Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Курсова робота

освітній ступінь – магістр

на тему: «**ПОШУК ТА ВИОКРЕМЛЕННЯ ВИЗНАЧЕНЬ З НАУКОВИХ ТЕКСТІВ**»

Виконав: студент 1-го року навчання,

Освітньої програми «Прикладна математика», 113

Мироненко Роман Олексійович

Керівник: Глибовець А. М.,

доктор технічних наук

Рецензент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена

з оцінкою _____

Секретар ЕК _____

«____» _____ 20____ р.

Київ – 2023

Зміст

Вступ

У науковому світі величезна кількість досліджень, статей та наукових публікацій. Ця нескінченна кількість текстових джерел ставить перед нами виклик - зібрати, організувати та ефективно використати цю велику кількість інформації.

Пошук та визначення термінів у наукових текстах є актуальною та важливою задачею, яка має великий потенціал для розвитку наукової спільноти. Однак, пошук та визначення термінів у наукових текстах стикається зі значними викликами. Ці виклики включають: амбігвітність та неоднозначність термінів, які можуть мати різні значення в різних контекстах або в різних наукових дисциплінах; великий обсяг наукових текстів, що ускладнює ручний пошук та аналіз термінів; зміна термінології та поява нових термінів у зв'язку зі швидким розвитком наукових досліджень. Впровадження автоматизованих методів та інструментів для пошуку та визначення термінів може сприяти прискоренню процесу наукового аналізу, забезпечити точність та однорідність використовуваних термінів, а також полегшити сприйняття наукової інформації для наукових спільнот.

Одним із напрямків розв'язання цих проблем є застосування методів обробки природної мови (Natural Language Processing, NLP).Використання NLP може допомогти у розробці алгоритмів та моделей для автоматичного пошуку, виділення та визначення термінів у наукових текстах.

В такому разі, **метою** даної **роботи** буде проаналізувати сучасні методи обробки природної мови, визначити, як можна використовувати їх для пошуку та виокремлення термінів із текстів, та розробити прототип такої програми.

Об'єкт дослідження: обробка природної мови, моделі глибокого навчання, наукові тексти.

Методи дослідження: аналіз сучасних підходів для роботи з текстами, аналіз наукової літератури

Мета роботи зумовила наступне **наукове завдання**:

1. Виконати аналіз інсуючих підходів для роботи з тескстами.

2. Зробити критичний огляд методів, за допомогою яких можна виконати поставлену задачу, зокерма:
    - Обробка природної мови, NLP. Бібліотека spaCy.
    - Регулярні вирази.
    - Моделі глибокого навчання.
3. Використовуючи методи, що були наведені вище, розробити програмне забезпечення по пошуку та виокремленню термінів та їх визначень.

Робота складається з вступу, трьох розділів, висновку та списку використаних джерел.

В першому розділі, ми аналізуємо існуючі підходи для роботи з текстом.

В другому розділі, ми проводимо огляд та аналіз рішень, що дозволять шукатии та виокремлювати визначення у текстах.

Третій розділ присвячений розробці програмного забезпечення методами, що були опиисані раніше.

Наукова новизна одержаних результатів: у роботі представлено декілька можливих способів пошуку та виокремлення визначень із текстів, використовуючи засоби обробки природної мови, такі як: бібліотека spaCy для обробки приордної мови, регулярні вирази, а також прототип програми із нейронними мережами. Також, виконано огляд архітектури вищезазначних методів.

Практичне значення одержаних результатів: робота може бути використана для пошуку термінів та визначень у наукових текстах.

1. Аналіз існуючих підходів для роботи з текстом

Застосування методів **обробки природної мови (NLP)**: NLP-підходи використовуються для аналізу, розуміння та обробки тексту. Вони можуть включати методи позначення частин мови, синтаксичний аналіз, аналіз сентименту, іменоване сутності, розпізнавання зв'язків між словами тощо. Ці методи дозволяють розуміти семантичну структуру тексту і виконувати завдання, такі як класифікація, кластеризація та екстракція інформації.

Мова програмування Python надає широкий спектр інструментів і бібліотек для вирішення конкретних завдань NLP. Багато з них можна знайти в Natural Language Toolkit, або NLTK, колекції бібліотек, програм і освітніх ресурсів з відкритим кодом для створення програм NLP.

Векторне представлення слів: Цей підхід базується на перетворенні слів у числові вектори, що дозволяє використовувати методи машинного навчання для роботи з текстом. Такі моделі, як Word2Vec, GloVe та FastText, можуть використовуватися для створення векторного представлення слів, що відображає семантичні відношення між ними. Цей підхід дозволяє враховувати значення слів у контексті аналізу тексту.

Розглянемо суть методу на прикладі: Хоча досить просто знайти зв’язок між словами France і Italy, або ж між будь-якими іншими країнами, пов’язати слова із більш віддаленими зв’язками є дещо складнішим завданням. Проте, досліджуючи питання, як знайти пару до слова small так, що зв’язок між ними буде подібний до того, що існує між словами big-biggest, з’являється дуже цікава думка. Виявляється, що розв’язком поставленого завдання може стати слово, яке у векторному просторі знаходитиметься найближче до вектора X = vector(“biggest”) — vetor(“big”) + vector(“small”). Таким способом можна отримати правильний результат коли мережа є достатньо натренованою.

