КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

СИСТЕМИ СИНТАКСИЧНОГО АНАЛІЗУ ДЛЯ РОСІЙСЬКОЇ МОВИ

Студента 4-го курсу
Альошка Олексія
Олексійовича

_____
(підпис)

Науковий керівник:
Тарануха Володимир
Юрійович

_____
(підпис)

Робота заслухана на засіданні кафедри математичної інформатики та рекомендована до захисту в ЕК, протокол № …… від ..... 2021р.

Завідувач кафедри Терещенко В.М.

Київ – 2021

РЕФЕРАТ

У дипломній роботі є вступ, 4 розділи, висновки та список використаних джерел (кількість - 23). Робота містить 11 малюнки. Загальний обсяг становить 39 сторінок, основний текст роботи викладено на 32 сторінках.

Ключові слова: СИНТАКСИЧНИЙ АНАЛІЗ, ОБРОБКА ТЕКСТУ, PYTHON, ТОКЕНІЗАЦІЯ, МАШИННЕ НАВЧАННЯ, ПРИРОДНА МОВА, МОРФОЛОГІЧНИЙ АНАЛІЗ, КОРПУС ТЕКСТІВ.

Об’єктом роботи є системи синтаксичних і текстових аналізаторів. Предметом роботи є створення систем синтаксичного аналізу для російської мови.

Метою роботи є аналіз методів і засобів створення синтаксичних аналізаторів тексту для побудови власної аналогічної системи.

Методи розроблення: розробка прикладного програмного забезпечення, консольних додатків.

Програма була розроблена за допомогою інтегрованого середовища розробки PyCharm, студентська версія, та Jupyter Notebook.

Результати роботи: було сплановано та розроблено систему синтаксичного аналізу тексту російською мовою за допомогою мови програмування Python, яка забезпечує простий прикладний програмний інтерфейс для синтаксичного аналізу. Дану розробку можна вважати закінченою та можна використовувати для інших систем роботи з текстами.

ВСТУП

Кожного дня автоматизація різноманітних процесів стає все більш актуальною тенденцією в сучасному світі. Кожний процес автоматизують і перетворюють на процеси, що вимагають менше часу або взагалі можуть обійтися без фізичного втручання людини.

На хвилі спрощення монотонних процесів з'явилися загальнодоступні алгоритми синтаксичного аналізу тексту, які використовуються в подальшому для пошуку синонімів слів, або навіть цілих речень.

Виходячи з актуальності явища текстових аналізаторів, об'єктом роботи стали синтаксичні аналізатори.

Предметом дослідження, в свою чергу є засоби і методи створення текстових аналізаторів.

Метою роботи виступає створення системи синтаксичного аналізу для російської мови засобами мови програмування Python.

Виконаємо наступні завдання, щоб досягнути поставленої мети необхідно виконати наступні завдання:

- Проведемо аналіз предметної області
    - Виявемо значення поняття синтаксичного аналізу тексту
    - Проаналізувати явища синтаксичних аналізаторів та підходи до їх створення
    - Розберемо попередню обробку вхідних даних
- Оберемо засоби розробки
- Проведемо проектування і розробку системи
    - Побудуємо діаграму варіантів використання
    - Побудуємо діаграму класів програмного продукту
    - Розробимо інтерфейс та інструкцію користувача
    - Проілюструємо роботу програми
- Оцінюємо роботу синтаксичного аналізатору

- Виберемо міру оцінювання алгоритму
- Розглянемо проблеми, які могли знизити результат

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Синтаксичний аналіз

Синтаксичний аналіз - це процес аналізу рядка символів, або природної мови, комп'ютерними мовами, або структурами даних, що відповідає правилам офіційної граматики. Термін синтаксичний розбір походить від латинського pars (orationis), що означає частина (мови). [1]

На даний момент різних галузях мовознавства та інформатики у терміна можуть бути різні значення. Традиційний синтаксичний розбір речень часто виконується як метод розуміння точного значення речення або слова, іноді використовуючи такі засоби, як схеми речень. Зазвичай він підкреслює важливість таких граматичних поділів, як підмет і присудок.

В обчислювальній лінгвістиці цей термін використовується для формального аналізу комп’ютером речення чи іншого рядка слів на його складові, в результаті чого дерево синтаксичного аналізу показує їх синтаксичне відношення одне до одного, яке може також містити семантичну та іншу інформацію. Деякі алгоритми синтаксичного аналізу можуть генерувати ліс синтаксичного аналізу або список дерев синтаксичного аналізу для синтаксично неоднозначного введення. [2]

Цей термін також використовується в психолінгвістиці при описі розуміння мови. У цьому контексті синтаксичний розбір стосується того, як люди аналізують речення або фразу (розмовною мовою чи текстом) "з точки зору граматичних складових, ідентифікуючи частини мови, синтаксичні відношення тощо". [1] Цей термін особливо поширене при обговоренні того, які мовні підказки допомагають мовцям інтерпретувати речення із садової доріжки.

В інформатиці цей термін використовується для аналізу комп'ютерних мов, посилаючись на синтаксичний аналіз вхідного коду на його складові

частини з метою полегшення написання компіляторів та інтерпретаторів. Цей термін може також використовуватися для опису розколу або поділу.

Традиційні методи

Традиційна граматична вправа синтаксичного аналізу, іноді відома як аналіз речень, передбачає розбиття тексту на речення, а самі речення - на складові частини мови з поясненням форми, функції та синтаксичного зв’язку кожної частини речення. [3] Це значною мірою визначається вивченням спряжень та відмінювань мови, які можуть бути досить складними для сильно переплетених мов. Щоб проаналізувати таку фразу, як "людина кусає собаку", слід зазначити, що іменник однини "людина" є предметом речення (суб’єктом), дієслово "кусає" - третя особа однини теперішнього часу дієслова "кусати", і іменник однини „собака” є об’єктом речення. Такі методи, як схеми речень, іноді використовуються для позначення зв'язку між елементами речення.

