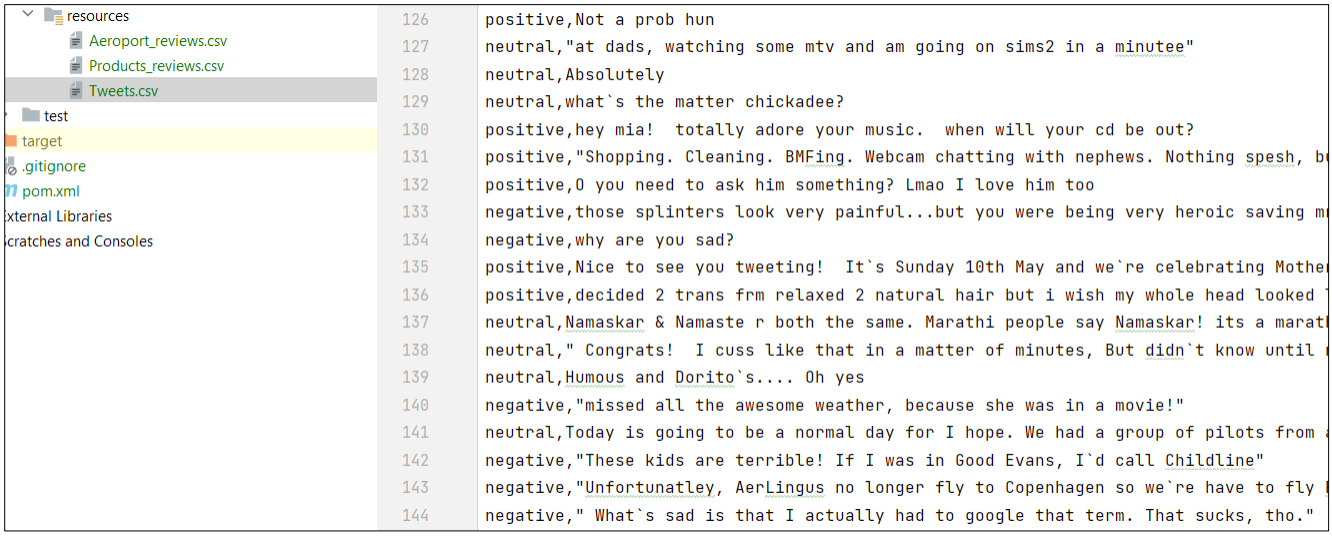


Рисунок 3.1.2 – Приклад вхідних текстових даних з корпусу

Значення в корпусі поділяються на стовпці при використанні файлу формату CSV, де кожне значення поділяється комою. Так першим значенням є очікувана тональність тексту, представлена трьома значеннями: positive – для позитивного настрою, negative – для негативного та neutral – для нейтрального. Наступний стовбець містить саме текстові дані, що будуть аналізуватися та результат аналізу тональності буде порівняно з очікуваною.

3.2 Поєднання систем аналізу

Для підвищення точності визначення тональності при аналізі тексту в роботі застосовано підходи об’єднання результатів, що реалізовані у програмному продукті. Кожен з результатів має відсотковий критерій точності та визначається кількістю вірно визначених настроїв кожного вхідного тексту, включеного в окремо взятий корпус. Отримані результати порівнюються з контрольними замірами окремого відпрацьованими системами на кожному з корпусів та представлені у вигляді порівняльних таблиць.

Системи, що були використані для розробки відрізняються за метриками оцінювання рівнів тональності настроїв, тому їх потрібно звести до однієї системи для компонування результатів. За основу була взята метрика результатів системи побудованої на машинному навчанні, тобто розділення кожного класу емоційних відтінків на 3 класи, де кожен клас має відсоткову частку від цілого значення.

Потрібно нормалізувати другу систему, результат аналізу якої представлений у вигляді діапазону чисел від -1 до 1, для нейтральної тональності абсолютне значення 0.05 є граничним найнижчим фактором визначення нейтральності тональності, а 0 в свою чергу найвищим. Для негативних та позитивних чисел відсоток буде вираховуватись в абсолютному діапазоні від 0.06 до 1, де 1 є максимальною границею, а 0.06 відповідно мінімальною.

Формула нормалізації нейтральної тональності виглядає наступним чином:

$$P = \frac{1-v}{max} \tag{3.1}$$

Для нормалізації чисел, що відображають негативні та позитивні настрої формула наступна:

$$P = \frac{v-min}{max-min}, \tag{3.2}$$

де P - відносний відсоток ймовірності конкретної тональності, v – значення метрики в діапазоні, max – максимальне граничне значення в діапазоні, min - мінімальне граничне значення в діапазоні.

```
sentimentValue = actualVaderSentiment.equals("neutral")
        ? (1 - (Math.abs(sentimentValue) / 0.05))
        : (Math.abs(sentimentValue) / 0.06) / (1 / 0.06);
```

Рисунок 3.2.1 – Реалізація нормалізації у програмі

3.2.1 Зважений середній результат

Кожна з систем по різному показує ефективність аналізу, одна система може давати кращі результати при аналізі порівняно з іншою, тому одним з варіантів підвищення точності визначення тональності тексту є присвоєння цим системам різної ваги в результаті аналізу з подальшим визначенням середнього зваженого результату для обох з систем.