Глибоке навчання (deep learning):

- Рекурентні нейронні мережі (RNN): RNN можуть моделювати послідовність слів у тексті та залежності між ними. Це корисно для завдань, таких як машинний переклад, аналіз тональності або генерація тексту. RNN мають "пам'ять" про попередні слова у послідовності, що дозволяє їм краще розуміти текстовий контекст. Наприклад, в задачі машинного перекладу RNN може використовуватись для перетворення вхідного речення з однієї мови на іншу, засновуючись на попередніх словах у вхідному реченні.
- Згорткові нейронні мережі (CNN): Хоча згорткові нейронні мережі зазвичай використовуються для обробки зображень, їх також можна успішно застосовувати до обробки тексту. У випадку тексту, CNN можуть використовуватись для

виявлення локальних шаблонів та функцій у послідовності слів, таких як n-грами. Це може бути корисно для класифікації тексту, виявлення спаму або аналізу настрою.

- Трансформери: Трансформери є новітнім розробленням у галузі обробки тексту, які стали основою багатьох відомих моделей, наприклад, BERT, GPT та Transformer-XL. Вони використовують механізм уваги для моделювання залежностей між словами у тексті та розуміння семантичного контексту. Трансформери демонструють високу ефективність у багатьох завданнях, таких як машинний переклад, генерація тексту, розпізнавання мови та відповіді на запитання.

Екстракція інформації: Цей підхід використовується для виокремлення структурованої інформації з тексту. Наприклад, витягаються іменовані сутності, дати, місця або ключові факти з новинних статей або наукових документів. Екстракція інформації може використовуватися для автоматичної індексації та організації великого обсягу документів.

Комбінація моделей: Іноді комбінування декількох моделей може покращити результати роботи з текстом. Ансамблі можуть використовувати різні методи, наприклад, комбінацію Bag-of-Words з Word Embeddings або поєднання RNN з CNN. Використання ансамблей може забезпечити більш точні та надійні результати, особливо в складних задачах.

Важливо розуміти, що даний список не є вичерпним, оскільки існує багато інших методів та підходів для роботи з текстом. Однак, він дає загальне розуміння про те, які підходи існують для роботи із текстовими даними.

2. Методи для пошуку та виокремлення із тексту

2.1Обробка природної мови, NLP

Типовий конвеєр для попередньої обробки тексту складається з наступних кроків:

1. Сегментація речення.
2. Нормалізація та токенізація.
3. Позначення частини мови (POS).

4. Розпізнавання іменованих сутностей.

У більшості програм не потрібно виконувати всі кроки. Потреба в розпізнаванні іменованих об’єктів залежить від бізнес-потреб програми, тоді як POS-тегування зазвичай виконується автоматично за допомогою сучасних інструментів для покращення частини етапів нормалізації та токенізації.

2.1.1 Сегментація речення

На першому етапі конвеєра попередньої обробки текст сегментується на речення. У багатьох мовах, таких як англійська, пунктуація, особливо крапка, знаки оклику та питання, можуть використовуватися для визначення кінця речення. Однак крапка також може використовуватися в абревіатурах, наприклад Ms. або UK, і в цьому випадку крапка не означає кінець речення. У цих випадках використовується таблиця скорочень, щоб уникнути неправильної класифікації меж речень. Якщо текст містить предметно-спеціальні терміни, необхідно створити додатковий словник скорочень, щоб уникнути неприродних лексем.

2.1.2 Токенізація та нормалізація.

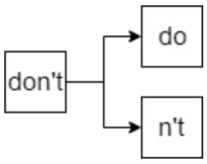

Рисунок 1

Токенізація — це поділ тексту на слова та знаки пунктуації, тобто лексеми. Як і у випадку з сегментацією речень, розділові знаки можуть бути складними. Наприклад, UK слід розглядати як одну лексему, тоді як «не» слід розділити на дві лексеми: «робити» та «не робити».

Визначення основи та лематизація є важливими частинами процесу нормалізації. Нормалізація складається з коріння та лематизації. Стемінг (stemming) - це процес в токенізації та нормалізації тексту, який використовується для зведення слів до їх основи або кореня (стема). Стем - це основна форма слова, від якої можна утворити різні словоформи шляхом додавання префіксів, суфіксів, закінчень та інших аффіксів. Лематизація (lemmatization) - це процес в токенізації та нормалізації тексту, який

полягає в зведенні слова до їх леми або базової форми, відповідно до словника або морфологічних правил мови. Лема є канонічною або словниковою формою слова, яка представляє його семантичний зміст.Під час процесу визначення основи основа слова ідентифікується шляхом видалення суфіксів, таких як –ed і -ing. Отримана основа не обов'язково є словом. Так само лематизація передбачає видалення префіксів і суфіксів з тією важливою різницею, що результат належить мові. Цей результат називається лемою. Приклади походження та лематизації можна побачити в таблиці нижче:

	Word 1	Word 2	Word 3
Original	studies	playing	best
Stem	studi	play	best
Lemma	study	play	good

Рисунок 2

Інші методи нормалізації включають: розширення абревіатур, видалення цифр і знаків пунктуації, виправлення типових граматичних помилок тощо. Більшість цих операцій можна виконати за допомогою регулярних виразів.

2.1.3 Позначення частини мови

Цей крок стосується класифікації токенів у класи частин мови (POS). POS (Part-of-Speech) - це класифікація токенів або слів в тексті на основі їх граматичної ролі або частини мови. Це важлива складова при аналізі природної мови, оскільки вказує на синтаксичну та семантичну функцію кожного слова у реченні. Класами POS є іменники, дієслова, прийменники, прислівники тощо. Лексичні категорії з прикладами

англійською мовою наведено в таблиці нижче. POS-тегування покращує лематизацію та є необхідним для розпізнавання іменованих сутностей.