Раніше синтаксичний аналіз був центральним для викладання граматики в усьому англомовному світі і широко розглядався як базовий для використання та розуміння письмової мови. Однак загальне викладання таких технік вже не є актуальним.

Обчислювальні методи

У деяких системах машинного перекладу та обробки природних мов письмові тексти людськими мовами аналізуються за допомогою комп'ютерних програм. [4] Людські речення нелегко аналізувати за програмами, оскільки існує значна неоднозначність у структурі людської мови, використання якої полягає у передачі значення (або семантики) серед потенційно необмеженого кола можливостей, але лише деякі з них є загальними для конкретного випадку. [5] Отже, вислів "Людина кусає собаку" проти "Собака кусає людину" є певним для однієї деталі, але іншою мовою може виглядати як "Людина собаку кусає" з опорою на ширший контекст, щоб розрізнити ці дві можливості, якщо справді ця різниця була

занепокоєння. Важко підготувати офіційні правила для опису неформальної поведінки, хоча очевидно, що деяких правил дотримуються.

Для того, щоб проаналізувати дані на природній мові, дослідники повинні спочатку домовитись про граматику, яку слід використовувати. На вибір синтаксису впливають як лінгвістичні, так і обчислювальні проблеми; наприклад, деякі системи синтаксичного аналізу використовують лексичну функціональну граматику, але загалом аналіз граматик цього типу, як відомо, є NP-повним. Граматика будови фраз, керована головою, - це ще один лінгвістичний формалізм, який був популярний серед парсингового співтовариства, але інші дослідницькі зусилля були зосереджені на менш складних формалізмах, таких як той, що використовується в Penn Treebank. Неглибокий розбір має на меті знайти лише межі основних складових, таких як іменні фрази. Ще однією популярною стратегією уникнення лінгвістичних суперечок є розбір граматики залежностей.

Більшість сучасних аналізаторів є принаймні частково статистичними; тобто вони покладаються на корпус навчальних даних, який вже анотовано (проаналізовано вручну). Цей підхід дозволяє системі збирати інформацію про частоту, з якою різні конструкції трапляються в конкретному контексті. Підходи, які були використані, включають прямі PCFG (імовірнісні безконтекстні граматики), [6] максимальна ентропія, [7] та нейронні мережі. [8] Більшість найбільш успішних систем використовують лексичну статистику (тобто вони враховують ідентичність залучених слів, а також їх частину мови). Однак такі системи вразливі до переобладнання і вимагають дуже багато часу, ресурсів та певного згладжування, щоб бути ефективними.

Алгоритми синтаксичного розбору для природної мови не можуть покладатися на граматику, яка має “ приємні “ властивості, як у граматиках, розроблених вручну для мов програмування. Як уже згадувалося раніше, деякі граматичні формалізми дуже важко обчислювати обчислювально; загалом, навіть якщо бажана структура не є контекстною, для виконання першого проходу використовується якогось безконтекстового наближення до

граматики. Алгоритми, що використовують безконтекстні граматики, часто покладаються на якийсь варіант алгоритму CYK, зазвичай з деякою евристикою, щоб обрізати малоймовірний аналіз, щоб заощадити час. Однак деякі системи обмінюються швидкістю для точності, використовуючи, наприклад, версії алгоритму зменшення зсуву за лінійним часом. Дещо недавно відбувся синтаксичний аналіз переоцінки, в якому аналізатор пропонує деяку кількість аналізів, а більш складна система вибирає найкращий варіант. Семантичні синтаксичні аналізатори перетворюють тексти у подання їх значень [9].

1.2. Синтаксичний аналізатор

Синтаксичний аналізатор або парсер - це програма, у яку в якості вхідних даних потрапляє речення або текст, вона їх аналізує та створює структуру даних - часто це якесь дерево синтаксичного аналізу, дерево абстрактного синтаксису або інша ієрархічна структура, що дає структурне представлення вхідних даних під час перевірки правильності синтаксису. Синтаксичному аналізу можуть передувати або слідувати інші етапи, наприклад, токенізація, лематизація та інші. Деякі з цих етапів нерідко об’єднуються в один крок. Синтаксичному аналізу часто передує окремий лексичний аналізатор, який створює лексеми з послідовності введених символів; в якості альтернативи їх можна поєднати в аналізі без сканера. Синтаксичні аналізатори можуть бути запрограмовані вручну або можуть бути автоматично або напівавтоматично сформовані генератором синтаксичного аналізатора. Синтаксичний аналіз доповнює шаблонування, яке дає форматований вивід. Вони можуть застосовуватися до різних доменів, але часто з'являються разом, наприклад, пара scanf / printf, або етапи введення (аналіз інтерфейсу) та вихід (генерація коду) компілятора.

Вхідними даними синтаксичного аналізатора часто є текст на якійсь комп’ютерній мові, але це може бути також текст на природній мові або менш структуровані текстові дані, для яких ми змушені брати лише певні частини тексту, оскільки дерево синтаксичного аналізу у таких випадках побудувати достатньо важко або взагалі неможливо. Важливий клас простого синтаксичного аналізу виконується за допомогою регулярних виразів, у яких група регулярних виразів визначає регулярну мову та механізм регулярних виразів, що автоматично генерує синтаксичний аналізатор для цієї мови, дозволяючи узгоджувати шаблони та виділяти текст. В інших контекстах регулярні вирази замість цього використовуються перед синтаксичним аналізом як крок лексингу, вихідні дані якого потім використовує синтаксичний аналізатор.