Формула для отримання середнього зваженого результату наступна:

$$A_{s_1,s_2} = \frac{(s_1 * w_1)+(s_2 * w_2)}{w_1 + w_2}, \quad (3.3)$$

де A_{s_1,s_2} - середній зважений результат, s_1 та s_2 – отримані числові показники відповідних систем, w_1 та w_2 – вага кожної з систем.

```
double firstSystemResult = actualSentimentValue * firstSystemWeight;
double secondSystemResult = Math.abs(sentimentValue) * secondSystemWeight;

double weightedAverageValue = (firstSystemResult + secondSystemResult) / (firstSystemWeight * secondSystemWeight);

String finalSentiment = weightedAverageValue == 0 ? "neutral"
        : weightedAverageValue < 0 ? "negative" : "positive";
```

Рисунок 3.2.2 – Застосування середнього зваженого результату

Результати практичного застосування цього підходу можна побачити на Таблиця 2.

Таблиця 2

Застосування підходу зваженого середнього результату

Назва системи	**Використані тестові корпуси**		
	Відгуки на авіакомпанії (14 000+)	**Відгуки на товари (18 000+)**	**Повідомлення в Twitter (27 000+)**
Система 1 (Vader)	**53.93%**	**65.65%**	**62.99%**
Система 2 (CoreNLP)	**42.84%**	**53.05%**	**53.08%**
Зважений середній результат	**66.53%**	**75.13%**	**74.79%**

Порівнявши результати застосування методу, можна зауважити приріст в точності, але результат не є оптимальним.

3.2.2 Підхід голосування

Метод базується на принципі більшості - якщо більше систем "голосують" за певний настрій, то цей настрій вважається загальним. Кожна з систем проводить власний аналіз та висуває результат у вигляді голосу за певний настрій, кожен з настроїв може бути представлено в числовому вигляді. Результат “голосування” обирається за принципом більшості, настрій, що набрав більшу кількість “голосів” вважається загальним для такої системи. Коефіцієнти для голосів можуть мати різну вагу та базуватися на впевненості поєднуваних систем.

```
Map<String, Integer> sentimentVotes = new HashMap<>();
sentimentVotes.put("negative", -1);
sentimentVotes.put("neutral", 0);
sentimentVotes.put("positive", 1);

String finalSentiment = sentimentVotes.get(actualCoreNLPSentiment) + sentimentVotes.get(actualVaderSentiment) == 0
        ? "neutral" : sentimentVotes.get(actualCoreNLPSentiment) + sentimentVotes.get(actualVaderSentiment) < 0
        ? "negative" : "positive";
```

Рисунок 3.2.3 – Застосування підходу голосування

Результати практичного застосування цього підходу можна побачити нижче (Таблиця 3).

Таблиця 3

Застосування підходу голосування

Назва системи	**Використані тестові корпуси**		
	Відгуки на авіакомпанії (14 000+)	**Відгуки на товари (18 000+)**	**Повідомлення в Twitter (27 000+)**
Система 1 (Vader)	**53.93%**	**65.65%**	**62.99%**
Система 2 (CoreNLP)	**42.84%**	**53.05%**	**53.08%**
Метод голосування	**63.63%**	**71.30%**	**70.22%**

Результат застосування методу голосуванням надає незначний приріст в порівнянні з іншими, що також не є найменш оптимальним підходом для об’єднання.

3.2.3 Адаптивне поєднання моделей

Поєднувані системи можуть мати різну впевненість у визначенні тональності проаналізованого тексту, в такому випадку результат моделі, що відображає кращі числові результати повинен підвищувати вагу значення отриманого при аналізі, підвищуючи ймовірність вважатися загальним для проаналізованого тексту.

В застосунку це досягається впровадженням двох змінних числових значень, що впливають на кінцевий результат систем.

```
double firstSystemResult = actualSentimentValue * firstSystemWeight;
double secondSystemResult = Math.abs(sentimentValue) * secondSystemWeight;

String finalSentiment = firstSystemResult > secondSystemResult
        ? actualCoreNLPSentiment
        : actualVaderSentiment;
```

Рисунок 3.2.4 – Застосування адаптивного параметру ваги

Результати практичного застосування цього підходу можна побачити нижче (таблиця 4).

Таблиця 4

Застосування підходу адаптивного поєднання моделей

Назва системи	**Використані тестові корпуси**		
	Відгуки на авіакомпанії (14 000+)	**Відгуки на товари (18 000+)**	**Повідомлення в Twitter (27 000+)**
Система 1 (Vader)	**53.93%**	**65.65%**	**62.99%**
Система 2 (CoreNLP)	**42.84%**	**53.05%**	**53.08%**
Адаптивне поєднання	**69.33%**	**78.79%**	**75.06%**

Найбільш дієвим виявився адаптивний підхід при поєднанні моделей, який в середньому дає приріст в 13% та більше відсотків на тестових корпусах, якщо брати до уваги якість попередньо визначених оцінок, результат може мати фактично більші показники, що будуть перевищувати 80%, що є оптимальним.