Lexical category	Example
Noun	book, girl, forest, moss
Verb	play, study, write, choose
Adjective	happy, short, brown, cool
Preposition	at, about, over, on
Determiner	the, a, this, those
Conjunction	and, but, or, if
Pronoun	I, she, you, they

Рисунок 3

Існує три типи POS-тегерів: заснований на правилах, на основі статистики та глибокого навчання. Теги на основі правил покладаються на явні правила, наприклад: артикль повинен супроводжуватися іменником, для позначення токенів. Статистичні теги використовують моделі ймовірностей для позначення окремих слів або послідовностей слів. Теги на основі правил дуже точні, але також залежать від мови. Розширення тегера для підтримки інших мов вимагає багато роботи. Статистичні тегери легше створити та не залежать від мови, хоча вони жертвують точністю. Зараз використовуються гібридні підходи моделей на основі правил і статистичних моделей, хоча більшість індустрії починає повільно рухатися до рішень глибокого навчання, де моделі навчаються на попередньо позначених наборах речень. Підходи на основі гібридного та глибокого навчання покращують контекстно-залежне тегування.

2.1.4 Розпізнавання іменованої сутності

Перед тим, як іменовані сутності можуть бути визначені, необхідно провести сегментацію та позначення токенів. Сегментація (chunking) - означає розділення і позначення груп токенів. Одним з найпоширеніших видів чанків є фраза іменників (noun phrase chunk), яка складається з визначника, прикметників та іменника, наприклад, "щасливий єдиноріг". У реченні "Він знайшов щасливого єдинорога" існують два чанки: "він" та "щасливого єдинорога".

Іменовані сутності є фразами, які вказують на конкретні об'єкти, такі як особи, організації, місця розташування, дати та геополітичні одиниці. Метою кроку

розпізнавання іменованих сутностей (Named Entity Recognition, NER) є виявлення згадок іменованих сутностей у тексті.

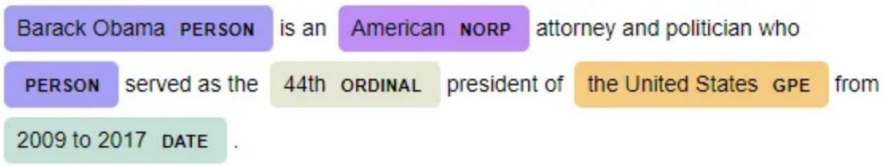

Рисунок 4

В подальшому, ми будемо використовувати бібліотеку spaCy. spaCy - це відкрита бібліотека для обробки природної мови (NLP), яка надає зручні та ефективні інструменти для виконання різноманітних завдань у сфері обробки текстів. Вона розроблена для швидкої та ефективної обробки великих обсягів текстових даних, зокрема для завдань таких як токенізація, лематизація, частиномовний аналіз, розпізнавання іменованих сутностей та зв'язків між ними.

Однією з головних переваг spaCy є її швидкодія, завдяки оптимізаціям, вбудованим у її ядро. Крім того, вона має широкий спектр мовових моделей, які охоплюють різні мови та географічні діалекти.

В цілому, spaCy є потужним інструментом для розв'язання різноманітних задач NLP, від базових операцій обробки тексту до складніших завдань аналізу мови та витягування інформації з текстових даних.

2.2 Регулярні вирази

Так як наша задача полягає в тому, щоб шукати терміни та їх визначення у текстах, можна звернутися до методу регулярних виразів, адже в нашому випадку речення, що містять певний термін та визначення, будуть мати певну структуру.

Регулярний вираз не є ні бібліотекою, ні мовою програмування. Натомість регулярний вираз (Regular Expressions або Regex) — це послідовність символів, яка визначає шаблон пошуку в будь-якому заданому тексті (рядку). Текст може складатися практично з усього, від літер до цифр, пробілів і спеціальних символів. Поки рядок відповідає певному зразку, регулярний вираз є достатньо надійним, щоб мати можливість захопити цей шаблон і повернути певну частину рядка.

Основні символи регулярних виразів, які потрібно знати, включають:

- . (крапка): Представляє будь-який один символ, крім символу нового рядка.
- * (зірочка): Позначає 0 або більше повторень попереднього символу або шаблону.
- + (плюс): Позначає 1 або більше повторень попереднього символу або шаблону.
- ? (знак питання): Позначає 0 або 1 повторення попереднього символу або шаблону.
- [] (квадратні дужки): Визначають набір символів, з яких потрібно знайти відповідність. Наприклад, [abc] знайде будь-який символ "a", "b" або "c".
- () (круглі дужки): Використовуються для групування символів або шаблонів, щоб застосовувати до них оператори, такі як *, + або ?, або для виокремлення підвиразів.
- | (вертикальна риска): Представляє альтернативу між двома варіантами. Наприклад, cat|dog знайде або слово "cat", або слово "dog".
- \ (зворотний слеш): Використовується для екранування спеціальних символів. Наприклад, \. буде сприйматися як крапка, а не як спеціальний символ "будь-який символ".
- ^ (царина): Вказує початок рядка. Наприклад, ^Hello знайде слово "Hello" тільки на початку рядка.
- $ (долар): Вказує кінець рядка. Наприклад, world$ знайде слово "world" тільки в кінці рядка.