Використання синтаксичних аналізаторів залежить від введення. У випадку з мовами даних синтаксичний аналізатор часто зустрічається як засіб читання файлів програми, наприклад, читання у тексті HTML або XML; ці приклади є мовами розмітки. У разі мов програмування синтаксичний аналізатор - це компонент компілятора або інтерпретатора, який аналізує вихідний код мови комп'ютерного програмування для створення певної форми внутрішнього представлення; синтаксичний аналізатор є ключовим кроком у інтерфейсі компілятора. Мови програмування, як правило, визначаються з точки зору детермінованої безконтекстної граматики, оскільки для них можна писати швидкі та ефективні парсери. Для компіляторів синтаксичний розбір може здійснюватися за один прохід або кілька проходів.

Безконтекстні граматики обмежені в тій мірі, в якій вони можуть виражати всі вимоги мови. Неформально, причина в тому, що пам’ять про таку мову обмежена. Граматика не може пам’ятати наявність конструкції над довільно довгим введенням; це необхідно для мови, якою, наприклад, ім'я повинно бути оголошено, перш ніж воно може бути посиланням. Однак більш потужні граматики, які можуть виражати це обмеження, не можуть бути ефективно проаналізовані.

1.3. Попередня обробка вхідних даних

Для сприйняття синтаксичним аналізатором тексту до нього необхідно застосувати методи попередньої обробки, а саме - токенізацію та морфологічний аналіз.

Токенізація - це процес, у якому речення, фраза або параграф розбиваються на менші шматки - слова та знаки пунктуації. Ці шматки і називаються токенами.

Морфологічний аналіз у комп'ютерній лінгвістиці - це процес, що визначає морфологічні, синтаксичні та, можливо, семантичні властивості слів. На цьому етапі визначаються властивості слів у реченні залежно від того, як слова написані, та незалежно від сусідніх слів та контексту; цей етап завжди передує синтаксичному аналізу. Детально, методами морфологічного аналізу можна вирішити такі задачі:

- Стемінг: Поділ токенів на вільні та зв'язані морфеми. До останніх належать префікси та суфікси.
- Лематизація: Повернення простого або складного слова до його леми - нормальна, словарна форма.
- Визначення граматичних характеристик слів, які називаються грамемами (граматичними значеннями). Наприклад, частина мова, відмінок, рід, число тощо. Множину всіх грамем слова зазвичай називають тегами.

Основні проблеми, з якими можна зіткнутися під час проведення морфологічного аналізу тексту російською мовою, заключаються в наступному:

- До сих пір немає загальноприйнятого стандарту морфологічної розмітки російської мови. Незважаючи на те, що на даний момент існують декілька корпусів у відкритому доступі, не у всіх співпадають розмітки. Тому може виникнути морфологічна неоднозначність при

використанні будь-якого морфологічного аналізатора, бо більшість з них не надають рішення цієї проблеми повністю.

- Деякі слова, хоча і звучать та пишуться однаково, мають різне значення. Це явище називається омонімією. Більш того, у найбільших з корпусів текстів, таких як OpenCorpora та SynTagRus, омонімія частково або повністю знята. Одним з найвідоміших прикладів є слово “мыла”, що може зводитись до словарної форми як дієслово “мыть” і як іменник “мыло”.

1.4. Підходи для синтаксичного аналізу

За допомогою синтаксичного аналізатора проводиться синтаксичний аналіз речень. Для розробки парсера можна виділити такі основні варіанти:

- Підхід з використанням граматик та правил
- Навчання з вчителем

У кожної природної мови є своя граматика та синтаксис. Щоб описати синтаксичну структуру російської та створити множину правил розглянемо такі класи сучасних граматичних теорій:

- Регулярна граматика
- Контекстно-вільна граматика
- Граматика залежностей
- Граматика складових

1.4.1 Регулярна граматика

Граматика 3-го типу в ієрархії Хомського називається регулярною граматикою. Через свою складність та слабку описову здатність з цими формами граматик не виходить працювати ефективно, тому вони не грають ролі навіть в описі мов програмування. Проте їх можна використовувати, якщо мета поставленої задачі - зробити поверхнево-синтаксичний аналіз. Регулярні граматики - окремий випадок контекстно-вільних граматик.

1.4.2 Контекстно-вільна граматика

Контекстно-вільна граматика описує контекстно-вільні мови в теоретичній інформатиці. Це такий кортеж (V, T, P, S), що складається зі словника, термінальних символів, виробничих правил і початкового символу. Контекстно-вільні граматики відповідають граматиці типу 2 ієрархії

Хомського. Вони також мають низку описову здатність, бо правила та смисл не залежить від попередніх слів. Контекстно-вільна граматика задається деяким кінцевим числом правил, чого робити з природними мовами не можна, не визначає структуру тексту і неточно визначає залежності у реченні через можливу розірваність послідовності словосполучень.

1.4.3 Граматика залежностей

Принцип залежності був введений в сучасну граматику в першу чергу Люсьеном Теснером. Теснер цікавився тільки внутрішніми відносинами між одиницями, на яких заснована лінійна теорема. Ці відносини описуються як залежності в граматиці залежностей. Відповідно, речення розуміється як ієрархічно впорядковане ціле. Структура речення не лежить в одновимірному порядку лінійного ланцюжка мови, а є результатом відносин між окремими частинами речення. Це відношення називається зв'язком. Вузли синтаксичного дерева відповідають словам, його краї - зв'язкам.