3.3 Висновки аналізу скомпонованих систем

Провівши аналіз застосування різних підходів компонування методів для визначення тональності тексту, можна зауважити приріст в точності визначення настроїв вхідного тексту. Отриманий результат значно відрізняється в залежності від якості вхідних даних. Приводом погіршення точності згідно проаналізованих тестових корпусів, можна вважати саме не правильно визначені тональності у великій кількості випадків або можливого подвійного інтерпритування контексту, в якому деякі представлені текстові дані можуть мати більш ніж одне значення. Також на точність може впливати використання людиною іронії в контексті, що для такого роду систем є значною проблемою, через важкість розпізнавання істинного сенсу закладеного в текст.

ВИСНОВКИ

В процесі виконання магістерської роботи, що включала в себе дослідження пов’язані з темою роботи, а саме область обробки природньої мови, аналіз існуючих алгоритмів та методів вилучення смислової інформації з тексту на основі методів NLP та було досягнуто наступних результатів:

1. Досліджено головні критерії, аспекти та проблематики, що присутні при розробці нових методик обробки природньої мови.

2. Проаналізовано методи попередньої обробки тексту та підходи до класифікації вхідних текстових даних.

3. Розглянуто існуючі програмні засоби та фреймворки, що реалізують алгоритми та моделі наведені в роботі. Створено порівняльну таблицю, що відображає функціональні можливості цих засобів та фреймворків.

4. Проаналізовано можливість об’єднання методик та моделей для класифікації тональності та настрою вхідного тексту для підвищення точності при виконанні задач визначення настроїв вхідного тексту.

5. Створено програмний застосунок, що реалізує підходи об’єднання моделей для підвищення точності в класифікації настрою текстів на реальних корпусах даних.

6. Проведено порівняльний аналіз отриманих результатів при застосуванні програмного застосунку для визначення найбільш оптимального підходу, що має найбільший вплив на точність при об’єднанні таких моделей.

ПЕРЕЛІК ПОСИЛАНЬ

1. . NLP in 2023: An Overview of the Latest Advancements in Natural Language Processing (2023). TechSmartFuture. [Електронний ресурс] TechSmartFuture – Режим доступу до ресурсу: https://www.techsmartfuture.com/nlp-advancements-2023/.
2. The State of the Art of Natural Language Processing—A Systematic Automated Review of NLP. (2023). Literature Using NLP Techniques. [Електронний ресурс] Data Intelligence, MIT Press – Режим доступу до ресурсу: https://direct.mit.edu/dint/article/5/3/707/115133/The-State-of-the-Art-of-Natural-Language.
3. Guntamukkala Gopi Krishna. Multilingual NLP [Електронний ресурс] / Guntamukkala Gopi Krishna // Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP). – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/371964082_Multilingual_NLP.
4. Drazen Draskovic. https://www.mdpi.com/2227-7390/10/18/3236#B11-mathematics-10-03236 [Електронний ресурс] / Drazen Draskovic, Darinka Zecevic, andBosko Nikolic // MDPI. – 2022. – Режим доступу до ресурсу: https://www.mdpi.com/2227-7390/10/18/3236#B11-mathematics-10-03236.
5. Jurafsky D. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition / D. Jurafsky, J. H. Martin., 2023. – 628 с. – (Third Edition).
6. Synthesizing Human Gaze Feedback for Improved NLP Performance [Електронний ресурс] / Yaman Kumar Singla, Varun Khurana, Nora Hollenstein та ін.] // ResearchGate. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/368473881_Synthesizing_Human_Gaze_Feedback_for_Improved_NLP_Performance.
7. A multi-semantic passing framework for semi-supervised long text classification [Електронний ресурс] / Wei Ai, Ze Wang, Hongen Shao, Tao Meng // ResearchGate. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/369688727_A_multi-semantic_passing_framework_for_semi-supervised_long_text_classification.
8. Parts of Speech Tagging: Rule-Based [Електронний ресурс] / Bao Pham // ResearchGate. – 2020. – Режим доступу до ресурсу: https://www.researchgate.net/publication/339972799_Parts_of_Speech_Tagging_Rule-Based.
9. Vasiliev Y. Natural Language Processing Using Python / Yuli Vasiliev., 2020. – 189 с.
10. Andreas Müller. Introduction to Machine Learning with Python: A Guide for Data Scientists / Andreas Müller, Sarah Guido., 2016. – 394 с. – (O'Reilly).

11. Steven Bird. Natural Language Processing with Python / Steven Bird, Ewan Klein, Edward Loper., 2009. – 504 с. – (O'Reilly).