Наведемо приклад роботи регулярних виразів для білшої ясності: є фрейм даних з прізвищами людей, титулами та іменами:

full_name
Smith, Mr. John
Davis, Ms Nicole
Robinson, Mrs. Rebecca
Armstrong, Dr Sam
Downey, Mr. Robert

Рисунок 5

Розділимо їх так, щоб у кожного з них були окремі колонки:

```
шаблон_назви = "(\\w+),\\s(Mr|Ms|Mrs|Dr).?\\s(\\w+)"
```

Рисунок 6

- (\\w+) - 1 або більше символів слова (перша група захоплення, тобто прізвище)
- , - символ коми
- \\s - один пробіл
- (Mr|Ms|Mrs|Dr) - Mr, Ms, Mrs або Dr (друга група захоплення, тобто заголовок)
- .? - 0 або 1 крапка після назви
- \\s - один пробіл
- (\\w+) - 1 або більше символів слова (третя група захоплення, тобто ім’я)

Помістимо їх в окремі стовпці:

```
names$family_name = str_match(names$full_name, name_pattern)[, 2]
names$title = str_match(names$full_name, name_pattern)[, 3]
names$given_name = str_match(names$full_name, name_pattern)[, 4]
```

Рисунок 7

В результаті отримаємо:

full_name	family_name	title	given_name
Smith, Mr. John	Smith	Mr	John
Davis, Ms Nicole	Davis	Ms	Nicole
Robinson, Mrs. Rebecca	Robinson	Mrs	Rebecca
Armstrong, Dr Sam	Armstrong	Dr	Sam
Downey, Mr. Robert	Downey	Mr	Robert

Рисунок 8

З охоплених матеріалів можна зрозуміти, що Регулярні вирази - це потужний інструмент для роботи з текстовими даними, який дозволяє здійснювати пошук, вилучення та маніпуляції з рядками на основі заданих шаблонів. Регулярні вирази широко застосовуються в області обробки тексту, редакторах коду, веб-розробці, аналізі даних, валідації введення користувача та інших сферах, де потрібні потужні та гнучкі інструменти для маніпуляції з текстом.

В подальшому, ми розробимо програму, що використовуватиме регулярні вирази для пошуку термінів та визначень у текстах.

2.3 Глибоке навчання

Використання нейронних мереж у роботі з текстом відіграє ключову роль у сфері обробки природної мови (Natural Language Processing, NLP). Нейронні мережі - це математичні моделі, що наслідують роботу нейронної системи людського мозку. Вони можуть бути використані для розпізнавання та розуміння тексту, генерації тексту, класифікації текстових даних, машинного перекладу, витягування інформації та багатьох інших задач NLP.

Багато завдань НЛП можна розглядати як проблеми класифікації. Прямим прикладом є аналіз настроїв, який може бути таким же простим, як класифікація рецензій на фільми як позитивні чи негативні. Не таким простим прикладом є завдання НЛП передбачення наступного слова. Коли я хочу передбачити наступне слово після «Я люблю їсти…», це наступне слово може бути будь-яким словом у моєму корпусі. Спосіб передбачення наступного слова полягає в тому, щоб обчислити ймовірності для всіх унікальних слів у моєму корпусі, а потім підібрати слово з максимальною ймовірністю. Softmax зазвичай використовується для нормалізації ймовірностей кількох вихідних класів.

Багато завдань у області обробки природної мови (NLP) можна розглядати як проблеми класифікації, де текстові дані відносяться до певних категорій або класів. Наприклад, аналіз настроїв може бути розглянутий як завдання класифікації, де текст рецензій на фільми класифікується як позитивний чи негативний.

Для класифікації тексту, як і у випадку аналізу настроїв, нейронні мережі можуть бути навчені моделювати залежності між текстом і відповідними класами. Це досягається шляхом навчання моделі на великому наборі прикладів, де кожен приклад містить текстові дані і відповідний мітку класу. Нейронна мережа вчиться визначати патерни і закономірності в тексті, які вказують на його класифікацію. Щоб отримати

нормалізовані ймовірності для кількох вихідних класів, таких як у випадку передбачення наступного слова, зазвичай використовується функція Softmax.

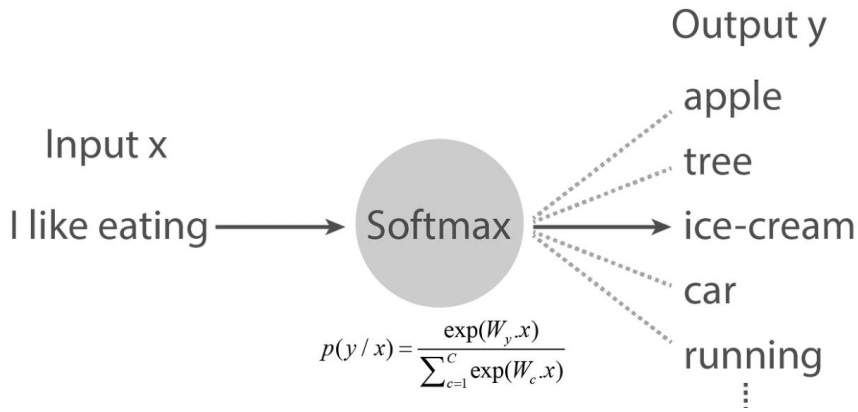

Рисунок 9

2.3.1 Нейрони

Нейрони є елементарними обчислювальними одиницями нейронної мережі. У нейроні після лінійного перетворення вхідних даних результат подається в логістичну регресію. Логістична регресія вводить нелінійність у обчислення в нейроні. Таким чином, нейрон, по суті, є двійковою одиницею логістичної регресії. Нейронна мережа складається з кількох шарів, де кожен шар додатково складається з групи нейронів. Ми можемо розглядати кожен рівень як декілька логістичних регресій, що виконуються одночасно, і результати потім передаються в іншу логістичну регресію.