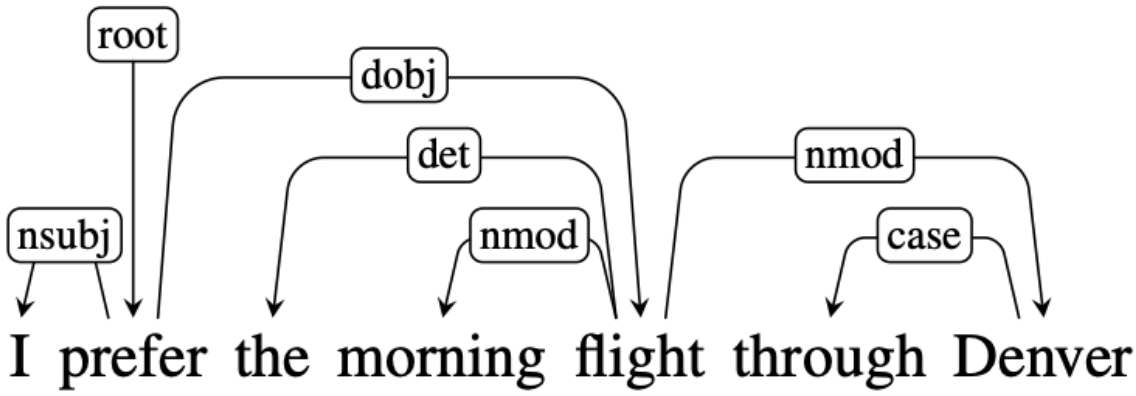


Рис.1. Подання синтаксичної структури речення у вигляді залежностей

Кожне дерево містить центральний вузол як єдину відправну точку і у реченні це завжди дієслово. Така структура залежностей містить в собі лише зв’язки безпосередньо між словами, а не між фразовими складовими.

Цей підхід має гарну репутацію для природних мов з вільним порядком слів, адже він не звертає уваги на порядок слів, а будує зв’язки між словами.

1.4.4 Граматика складових

Поняття складової у синтаксичному аналізі визначає єдине ціле у ієрархічній структурі, що може складатись зі слова або групи слів. Граматика складових виділяє найбільш зв’язані між собою слова в реченні у складові по сенсу. Загалом кожно речення в першу чергу намагаються розбити на групи імену та дієслівну.

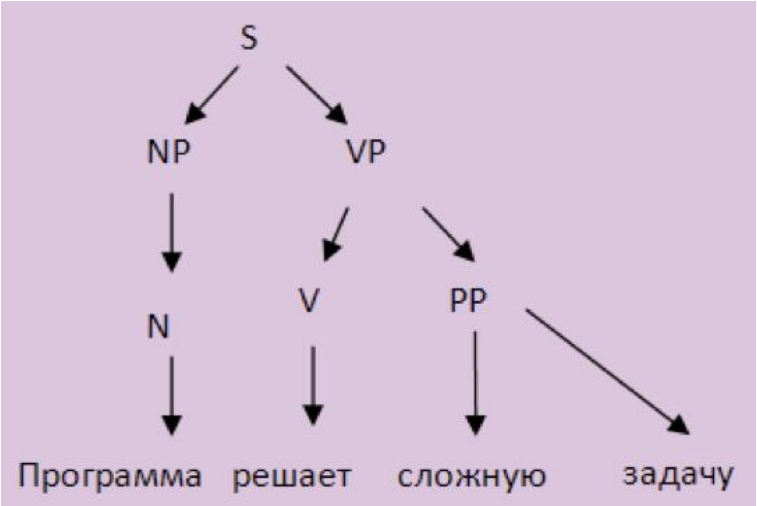


Рис.2. Подання синтаксичної структури речення у вигляді складових

Прикладами складових, що на малюнку, є: NP - називна імена група, іменник у називному відмінку або прикметник у називному + називна імена група або фраза тощо; VP - перехідна дієслівна група, перехідне дієслово + іменник у орудному відмінку або прикметник + перехідна дієслівна група тощо.

Зазвичай граматику складових важко застосувати до природних мов з вільним порядком слів (а саме такою російська мова і є) через відсутність синтаксичних зв’язків між словами.

Можна зробити такий проміжний висновок: правильно підібрана граматика та набір правил зазвичай будуть давати високий результат при синтаксичному парсингу текстів. Тим не менш, для того, щоб реалізувати даний підхід, не завадять команда професіональних лінгвістів, багато

ресурсів та часу, щоб скласти усі правила для російської мови. Тому мій вибір пав на машине навчання.

1.4.5 Навчання з учителем

Навчання з учителем - це засіб машинного навчання, що полягає в навчанні інтелектуальної системи на прикладах з заздалегідь визначеними правильними відповідями. У навчанні з учителем вхідні дані потрапляють з вхідними змінними і вихідною зміною, яку алгоритм і намагається правильно предсказати. Загальна формула має такий вид: □ = □(□).

Мета цього засоба - апроксимувати функцію відображення f якнайкраще, щоб точність передбачень була якомога високою.

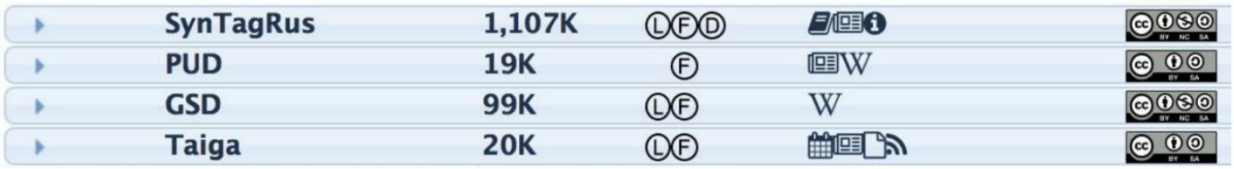
В нашому випадку ми даємо системі на вхід приклади з визначеними відповідями, що являють собою речення з розмічених корпусів текстів, в яких заздалегідь відома синтаксична структура.

РОЗДІЛ 2. ОГЛЯД ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ВИБІР

2.1. Корпуси текстів російською мовою

Universal Dependencies - це фреймворк для узгодженого анотування корпусів текстів для різних природних мов в рамках граматики залежностей.

В Universal Dependencies для російської мови існують 4 таких корпусів текстів.