12. Natural Language Processing: Python and NLTK / Nitin Hardeniya, Jacob Perkins, Deepti Chopra та ін.]., 2016. – 687 с. – (Packt Publishing Ltd).

13. AllenNLP Library [Електронний ресурс] – Режим доступу до ресурсу: https://allenai.org/allennlp/software/allennlp-library.

14. Patrick Muriuki. COMPARISON OF NLTK AND SPACY LANGUAGE PROCESSING LIBRARIES [Електронний ресурс] / Patrick Muriuki. – 2022. – Режим доступу до ресурсу: https://www.researchgate.net/publication/375632138_COMPARISON_OF_NLTK_AND_SPACY_LANGUAGE_PROCESSING_LIBRARIES.

15. Raymond Lee. Natural Language Processing : A Textbook with Python Implementation / Raymond Lee., 2023. – 399 с. – (Springer NatureISBN).

16. Roman Sydorenko. Модель та засоби збирання та оброблення даних з використанням машинного навчання [Електронний ресурс] / Roman Sydorenko, Vasyl Teslyuk // Scientific Bulletin of UNFU. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/372016826_Model_ta_zasobi_zbiranna_ta_obroblenna_danih_z_vikoristannam_masinnogo_navcanna.

17. Наталія Лукова-Чуйко. УДОСКОНАЛЕННЯ МЕТОДУ ВИЯВЛЕННЯ НЕПРАВДИВОЇ ІНФОРМАЦІЇ ЗА ДОПОМОГОЮ БАЙЕСОВСКОГО КЛАСИФІКАТОРА [Електронний ресурс] / Наталія Лукова-Чуйко, Tetiana Laptieva // Ukrainian Scientific Journal of Information Security. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/370822543_UDOSKONALENNA_METODU_VIAVLENNA_NEPRAVDIVOI_INFORMACII_ZA_DOPOMOGOU_BAJESOVSKOGO_KLASIFIKATORA.

18. С.В. Знахур. Прогнозування пунктуації тексту на основі моделі BERT [Електронний ресурс] / С.В. Знахур, Л.В. Знахур. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/340457552_Prognozuvanna_punktuacii_tekstu_na_osnovi_modeli_BERT.

19. Botir Elov. The pipeline processing of NLP [Електронний ресурс] / Botir Elov, Shahlo Khamroeva, Zilola Khusainova // E3S Web of Conferences. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/373072593_The_pipeline_processing_of_NLP.

20. Johannes Bjerva. The Role of Typological Feature Prediction in NLP and Linguistics [Електронний ресурс] / Johannes Bjerva // Computational Linguistics. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/375795392_The_Role_of_Typological_Feature_Prediction_in_NLP_and_Linguistics.

21. Sangavi N. NLP BASED TEXT SUMMARIZATION USING BART MODEL [Електронний ресурс] / Sangavi N, Umamaheswari M, Subasri V // INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/374558913_NLP_BASED_TEXT_SUMMARIZATION_USING_BART_MODEL.

22. Narendra Patwardhan. Transformers in the Real World: A Survey on NLP Applications [Електронний ресурс] / Narendra Patwardhan, Stefano Marrone, Carlo Sansone. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/370094802_Transformers_in_the_Real_World_A_Survey_on_NLP_Applications.

23. Dhawal Khem. An Overview of Context Capturing Techniques in NLP [Електронний ресурс] / Dhawal Khem, Shaileshkumar Panchal, Chetan Bhatt // International Journal on Recent and Innovation Trends in Computing and Communication. – 2023. – Режим доступу до ресурсу: https://www.researchgate.net/publication/370826816_An_Overview_of_Context_Capturing_Techniques_in_NLP.

24. Syntactic annotations for the Google books NGram corpus [Електронний ресурс] / Lin, Y., J.-B. Michel, E. Aiden Lieberman та ін.] // ACL. – 2012. – Режим доступу до ресурсу: https://aclanthology.org/P12-3029/.

25. The Corpus of Contemporary American English (COCA): One billion words, 1990-2019 [Електронний ресурс] Davies, M – 2020. – Режим доступу до ресурсу: https://www.english-corpora.org/coca/

26. The Flair NLP Framework [Електронний ресурс] // Humboldt-Universität zu Berlin. – 2018. – Режим доступу до ресурсу: https://www.informatik.hu-berlin.de/en/forschung-en/gebiete/ml-en/Flair.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО - КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерськаробота

«РОЗРОБКА МЕТОДИКИ ВИЛУЧЕННЯ СМИСЛОВОЇ ІНФОРМАЦІЇ З ТЕКСТУ НА ОСНОВІ МЕТОДІВ NLP

Виконав: студент групи ПДМ-61 Музика Владислав Павлович

Керівник: к.т.н., доц., доцент кафедри ІТ Трінтіна Наталія Альбертівна

Київ - 2024

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення ефективності вилучення смислової інформації з тексту за рахунок створення методики на основі методів NLP.