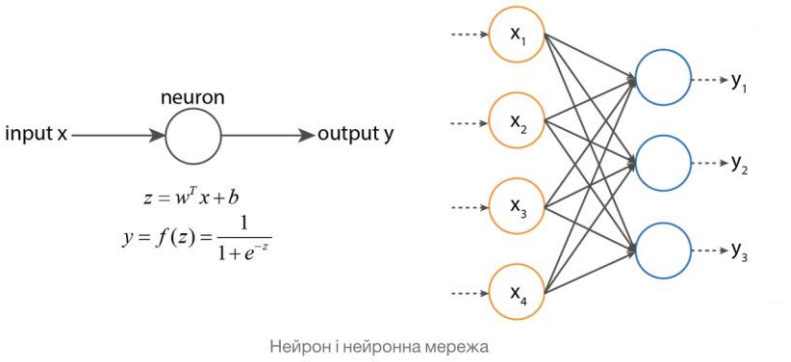

Рисунок 10

2.3.2 Повторювана нейронна мережа (RNN)

Часто в завданнях NLP ми маємо справу з послідовністю слів, якими можуть бути рецензії на фільми з неповними реченнями або добре написані дослідницькі статті. Щоб передбачити наступне слово, потрібне загальне розуміння попередньої послідовності слів. RNN — це архітектура, яка підходить для обробки послідовності

даних. По суті, на кожному кроці часу вектор слова подається в нейрон у RNN. На додаток до вектора вхідного слова на цьому часовому кроці нейрон також отримує прихований стан, який несе інформацію про попередні часові кроки. Таким чином інформація про послідовність слів передається в нейронну мережу.

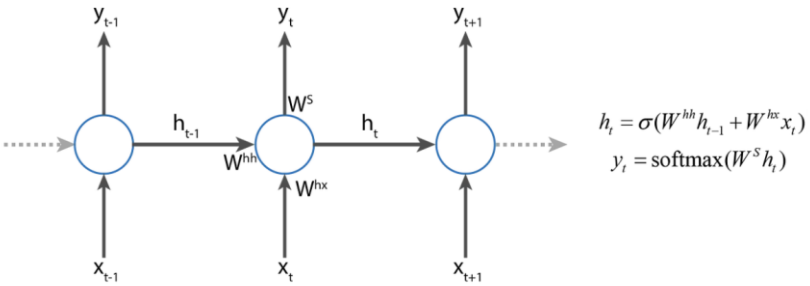

Рисунок 11

Архітектура RNN також називається vanilla RNN. Vanilla RNN страждає від проблем із зникаючим градієнтом, що спричинено довгим ланцюгом оновлення параметрів у процесі зворотного поширення. Зворотне розповсюдження спрямоване на оновлення параметрів нейронної мережі, щоб мінімізувати вихідну помилку. Однак через природу послідовного з'єднання ванільного RNN параметри подальших нейронів не можуть ефективно оновлюватися. Таким чином нейрони легко гинуть. Щоб подолати проблему зникнення градієнта у ванільній RNN, запропоновано два вдосконалених рішення архітектури RNN: блоки із закритими повторюваними блоками (GRU)[2] і довготривалі короткочасні пам'яті (LSTM).

STM - це шар, що базується на ідеї короткотермінової та довготермінової пам'яті. Він має внутрішні комірки пам'яті, які зберігають інформацію з попередніх часових кроків. LSTM використовує гейти (ворота), такі як ворота забування, ворота входу та ворота виходу, для контролю потоку інформації у комірку пам'яті. Це дозволяє LSTM "вирішувати", які частини інформації зберігати та які забувати.

GRU - це спрощена версія LSTM, яка також використовує гейти для контролю потоку інформації. GRU має два гейти - гейт зворотного зв'язку та гейт збереження. Гейт зворотного зв'язку визначає, яка частина попередньої інформації повинна бути передана на вихід, а гейт збереження визначає, яка частина нової інформації повинна бути збережена в комірці пам'яті.

Як LSTM, так і GRU допомагають уникнути проблеми зниклого градієнту, дозволяють моделі зберігати та використовувати важливу інформацію з великих послідовностей та довше зберігати залежності у даних. Обидва шари можуть бути використані в RNN для обробки послідовностей з різною довжиною та моделювання складних залежностей у даних.

2.3.3 Згорткова нейронна мережа (CNN)

RNN забезпечує потік інформації в часі, що дозволяє їй розуміти глобальну семантику послідовностей. Це досягається завдяки зворотньому зв'язку, який дозволяє нейронам RNN передавати інформацію від попередніх часових кроків до наступних.

Проте, через цей потік інформації в часі обчислення RNN можуть бути повільними порівняно з іншими архітектурами, такими як CNN (згорткова нейронна мережа). Одна з причин цього повільного процесу - залежність від попередніх нейронів для обчислення в кожному часовому кроці. Кожен наступний нейрон очікує отримати вхідні дані від попередніх нейронів, і це може призвести до затримок у обчисленнях.