Щорічно в рамках проекту проводиться змагання між розробниками систем синтаксичного аналізу, і всі алгоритми для російської навчалися на SynTagRus. Тому було обрано та завантажено цей корпус текстів, який доступно у відкритому доступі на Github. На даний момент SynTagRus налічує в собі понад міліон слів та понад 66 тисяч речень на різну тематику (сучасна фантастика, науково-популярні, газетні та журнальні статті, датовані між 1960 і 2016 роками, тексти новин тощо). Цей корпус було проаналізовано вручну, в ньому зберігаються морфологічні та синтаксичні анотації у вигляді повного дерева залежностей для кожного речення.

Корпус текстів складається з трьох частин (навчальної, валідаційної та тестової) та зберігається у форматі conllu. Тестові дані можна відкинути, адже вони були використані спеціально для змагання. Навчальні дані будуть використовуватися для навчання алгоритму синтаксичного аналізу. На валідаційних даних робиться оцінка навченного алгоритму.

2.2. Мова програмування

Для цієї роботи була обрана мова програмування Python.

Python - мова комп’ютерного програмування загального призначення, що поєднує процедурні, функціональні та об’єктно-орієнтовані парадигми.

Python є дуже читабельною та узгодженою мовою, і це в цілому виділяє його на фоні інших мов програмування. Строки коду Python з легкістю сприймаються, навіть якщо ви його не писали.

Python код в порівнянні з компільованими або статично типізованими мовами (C++, Java, C#) часто займає третину еквівалентного коду. Саме це багаторазово знижує час написання програми на Python. Більшість програм Python універсальні та запускаються без проблем на основних комп'ютерних платформах як Windows, Linux, MacOs.

Python має багату стандартну бібліотеку, в якій зібрано широкий спектр зручних попередньо побудованих та портативних функціональних можливостей. Для підтримки сторонніх програм Python можна розширити великим набором бібліотек. На сьогоднішній день Python широко використовується у таких галузях як машинне навчання, комп’ютерна лінгвістика, аналіз даних тощо. Це через те, що Python має багату екосистему бібліотек для наукових обчислень, наприклад numpy, scikit-learn, tensorflow, pandas, jupyter-notebook, matplotlib, nltk, stanza та інші.

Python з його широкою множиною різних технологій значно облегшує життя програмістам та наділяє їх більше задоволенням, ніж рутиною.

Завдяки простоті використання Python і вбудованому набору інструментів, він може зробити процес програмування більше задоволенням, ніж рутиною. Хоча це може бути нематеріальним перевагою, Python сильно впливає на продуктивність розробників.

Найбільшими перевагами для мене є перші два пункти (якість та продуктивність) та наявність великої кількості бібліотек, за допомогою яких

процес синтаксичного аналізу значно спрощується. Саме з цих міркувань для цієї роботи я вибираю мову програмування Python.

2.3. Огляд бібліотек Python для парсингу тексту

Python має широкий ассортимент бібліотек, якими зручно користуватися для обробки тексту, морфологічного аналізу, синтаксичного аналізу, обробки природних мов та машинного навчання.

pymorphy2 - морфологічний аналізатор, що повністю написан на Python. Його можливості заключаються в приведенні слів до нормальної (словникової форми), ставленні слів у відповідну потрібну форму та доставанні інформації про грамеми слів.

pymorphy2 базується на словнику OpenCorpora (містить понад 500 тис. лексем та 5 мільйонів слів); якщо слово не знайдене у словнику, для нього витягуються парадигми з лексем (будування гіпотези) та предсказується найкращий варіант.

До кожного слова pymorphy2 пропонує декілька розборів - об’єктів, що містять дані усіх грамем, нормальної (словникової) форми, та параметру score - оцінка ймовірності того, даний розбір був правильним. Так влаштовано, що самий перший розбір завжди найбільш ймовірний, тому його завжди і будемо вибирати в якості кандидата.

nltk - провідна платформа для побудови програм Python для роботи з даними людської мови. І хоча за допомогою цієї бібліотеки в нас є можливість працювати з багатьма засобами обробки людської мови, нам потрібно лише дві функції для токенізації: sent_tokenize (розбиває текст на речення), word_tokenize (розбиває речення на слова/токени).

stanza - ще одна провідна бібліотека для обробки природних мов. Починаючи з початкового тексту і закінчуючи синтаксичним аналізом і розпізнаванням сутностей, Stanza пропонує сучасні моделі для мов, а саме - інструменти, які можна використовувати в конвеєрі, для перетворення рядка,

що містить текст на людській мові, в списки пропозицій і слів, для генерації базових форм цих слів, їх частин мови і морфологічних характеристик, для аналізу залежності синтаксичної структури, і розпізнавати іменовані об'єкти.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Діаграма варіантів використання

Найпростіша діаграма використання - це представлення взаємодії між користувачем та системою, яка показує взаємозв'язок між користувачем та різними видами використання, в яких користувач бере участь. Діаграми випадків використання можуть ідентифікувати різні типи користувачів системи та різні випадки використання, і часто супроводжуються іншими типами діаграм. Мета представлена колом або еліпсом.

Хоча сам випадок використання може детально розглянути кожну можливість, використання прикладних діаграм може допомогти забезпечити огляд системи більш високого рівня. Хтось раніше говорив, що "план використання - це принцип вашої системи".

Завдяки спрощеному характеру, план використання може бути хорошим інструментом комунікації для зацікавлених сторін. Ці креслення намагаються імітувати реальний світ і дати зрозуміти зацікавленим сторонам, як буде розвиватися система. Сіау та Лі провели дослідження, щоб визначити, чи є реальні ситуації у сценаріях використання чи вони непотрібні. Було виявлено, що діаграми випадків використання передають намір системи зацікавленим сторонам у більш спрощений спосіб, і вони "пояснюються більш повно, ніж схеми класів".