Об'єкт дослідження: вилучення смислової інформації з тексту.

Предмет дослідження: методи вилучення смислової інформації з тексту.

ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ NLP БІБЛІОТЕК

Функція	spaCy	NLTK	AllenNLP	Stanza	flair	Transformers	Stanford NLP
Кількість мов	50+	10+ (залежить від реалізації)	1 (англійська)	80+ (включаючи українську)	270+ (включаючи українську)	100+ (включаючи українську)	**50+ (включаючи українську)**
Токенізація	✓	✓	✓	✓	✓	✓	✓
Визначення частин мови (POS)	✓	✓	✓	✓	✓	✓	✓
Визначення іменованих сутностей	✓	✓	✓	✓	✓	✓	✓
Синтаксичний аналіз залежностей	✓	✓	✓	✓	✓	✓	✓
Сегментація речень	✓	✓	✓	✓	✓	✓	✓
Векторне представлення слів	✓	✗	✓	✗	✓	✓	✗
Визначення іменованих сутностей	✓	Обмежено	✓	Обмежено	✓	✓	✗
Правила відповідності	✓	✓	✗	✗	✓	✓	✗
Машинне навчання	✓ (опціонально)	✓	✓	✓	✓	✓	✓
Попередньо навчені моделі	✓	✓	✓	✓	✓	✓	✓
Підтримка багатомовності	✓	✓	Обмежено	✓	✓	✓	✓
Аналіз настрою	✓	✓	✓	✗	✓	✓	✓
Вирішення кореференції	✓	✓	Обмежено	✓	✗	✓	✓
Сумаризація тексту	Обмежено	✓	✗	✗	✗	✓	✗
Синтаксичні дерева	✓	✓	✓	✓	✗	✓	✓
Власні конвеєри	✓	✓	✓	✓	✓	✓	✗
Підтримка активного навчання	✓	✗	✗	✗	✗	✓	✗

3

АЛГОРИТМ РОБОТИ ПРОГРАМИ

4

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Назва системи	Використані тестові корпуси		
	Відгуки на авіакомпанії (14 000+)	Відгуки на товари (18 000+)	Повідомлення в Twitter (27 000+)
Система 1 (Vader)	53.93%	65.65%	62.99%
Система 2 (CoreNLP)	42.84%	53.05%	53.08%
Метод голосування	63.63%	71.30%	70.22%
Зважений середній результат	66.53%	75.13%	74.79%
Адаптивне поєднання	69.33%	78.79%	75.06%

5

ВИСНОВКИ

1. Досліджено головні критерії, аспекти та проблематики присутні при розробці нових NLP моделей.
2. Проаналізовано існуючі методи та алгоритми для обробки природньої мови.
3. Створено порівняльну характеристику найпопулярніших фреймворків, що реалізують проаналізовані алгоритми та методи в роботі.
4. Застосовано підхід N-грам для розширення базового функціоналу NLP бібліотеки в Java.
5. Розроблено застосунок для визначення мови на основі методів обробки природньої мови.

6

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Музика В.П. Фундаментальна обробка тексту при використанні NLP. // НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ «ПРОБЛЕМИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ» – Київ: ДУТ, 2023. – подано до друку

2. Музика В.П. Етичні аспекти NLP. Конфіденційність , асоціації та вплив на суспільство. // І ВСЕУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ «ТЕХНОЛОГІЧНІ ГОРИЗОНТИ: ДОСЛІДЖЕННЯ ТА ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ТЕХНОЛОГІЧНОГО ПРОГРЕСУ УКРАЇНИ І СВІТУ» – Київ: ДУТ, 2023. – подано до друку

7

ДЯКУЮ ЗА УВАГУ!