В деяких випадках, коли завдання NLP не вимагає сильного фокусу на глобальній семантиці, а замість цього акцентується на локальних ознаках або локальному контексті, CNN може бути більш швидким та ефективним вибором. Згорткова нейронна мережа використовує оператори згортки, які допомагають виявляти локальні ознаки у вхідних даних, і не має такої сильної залежності від попередніх кроків, як у RNN.

CNN широко використовується в моделях комп'ютерного зору. Основна ідея CNN полягає в паралельному застосуванні кількох фільтрів до вхідних даних, де кожен фільтр виділяє певну функцію. У завданнях комп’ютерного зору ці фільтри розпізнають локальні шаблони в просторі. Подібним чином, при застосуванні до завдань NLP, ці фільтри можна навчити розпізнавати локальні закономірності з часом.

3. Пошук та виокремлення термінів у наукових текстах

3.1 Регулярні вирази

Спочатку, спробуємо вирішити поставлену задачу за допомогою регулярних виразів. Ось приклад такої програми:

```python
import re
import requests
from bs4 import BeautifulSoup

def find_definitions_from_url(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    # Видалення шапки сторінки
    header = soup.find(class_='page-header')
    if header:
        header.extract()

    text = soup.get_text()

    pattern = r'([^.\n]+) – ([^.\n]+)'

    matches = re.findall(pattern, text, re.DOTALL)
    for match in matches:
        term = match[0].strip()
        definition = match[1].strip()
        print(f'Term: {term}\nDefinition: {definition}\n')

# Приклад використання
url = 'https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%
find_definitions_from_url(url)

```

Цей код містить функцію find_definitions_from_url, яка отримує URL-адресу веб-сторінки в якості вхідного параметра. В нашому випадку це посилання на сторінку Вікіпедії, а саме – Штучний інтелект(https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82). Ось кроки, що йдуть далі:

1. Виконує HTTP-запит до вказаного URL за допомогою requests.get(url) для отримання вмісту сторінки.
2. Використовує BeautifulSoup для створення об'єкта soup, який представляє розібраний HTML-вміст сторінки. BeautifulSoup - це бібліотека для парсингу HTML і XML документів у Python.
3. Видаляє шапку сторінки, якщо вона присутня, за допомогою методу header.extract().
4. Отримує текст сторінки за допомогою soup.get_text(), який видаляє HTML-теги і повертає тільки текстовий зміст.

5. Використовує регулярний вираз r'([^.\n]+) — ([^.\n]+)' для знаходження відповідностей, де кожен знайдений відповідний вираз містить термін та його визначення, розділені тире.
6. Проходиться по знайдених відповідностях (matches) і виводить на екран кожен термін та його визначення, використовуючи print(f'Term: {term}\nDefinition: {definition}\n').

Результат, що видає програма:

```
Term: Штучний інтелект
Definition: Вікіпедія

Term: Матеріал з Вікіпедії
Definition: вільної енциклопедії

Term: Штучний інтелект (ШІ)
Definition: це галузь інформатики, яка займається розробкою інтелектуальних машин, здатних виконувати завдання, які зазвич
ай потребують людського інтелекту

Term: Машинне навчання
Definition: це тип штучного інтелекту, який передбачає навчання алгоритмів навчанню на основі вхідних даних і покращенню ї
х продуктивності з часом

Term: Обробка природної мови (NLP)
Definition: це тип ШІ, який зосереджується на взаємодії між комп’ютерами та людськими мовами

Term: Робототехніка
Definition: це сфера штучного інтелекту, яка зосереджена на проектуванні та розробці роботів, які можуть виконувати завдан
ня у фізичному світі

Term: Експертні системи
Definition: це системи штучного інтелекту, призначені для надання порад і підтримки прийняття рішень у певних сферах, таки
х як маркетинг[13], дизайн[14], медицина[15], право[16] та інженерія[17]

Term: Загальний ШІ, також відомий як сильний штучний інтелект (AGI),
Definition: це теоретична концепція створення машин, які можуть міркувати та навчатися, як люди

Term: Алгоритм перемагали всього два рази — вперше це зробив штучний інтелект Stockfish 15, вдруге
Definition: нідерландський шахіст Бенджамін Бок[47][48]
```

Як можна побачити, цей підхід дійсно шукає підходящі за структурою речення, в яких є форма “Термін - Визначення”, але результат не завжди є вірним.

3.2 spaCy

Використовуючи бібліотеку spaCy для обробки природної мови, ми розробили таку програму, для пошуку підходящих структур у тексті:

```python
import spacy
from rich import print

def find_structures_in_text(text):

    nlp = spacy.load('en_core_web_sm')
    doc = nlp(text)
    matching_structures = []

    for sentence in doc.sents:

        sentence_text = sentence.text.strip()

        if '-' in sentence_text:

            parts = sentence_text.split('-', 1)
            term = parts[0].strip()
            definition = parts[1].strip()
            matching_structures.append((term, definition, sentence))

        elif ' - ' in sentence_text:

            parts = sentence_text.split(' - ', 1)
            term = parts[0].strip()
            definition = parts[1].strip()
            matching_structures.append((term, definition, sentence))

        elif ' це ' in sentence_text:

            parts = sentence_text.split(' це ', 1)
            term = parts[0].strip()
            term = term[ term.find(",")+1: ]
            definition = parts[1].strip()
            matching_structures.append((term, definition, sentence))

        elif 'це ' in sentence_text:
            parts = sentence_text.split('це ', 1)
            term = parts[0].strip()
            definition = parts[1].strip()
            matching_structures.append((term, definition, sentence))

    return matching_structures

# Текст для пошуку підходящих структур
text = "Піфагорові трійки – це три натуральні числа a, b, та c такі, що виконується рівність a2 + b2 = c2. Іншими слова

matching_structures = find_structures_in_text(text)
highlighted_text = text

if matching_structures:

    for term, definition, sentence in matching_structures:
        sentence_text = sentence.text
        highlighted_sentence = sentence_text.replace(term, f"[bold]{term}[/bold]")
        highlighted_sentence = highlighted_sentence.replace(definition, f"[bold]{definition}[/bold]")
        highlighted_text = highlighted_text.replace(sentence_text, highlighted_sentence)

print(f"Повний текст з підсвіченими реченнями, що підходять по структурі:\n\n{highlighted_text}")
```