Мета використання діаграм - показати динамічні аспекти системи. Додаткові схеми та документи можуть бути використані для забезпечення повного функціонального та технічного представлення системи. Вони забезпечують спрощене і графічне представлення того, що система насправді повинна робити.

Проект:

- Межа системи - прямокутник з іменем зверху та еліпсом (прецедентом) всередині. Корисну інформацію без корисної інформації часто можна опустити,
- Актор (англ. Actor) - стилізована людська роль, що представляє набір ролей користувачів (широке розуміння: люди, зовнішні сутності, класи, інші системи), що взаємодіють із суттю (системи, підсистеми, класи). Актори не можуть бути пов'язані між собою (за винятком взаємозв'язку з обробкою / дослідженнями),
- Прецедент - еліпс з написом, що вказує на роботу системи (можливо, включаючи можливі варіанти), що призвело до результатів, які спостерігали учасники. Заголовок може бути назвою або описом (з точки зору актора) системи "що" (а не "як"). Вони пов'язують прецеденти з безперервними (атомними) конкретними послідовностями дій, що ілюструють поведінку. Під час сценарію актори обмінюються системною інформацією. Сценарій може відображатися на попередній схемі відео коментарів UML. Кілька різних ситуацій можуть бути пов’язані з прецедентами

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

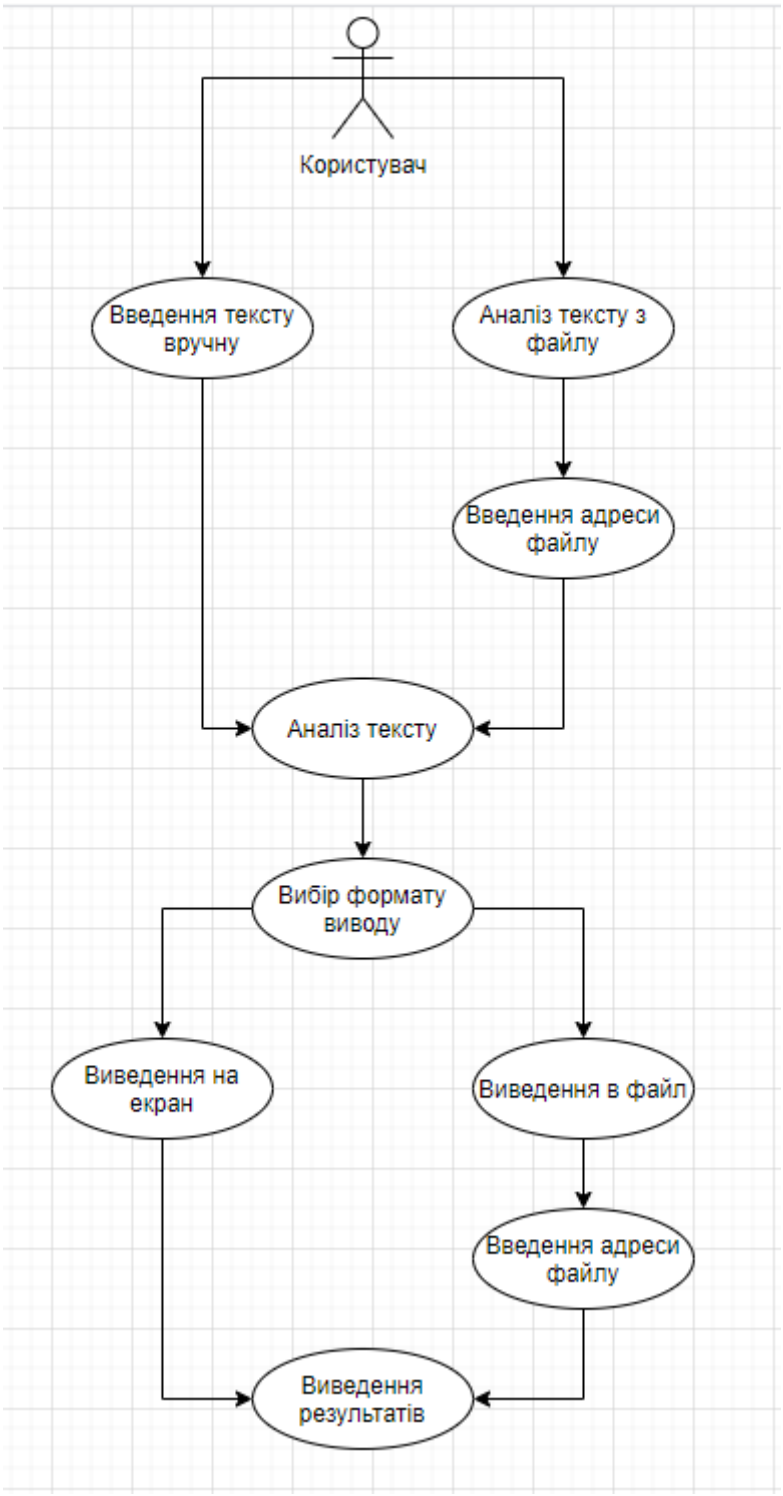


Рис. 3.1 — Діаграма варіантів використання

3.2. Діаграма класів програмного продукту

Діаграма класів Unified Modeling Language (UML) - це статична структурна діаграма, яка описує структуру системи та відображає системні класи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграми класів, що являються будівельними блоками об’єктно-орієнтованого моделювання, використовуються для загального концептуального моделювання структури програми та детального моделювання для перетворення моделі в програмний код, а також для моделювання даних. Класи на схемі класів представляють основні елементи, взаємодії в програмі та класи програмування.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При розробці системи виявлення багатьох класів та згрупування їх у схему класів це допомагає визначити статичний зв’язок між ними. При детальному моделюванні категорії концептуальних об’єктів зазвичай поділяються на кілька підкатегорій.