8

ДОДАТОК А

ЛІСТИНГ

```
package org.example;

import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.util.List;

public class CSVParserUtils {
    private CSVParser parser;

    public CSVParserUtils() {
        parser = initParser(parser);
    }

    public List<CSVRecord> getRecords() {
        return parser.getRecords();
    }

    private CSVParser initParser(CSVParser parser) {
        try {
            return CSVParser.parse(new File("src/main/resources/Aero-port_reviews100.csv"),
Charset.defaultCharset(), CSVFormat.DEFAULT);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }


}



package org.example;

import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.PropertiesUtils;

import java.util.Properties;

public class StanfordNLPPipeline {
    private static Properties properties =
PropertiesUtils.asProperties("annotators","tokenize, pos, parse, sentiment");
    private static StanfordCoreNLP stanfordCoreNLP;

    public static StanfordCoreNLP getPipeline() {
        if (stanfordCoreNLP == null) {
            stanfordCoreNLP = new StanfordCoreNLP(properties);
        }
        return stanfordCoreNLP;
    }
}



package org.example;

import com.vader.sentiment.analyzer.SentimentAnalyzer;
import edu.stanford.nlp.neural.rnn.RNNCoreAnnotations;
import edu.stanford.nlp.pipeline.CoreDocument;
import edu.stanford.nlp.pipeline.CoreSentence;
```

```
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.trees.Tree;
import edu.stanford.nlp.util.PropertiesUtils;
import org.apache.commons.csv.CSVRecord;
import org.ejml.simple.SimpleMatrix;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;

public class CoreNLPSentimentAnalyzer {
    private StanfordCoreNLP stanfordCoreNLP = StanfordNLPPipeline.getPipeline();
    private final CSVParserUtils parser = new CSVParserUtils();
    private static final AtomicInteger guessCounter = new AtomicInteger(0);

    public void coreNLPAnalysis() {
        List<CSVRecord> records = parser.getRecords();

        records.parallelStream().forEach(record -> {
            Map<String, Double> sentimentValues = new HashMap<>();
            sentimentValues.put("negative", 0d);
            sentimentValues.put("neutral", 0d);
            sentimentValues.put("positive", 0d);

            CoreDocument coreDocument = new CoreDocument(record.get(1));
            stanfordCoreNLP.annotate(coreDocument);
            List<CoreSentence> sentences = coreDocument.sentences();

            for (CoreSentence sentence : sentences) {
                Tree tree = sentence.sentimentTree();
                SimpleMatrix predictions = RNNCoreAnnotations.getPredictions(tree);
                sentimentValues.put("negative", sentimentValues.get("negative") +
predictions.get(0) + predictions.get(1));
                sentimentValues.put("neutral", sentimentValues.get("neutral") +
predictions.get(2));
                sentimentValues.put("positive", sentimentValues.get("positive") +
predictions.get(3) + predictions.get(4));
            }
            sentimentValues.replaceAll((key, sentimentValue) -> sentimentValue /
sentences.size());

            Double actualSentimentValue = Collections.max(sentimentValues.values());
            String actualSentiment = sentimentValues.entrySet()
                    .stream().filter(i -> actualSentimentValue.equals(i.getValue()))
                    .map(Map.Entry::getKey).findFirst().get();


            if (record.get(0).equals(actualSentiment)) {
                guessCounter.incrementAndGet();
            } else {//incorrect values output
                printIncorrectValues(record, sentimentValues, actualSentiment);
            }
        });

        StanfordCoreNLP.clearAnnotatorPool();
        System.out.printf("%s%%", ((double) guessCounter.get() / records.size()) * 100);
    }

    private static void printIncorrectValues(CSVRecord record, Map<String, Double>
sentimentValues, String actualSentiment) {
        System.out.printf("%s \n (%s -> %s (actual)) | %s \n %s - vader \n\n",
sentimentValues,
                record.get(0), actualSentiment,
                record.get(1),
                SentimentAnalyzer.getScoresFor(record.get(1)));
    }
}
```

```java
package org.example;

import com.vader.sentiment.analyzer.SentimentAnalyzer;
import com.vader.sentiment.analyzer.SentimentPolarities;
import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.semgraph.SemanticGraph;
import edu.stanford.nlp.semgraph.SemanticGraphCoreAnnotations;
import edu.stanford.nlp.util.CoreMap;
import edu.stanford.nlp.util.PropertiesUtils;
import org.apache.commons.csv.CSVRecord;

import java.util.Collections;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicInteger;

public class VaderSentimentAnalyzer {
    private final CSVParserUtils parser = new CSVParserUtils();
    private static final AtomicInteger guessCounter = new AtomicInteger(0);

    public static double analyzeInput(String sentence) {
        SentimentPolarities scoresFor = SentimentAnalyzer.getScoresFor(sentence);
        return scoresFor.getCompoundPolarity();
    }

    public void vaderAnalysis() {
        List<CSVRecord> records = parser.getRecords();
        records.forEach(record -> {
            double sentimentValue = VaderSentimentAnalyzer.analyzeInput(record.get(1));
            String actualSentiment = sentimentValue < -0.05 ? "negative"
                    : sentimentValue > 0.05 ? "positive" : "neutral";
            if (record.get(0).equals(actualSentiment)) {
                guessCounter.incrementAndGet();
            } else {//incorrect values output
                printIncorrectValues(record, actualSentiment);
            }
        });
        System.out.printf("%s%%", ((double) guessCounter.get() / records.size()) * 100);