Даний код виконує пошук підходящих структур у заданому тексті та підсвічує їх для візуалізації. Основні кроки в коді:

1. Імпортуються необхідні бібліотеки: spacy для обробки тексту і rich для форматованого виводу.
2. Оголошується функція find_structures_in_text, яка приймає текст і шукає підходящі структури використовуючи модель en_core_web_sm з бібліотеки spacy.

3. В циклі обробляються речення тексту, аналізується їх текст і перевіряється, чи містяться в них певні шаблони структур (наприклад, присутність дефісу або фрази "це").
4. Якщо знайдено підходящу структуру, вона додається до списку matching_structures разом із терміном, визначенням та самим реченням.
5. Функція повертає список знайдених підходящих структур.
6. Задається вихідний текст text, в якому потрібно знайти підходящі структури.
7. Викликається функція find_structures_in_text для пошуку підходящих структур у тексті.
8. Ініціалізується змінна highlighted_text з початковим текстом.
9. Якщо знайдено підходящі структури: Проходиться по кожній знайденій структурі; Замінюються відповідні терміни та визначення у тексті на їх виділені (підсвічені) версії; Оновлений текст з підсвіченими реченнями зберігається у змінну highlighted_text.

Виводиться повний текст з підсвіченими реченнями, що підходять по структурі.

В даному коді ми мали такий текст на вході: Піфагорові трійки — це три натуральні числа a, b, та c такі, що виконується рівність a2 + b2 = c2. Іншими словами, Піфагорові трійки — це сторони прямокутного трикутника, якщо всі вони є цілочисельними. На мегалітичних спорудах в північній Європі є свідчення, що відомості про такі трійки були відомі до винайдення писемності. Такі трійки зазвичай записують у вигляді (a, b, c). Деякі найвідоміші приклади: (3, 4, 5) та (5, 12, 13).

В результаті, програма знайшла відповідну структуру в тексті:

```
Повний текст з підсвіченими реченнями, що підходять по структурі:

Піфагорові трійки – це три натуральні числа a, b, та c такі, що виконується рівність a2 + b2 = c2. Іншими словами,
Піфагорові трійки – це сторони прямокутного трикутника, якщо всі вони є цілочисельними. На мегалітичних спорудах в
північній Європі є свідчення, що відомості про такі трійки були відомі до винайдення писемності. Такі трійки
зазвичай записують у вигляді (a, b, c). Деякі найвідоміші приклади: (3, 4, 5) та (5, 12, 13).
```

Варто зазначити, що для рішення більш складних задач по пошуку термінів та визначень варто точніше налаштувати метод пошуку.

3.3 Глибоке начання

В даному випадку, ми розробили програму, яка використовує нейронну мережу з використанням регулярних виразів для розпізнавання відповідної структури речень у тексті.

```python
import re
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Навчальні дані
sentences = [
    "Термін1 - Визначення1",
    "Термін2 - Визначення2",
    "Термін3 - Визначення3",
    "Термін1 це Визначення1",
    "Термін2 це Визначення2",
    "Термін3 - це Визначення3",
    "Термін1 - це Визначення1"
]

# Побудова моделі нейронної мережі
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
vocab_size = len(tokenizer.word_index) + 1
max_sequence_len = max([len(sentence.split()) for sentence in sentences])

model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(vocab_size, 10, input_length=max_sequence_len-1),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(10)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(vocab_size, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Тренування моделі
input_sequences = []
for sentence in sentences:
    token_list = tokenizer.texts_to_sequences([sentence])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre')
predictors, label = input_sequences[:,:-1], input_sequences[:,-1]
label = tf.keras.utils.to_categorical(label, num_classes=vocab_size)
```

```python
model.fit(predictors, label, epochs=100, verbose=1)

# Функція для розпізнавання речення з заданою структурою
def recognize_sentence(input_text):
    input_sequence = tokenizer.texts_to_sequences([input_text])[0]
    input_sequence = pad_sequences([input_sequence], maxlen=max_sequence_len-1, padding='pre')
    predicted_index = tf.argmax(model.predict(input_sequence)[0], axis=-1)
    predicted_word = tokenizer.index_word[int(predicted_index)]
    return predicted_word

# Вхідний текст для перевірки
input_text = '''
Машинне навчання - підгалузь штучного інтелекту в галузі інформатики.
Назву «машинне навчання» (англ. machine learning) було започатковано 1959 року Артуром Семюелем.
Шту́чний інтеле́кт - це розділ комп'ютерної лінгвістики та інформатики, який швидко розвивається, і зосереджений на розробці
'''

# Розділення тексту на речення
sentences = re.split(r'(?<=[.!?])\s+', input_text)

# Пошук речень з відповідною структурою
matching_sentences = []
for sentence in sentences:
        term_definition_match = re.search(r'(.+)\s+(це|-)\s+(.+)', sentence)
        if term_definition_match:
            term = term_definition_match.group(1)
            definition = term_definition_match.group(3)
            predicted_term = recognize_sentence(term)
            predicted_definition = recognize_sentence(definition)
            if predicted_term == '-' or predicted_term == 'це' or predicted_term == '- це' or predicted_definition == '-'
                matching_sentences.append(sentence)

# Виведення знайдених речень
if matching_sentences:
    print("Знайдені речення з відповідною структурою:")
    for sentence in matching_sentences:
        print(sentence)
else:
    print("Не знайдено речень з відповідною структурою.")
```