Залежність - це семантичний зв’язок між залежними елементами та незалежними елементами моделі. Якщо зміна визначення одного елемента (сервера або цілі) може змінити іншого (клієнта або джерела), воно існує між двома елементами. Ця асоціація однобічна. Актуальність представлена пунктирною стрілкою, де відкрита стрілка вказує від замовника до постачальника.

Для подальшого опису поведінки системи ці діаграми класів можуть бути доповнені діаграмами стану або автоматами стану UML.

Асоціація пропонує безліч посилань. Бінарні асоціації (з двома кінцями) зазвичай виражаються у вигляді рядків. Асоціації можуть пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінець асоціації можна змінити, використовуючи такі атрибути, як ім'я ролі, індикатор атрибутів, кратність та видимість.

Існує чотири різні типи асоціацій: двосторонні, односторонні, агрегаційні (включаючи комбіновану агрегацію) та рефлексивні. Найпоширенішими є двосторонні та односторонні асоціації.

Наприклад, клас польоту асоціюється з класом двостороннього літака. Асоціація - це статичний зв’язок, що використовується спільно між об’єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частину або частину відносин. Як показано на малюнку, професор "має" клас для викладання. Як асоціація, агрегація може бути названа та мати ті ж модифікації, що і асоціація. Однак агрегація не може включати більше двох категорій; вона повинна бути бінарною асоціацією. Крім того, майже немає різниці між агрегуванням та асоціацією в процесі реалізації, а планування може повністю пропустити взаємозв'язок агрегування. [7]

Коли клас є колекцією або контейнером інших класів, відбувається агрегування, але розміщений клас не сильно залежить від життєвого циклу контейнера. Коли контейнер знищений, вміст контейнера все ще існує.

В UML він зображений графічно у вигляді порожнистого діаманта на хост-класі, що з'єднує його з рядком хост-класу. Агрегація - це семантично розширений об’єкт, який розглядається як одиниця в багатьох операціях, хоча фізично складається з декількох менших об’єктів.

Наприклад: бібліотека та студенти. Тут студенти можуть існувати без бібліотеки, і між студентом і бібліотекою існує ціле.

Це показує, що один із двох пов’язаних класів (підкласів) вважається спеціалізацією іншого (супертип), а суперклас - узагальненням підкласів. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром супертипу. Типове дерево узагальнення цієї форми можна знайти в біологічній систематиці: людина - це підгрупа мавп, мавпа - підгрупа ссавців тощо. Цей зв’язок найкраще зрозуміти з фрази «А - це В» (люди - це ссавці, а ссавці - тварини).

Графічне зображення узагальнення UML - це форма порожнистого трикутника в кінці лінії суперкласу (або дерева ліній), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також називають спадкоємством або відносинами "є".

Суперклас (базовий клас) у відносинах узагальнення також називають "батьківським класом", суперкласом, базовим класом або базовим типом.

Підтипи у відносинах спеціалізації також називаються "дочірніми" підкласами, похідними класами, похідними типами, успадкованими класами або успадкованими типами.

Зверніть увагу, що ці стосунки повністю відрізняються від біологічних стосунків між батьком та дитиною: використання цих термінів дуже поширене, але може ввести в оману.

А - це тип В

Наприклад, "дуб - це дерево", "машина - транспортний засіб"

Короткий зміст може відображатися лише на схемі класів та діаграмі використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, де один елемент моделі (замовник) реалізує (реалізує або виконує) поведінку, визначену іншим елементом (постачальником) моделі.

Графічне представлення реалізації UML - це порожнистий трикутник, який підключений до кінця пунктирного інтерфейсу (або дерева рядків)

одного або кількох реалізаторів. Проста стрілка використовується в кінці пунктирного інтерфейсу, щоб підключити її до користувача. Схема компонентів використовує графічну умову «кульова розетка» (продавець розміщує кульку або льодяник, а користувач відображає кульову розетку). Реалізація може відображатися лише в режимі класу або компонента. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднують елементи замовника з елементами постачальника. Зв'язок реалізації між класом / компонентом та інтерфейсом вказує на те, що клас / компонент реалізує операцію.

Залежність - це слабша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Якщо незалежний клас залежить від змінної параметра або локальної змінної методу, то один клас залежить від іншого класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між цими двома класами слабкі. Вони взагалі не реалізовані зі змінними-членами. Натомість вони можуть бути реалізовані як параметри функцій-членів.

Інші стосунки часто описують як "А має В" (у самки кота є кошеня і кошка).

Представлення асоціації UML - це лінія, що з'єднує два пов'язаних класи. У кожному кінці рядка є зайві символи. Наприклад, ми можемо використовувати кінчик стрілки, щоб вказати, що кінчик видно з хвоста стрілки. Ми можемо вказати право власності, поставивши кульку на цьому кінці, яку роль відіграє елемент, і вказати ім'я ролі та кілька екземплярів сутності (з іншого боку, діапазон об'єктів, що беруть участь в асоціації).

Довгострокова інформація, що обробляється системою моделювання сутності, і іноді поведінка, пов'язана з інформацією, моделюється. Вони не повинні розпізнаватися як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Або їх можна намалювати як звичайні заняття з "суттєвим" стереотипом щодо назви класу.

Рис. 3.2 — Діаграма класів

3.3. Розробка інтерфейсу користувача та інструкція користувача

Графічний інтерфейс користувача представлено інтерфейсом командного рядку, який виводить всі необхідні дані і запрошує дані в користувача, якщо є така необхідність.

Приклади взаємодії користувача з інтерфейсом зображено на рисунку 3.3.