    }

    private static void printIncorrectValues(CSVRecord record, String actualSentiment) {
        System.out.printf("(%s -> %s (actual)) | %s \n %s - vader \n\n",
                record.get(0),
                actualSentiment,
                record.get(1),
                SentimentAnalyzer.getScoresFor(record.get(1)));
    }
}
```

```java
package org.example;

import com.vader.sentiment.analyzer.SentimentAnalyzer;
import edu.stanford.nlp.neural.rnn.RNNCoreAnnotations;
import edu.stanford.nlp.pipeline.CoreDocument;
import edu.stanford.nlp.pipeline.CoreSentence;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.trees.Tree;
import org.apache.commons.csv.CSVRecord;
import org.ejml.simple.SimpleMatrix;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;

public class GeneralSentimentAnalyzer {
    private StanfordCoreNLP stanfordCoreNLP = StanfordNLPPipeline.getPipeline();
    private final CSVParserUtils parser = new CSVParserUtils();
    private static final AtomicInteger guessCounter = new AtomicInteger(0);
```

```
    private static final double firstSystemWeight = 1;
    private static final double secondSystemWeight = 1.3;

    public void adaptiveAnalysis() {
        List<CSVRecord> records = parser.getRecords();

        records.parallelStream().forEach(record -> {
            Map<String, Double> sentimentValues = new HashMap<>();
            sentimentValues.put("negative", 0d);
            sentimentValues.put("neutral", 0d);
            sentimentValues.put("positive", 0d);

            CoreDocument coreDocument = new CoreDocument(record.get(1));
            stanfordCoreNLP.annotate(coreDocument);

            List<CoreSentence> sentences = coreDocument.sentences();

            for (CoreSentence sentence : sentences) {
                Tree tree = sentence.sentimentTree();
                SimpleMatrix predictions = RNNCoreAnnotations.getPredictions(tree);
                sentimentValues.put("negative", sentimentValues.get("negative") +
predictions.get(0) + predictions.get(1));
                sentimentValues.put("neutral", sentimentValues.get("neutral") +
predictions.get(2));
                sentimentValues.put("positive", sentimentValues.get("positive") +
predictions.get(3) + predictions.get(4));
            }
            sentimentValues.replaceAll((key, sentimentValue) -> key.equals("neutral")
                    ? (sentimentValue / sentences.size())
                    : (sentimentValue / sentences.size()) * 1.2);

            Double actualSentimentValue = Collections.max(sentimentValues.values());
            String actualCoreNLPSentiment = sentimentValues.entrySet().stream()
                    .filter(i -> actualSentimentValue.equals(i.getValue()))
                    .map(Map.Entry::getKey).findFirst().get();


            double sentimentValue = VaderSentimentAnalyzer.analyzeInput(record.get(1));
            String actualVaderSentiment = sentimentValue < -0.05
                    ? "negative" : sentimentValue > 0.05
                    ? "positive" : "neutral";

            sentimentValue = normalizeValue(sentimentValue, actualVaderSentiment);

            double firstSystemResult = actualSentimentValue * firstSystemWeight;
            double secondSystemResult = Math.abs(sentimentValue) * secondSystemWeight;

            String finalSentiment = firstSystemResult > secondSystemResult
                    ? actualCoreNLPSentiment
                    : actualVaderSentiment;

            if (record.get(0).equals(finalSentiment)) {
                guessCounter.incrementAndGet();
            } else {//incorrect values output
                printIncorrectValues(record, finalSentiment, actualSentimentValue,
sentimentValues, secondSystemResult, firstSystemResult);
            }
        });

        StanfordCoreNLP.clearAnnotatorPool();
        System.out.printf("%s%%", ((double) guessCounter.get() / records.size()) * 100);
    }

    public void votingAnalysis() {
        List<CSVRecord> records = parser.getRecords();

        records.parallelStream().forEach(record -> {
            Map<String, Double> sentimentValues = new HashMap<>();
            sentimentValues.put("negative", 0d);
            sentimentValues.put("neutral", 0d);
```

```
            sentimentValues.put("positive", 0d);

            CoreDocument coreDocument = new CoreDocument(record.get(1));
            stanfordCoreNLP.annotate(coreDocument);

            List<CoreSentence> sentences = coreDocument.sentences();

            for (CoreSentence sentence : sentences) {
                Tree tree = sentence.sentimentTree();
                SimpleMatrix predictions = RNNCoreAnnotations.getPredictions(tree);
                sentimentValues.put("negative", sentimentValues.get("negative") +
predictions.get(0) + predictions.get(1));
                sentimentValues.put("neutral", sentimentValues.get("neutral") +
predictions.get(2));
                sentimentValues.put("positive", sentimentValues.get("positive") +
predictions.get(3) + predictions.get(4));
            }

            Double actualSentimentValue = Collections.max(sentimentValues.values());
            String actualCoreNLPSentiment = sentimentValues.entrySet().stream()
                    .filter(i -> actualSentimentValue.equals(i.getValue()))
                    .map(Map.Entry::getKey).findFirst().get();


            double sentimentValue = VaderSentimentAnalyzer.analyzeInput(record.get(1));
            String actualVaderSentiment = sentimentValue < -0.05
                    ? "negative" : sentimentValue > 0.05
                    ? "positive" : "neutral";

            sentimentValue = normalizeValue(sentimentValue, actualVaderSentiment);