Цей код виконує наступні дії:

1. Визначає навчальні дані, які складаються з рядків, що містять терміни та визначення.
2. Використовує Tokenizer з бібліотеки Keras для побудови словника слів на основі навчальних даних і перетворення речень на послідовності чисел.
3. Побудовує модель нейронної мережі з використанням вбудовування слів, двостороннього GRU шару та двох повно з'єднаних шарів. Модель компілюється з використанням категоріальної кросс-ентропійної функції втрат та оптимізатора Adam.
4. Генерує послідовності для тренування моделі, розбиваючи речення на N-грами, де N змінюється від 1 до довжини речення.
5. Виконує тренування моделі з використанням побудованих послідовностей.
6. Визначає функцію recognize_sentence, яка приймає текстовий ввід і передбачає останнє слово в реченні з використанням навченої моделі.
7. Вхідний текст розділяється на речення за допомогою регулярних виразів.

8. Застосовується пошук речень з відповідною структурою, де термін і визначення розділені дефісом або словом "це". Застосовується функція recognize_sentence для передбачення слів і порівнюється зі знаками дефісу або словом "це".
9. Виводиться знайдені речення з відповідною структурою або повідомлення, якщо речень не знайдено.

В результаті, ми отримуємо:

```
Машинне навчання - підгалузь штучного інтелекту в галузі інформатики.
Шту́чний інтеле́кт - це розділ комп'ютерної лінгвістики та інформатики, який швидко розвивається, і зосереджений на розробці
інтелектуальних машин, здатних виконувати завдання, які зазвичай потребують людського інтелекту.
```

Рисунок 12

Висновки

У цій роботі ми розглянули засоби для роботи із текстовими даними. Для цього було запропоновано три методи для рішення поставленої задачі: регулярні вирази, робота із бібліотекою spaCy для обробки природної мови та метод глибокого навчання. Ми також надали огляд моделей та того, як вони працюють.

Розроблено три прототипи програм, які базуються на вищеперерахованих методах, що здійснюють пошук та виокремлення термінів і визначень із тексту. Варто зазначити, що на цих прототипах може відбуватись подальша робота по удосконаленю та більш точного пошуку підходящих структур, адже наше дослідження відбувалось з акцентом саме на структуру речень та того, як визначення подаються у різних текстах.

Між трьома методами, які ми розглядали, неможливо однозначно визначити, який є найефективнішим без проведення додаткових тестів та оцінки продуктивності. Ефективність методів може залежати від різних факторів, таких як розмір даних, складність шаблонів, обсяг обробки та інші. Однак, можна надати загальну уяву про ефективність цих методів:

1. Метод, який використовує модуль **re** у першому коді, використовує регулярні вирази для знаходження термінів та визначень. Регулярні вирази можуть бути швидкими та ефективними для простих шаблонів, але можуть стати менш ефективними для складніших та багатошарових структур. Крім того, вони можуть вимагати пильного створення та налагодження правильного регулярного виразу для виявлення бажаних структур.
2. Метод, який використовує модуль spaCy у другому коді, має прямий доступ до лінгвістичних моделей та функціоналу NLP. spaCy використовується для розбору тексту, виявлення структур та визначення термінів та їх визначень. Використання готової NLP-бібліотеки може забезпечити швидку та ефективну обробку тексту.
3. Метод, який використовує модуль **tensorflow** у третьому коді, базується на нейронних мережах та обробці послідовностей. Застосування моделей нейронних мереж може бути ефективним у визначенні послідовностей, таких як

терміни та їх визначення, однак, тренування та передбачення з використанням нейронних мереж можуть бути витратними за ресурсами та часом.

Список використаних джерел

1. What is natural language processing? / IBM [Електронний ресурс] - https://www.ibm.com/topics/natural-language-processing
2. The theory you need to know before you start an NLP project / Arne Laponin [Електронний ресурс] - https://towardsdatascience.com/the-theory-you-need-to-know-before-you-start-an-nlp-project-1890f5bbb793
3. Regular Expressions Clearly Explained with Examples / Jason Chong [Електронний ресурс] - https://towardsdatascience.com/regular-expressions-clearly-explained-with-examples-822d76b037b4
4. Introduction to NLP Deep Learning Theories / Jiahui Wang [Електронний ресурс] - https://towardsdatascience.com/introduction-to-nlp-deep-learning-theories-9d6801e3aa7d
5. Begginers Guide to Regular Expressions in Natural Language Procesing / Himanshi Singh [Електронний ресурс] - https://www.analyticsvidhya.com/blog/2021/03/beginners-guide-to-regular-expressions-in-natural-language-processing/
6. Natural Language Processing (NLP) / Ben Lutkevich [Електронний ресурс] - https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP
7. All about Embedings / Kashyap Kethrani [Електронний ресурс] - https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b