            Map<String, Integer> sentimentVotes = new HashMap<>();
            sentimentVotes.put("negative", -1);
            sentimentVotes.put("neutral", 0);
            sentimentVotes.put("positive", 1);

            String finalSentiment = sentimentVotes.get(actualCoreNLPSentiment) +
sentimentVotes.get(actualVaderSentiment) == 0
                    ? "neutral" : sentimentVotes.get(actualCoreNLPSentiment) +
sentimentVotes.get(actualVaderSentiment) < 0
                    ? "negative" : "positive";

            if (record.get(0).equals(finalSentiment)) {
                guessCounter.incrementAndGet();
            } else {//incorrect values output
                printIncorrectValues(record, finalSentiment, actualSentimentValue,
sentimentValues, actualSentimentValue, sentimentValue);
            }
        });

        StanfordCoreNLP.clearAnnotatorPool();
        System.out.printf("%s%%", ((double) guessCounter.get() / records.size()) * 100);
    }

    public void weightedAverageAnalysis() {
        List<CSVRecord> records = parser.getRecords();

        records.parallelStream().forEach(record -> {
            Map<String, Double> sentimentValues = new HashMap<>();
            sentimentValues.put("negative", 0d);
            sentimentValues.put("neutral", 0d);
            sentimentValues.put("positive", 0d);

            CoreDocument coreDocument = new CoreDocument(record.get(1));
            stanfordCoreNLP.annotate(coreDocument);

            List<CoreSentence> sentences = coreDocument.sentences();

            for (CoreSentence sentence : sentences) {
                Tree tree = sentence.sentimentTree();
```

```
                SimpleMatrix predictions = RNNCoreAnnotations.getPredictions(tree);
                sentimentValues.put("negative", sentimentValues.get("negative") +
predictions.get(0) + predictions.get(1));
                sentimentValues.put("neutral", sentimentValues.get("neutral") +
predictions.get(2));
                sentimentValues.put("positive", sentimentValues.get("positive") +
predictions.get(3) + predictions.get(4));
            }

            Double actualSentimentValue = Collections.max(sentimentValues.values());
            String actualCoreNLPSentiment = sentimentValues.entrySet().stream()
                    .filter(i -> actualSentimentValue.equals(i.getValue()))
                    .map(Map.Entry::getKey).findFirst().get();


            double sentimentValue = VaderSentimentAnalyzer.analyzeInput(record.get(1));
            String actualVaderSentiment = sentimentValue < -0.05
                    ? "negative" : sentimentValue > 0.05
                    ? "positive" : "neutral";

            sentimentValue = normalizeValue(sentimentValue, actualVaderSentiment);

            double firstSystemResult = actualSentimentValue * firstSystemWeight;
            double secondSystemResult = Math.abs(sentimentValue) * secondSystemWeight;

            double weightedAverageValue = (firstSystemResult + secondSystemResult) /
(firstSystemWeight * secondSystemWeight);

            String finalSentiment = weightedAverageValue == 0 ? "neutral"
                    : weightedAverageValue < 0 ? "negative" : "positive";

            if (record.get(0).equals(finalSentiment)) {
                guessCounter.incrementAndGet();
            } else {//incorrect values output
                printIncorrectValues(record, finalSentiment, actualSentimentValue,
sentimentValues, secondSystemResult, firstSystemResult);
            }
        });

        StanfordCoreNLP.clearAnnotatorPool();
        System.out.printf("%s%%", ((double) guessCounter.get() / records.size()) * 100);
    }

    private static void printIncorrectValues(CSVRecord record, String finalSentiment,
Double actualSentimentValue, Map<String, Double> sentimentValues, double
secondSystemResult, double firstSystemResult) {
        System.out.printf("(%s -> %s (actual)) | %s \nVader:%s vs CoreNLP:%s(%s) \n 1)%s vs
2)%s \n\n",
                record.get(0), finalSentiment,
                record.get(1),
                SentimentAnalyzer.getScoresFor(record.get(1)).getCompoundPolarity(),
                actualSentimentValue,
                sentimentValues.values(),
                secondSystemResult,
                firstSystemResult);
    }

    private static double normalizeValue(double sentimentValue, String
actualVaderSentiment) {
        sentimentValue = actualVaderSentiment.equals("neutral")
                ? (1 - (Math.abs(sentimentValue) / 0.05))
                : (Math.abs(sentimentValue) / 0.06) / (1 / 0.06);
        return sentimentValue;
    }

}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>nlp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <repositories>
        <repository>
            <id>jitpack.io</id>
            <url>https://jitpack.io</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>edu.stanford.nlp</groupId>
            <artifactId>stanford-corenlp</artifactId>
            <version>4.5.5</version>
        </dependency>
        <dependency>
            <groupId>edu.stanford.nlp</groupId>
            <artifactId>stanford-corenlp</artifactId>
            <version>4.5.5</version>
            <classifier>models</classifier>
        </dependency>
        <dependency>
            <groupId>com.github.apanimesh061</groupId>
            <artifactId>VaderSentimentJava</artifactId>
            <version>v1.1.1</version>
        </dependency>
        <dependency>
            <groupId>com.github.pemistahl</groupId>
            <artifactId>lingua</artifactId>
            <version>1.2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-csv</artifactId>
            <version>1.10.0</version>
        </dependency>
    </dependencies>

</project>
```