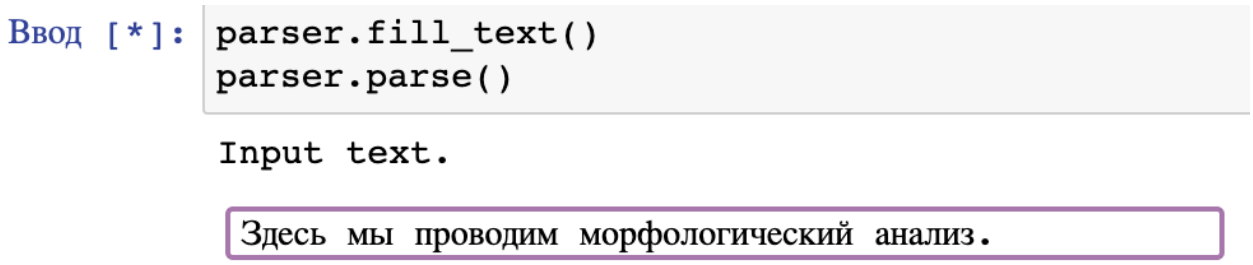


Рис. 3.3 — Приклад взаємодії користувача з інтерфейсом

Для роботи з програмою алгоритм дій наступний:

- Відкрити додаток через IDE або командний рядок
- Обрати формат введення тексту (ввести відповідну цифру)
    - При введенні вручну — ввести текст
    - При введенні з файлу — ввести адресу файлу
- Обрати формат виводу результатів (ввести відповідну цифру)
    - При виводі в файл — ввести адресу файлу

3.4. Робота програми

У даному підрозділі поетапно проілюструємо виконання програми синтаксичного аналізу.

3.4.1. Токенізація та морфологічний аналіз

Для зручності було написано класс, який приймає строку тексту, розбиває його на речення, а речення в свою чергу - на слова (далі токени).

```
def input_text():
    print("Input text.")
    return input()


class MorphParser:
    def __init__(self, lang=None):
        self.morph = MorphAnalyzer() if lang is None else MorphAnalyzer(lang='uk')
        self.sentences = []
        self.text = None
        self.sentence_words = []
        self.sentence_tokens = []
        self.sentence_parses = []
        self.candidate_parses = []

    def fill_text(self, text=None):
        self.text = input_text() if text is None else text

    def parse(self):
        self.sentences = np.array(sent_tokenize(self.text), str)
        self.sentence_words = [word_tokenize(s) for s in self.sentences]
        self.sentence_parses = [[self.morph.parse(word) for word in words] for words in self.sentence_words]
        self.candidate_parses = [parses[0] for parses in self.sentence_parses]
        self.candidate_parses = [[word_parses[0] for word_parses in words_parses] for words_parses in self.sentence_parses]

    def sentence_data(self, sentence_ix=0):
        return [self.word_sentence_data(i, sentence_ix) for i in range(len(self.sentence_words[sentence_ix]))]

    def word_sentence_data(self, word_ix=0, sentence_ix=0):
        return self.sentence_parses[sentence_ix][word_ix]

    def sentence_candidates_data(self, sentence_ix=0):
        return pd.Series({
            'sentence': self.sentences[sentence_ix],
            'candidates': pd.DataFrame(self.candidate_parses[sentence_ix])
        })
```

Рис. 3.4 — написаний класс

Розбивання на речення відбувається за допомогою функції бібліотеки nltk sent_tokenize(), на токени - word_tokenize(). Морфологічний аналіз

відбувається за допомогою pymorphy2.MorphAnalyzer().parse(), що приймає на вхід слово та повертає список усіх можливих розбірів цього слова. Розбір в данному випадку - це об'єкт класу Parse(), що складається з набору усіх грамем, нормальної (словникової) форми, та параметр score - оцінка ймовірності того, даний розбір був правильним. Надалі кожного разу за допомогою написаного класу буде повертатися розбір з найвищим score, цей розбір завжди буде перший у списку.

Ввод [22]:
```
parser.fill_text('Здесь мы проводим морфологический анализ.')
parser.parse()
```

Ввод [23]:
```
data = parser.sentence_candidates_data(0)
data.candidates
```

Out[23]:

	word	tag	normal_form	score	methods_stack
0	здесь	ADVB,Prdx	здесь	1.000000	((DictionaryAnalyzer(), здесь, 404, 0),)
1	мы	NPRO,1per plur,nomn	мы	1.000000	((DictionaryAnalyzer(), мы, 2072, 0),)
2	проводим	VERB,perf,tran plur,1per,futr,indc	проводить	0.250000	((DictionaryAnalyzer(), проводим, 979, 6),)
3	морфологический	ADJF masc,sing,nomn	морфологический	0.500000	((DictionaryAnalyzer(), морфологический, 16, 0),)
4	анализ	NOUN,inan,masc sing,nomn	анализ	0.783783	((DictionaryAnalyzer(), анализ, 34, 0),)
5	.	PNCT	.	1.000000	((PunctuationAnalyzer(score=0.9), .),)

Рис. 3.5 — приклад розбиття речення на слова з найімовірнішим морфологічним розбором

3.4.2. Машинне навчання для синтаксичного аналізу

Для синтаксичного аналізу було обрано нейроний пайплайн stanza.Pipeline('ru'). Спочатку завантажемо наш навчальний корпус у об'єкт Stanza Document:

Ввод [8]:
```
from stanza.utils.conll import CoNLL
train_data = 'ru_corpus/ru_syntagrus-ud-train.conllu'
validation_data = 'ru_corpus/ru_syntagrus-ud-test.conllu'
test_data = 'ru_corpus/ru_syntagrus-ud-dev.conllu'

print('Loading datasets...')

train_doc = CoNLL.conll2doc(train_data)
validation_doc = CoNLL.conll2doc(validation_data)
test_doc = CoNLL.conll2doc(test_data)

print('Loading finished.')
```

```
Loading datasets...
Loading finished.
```

Рис. 3.6 — зчитування трьох датасетів у Document`и

Після завантаження перевіримо якесь речення російською мовою.

```
Ввод [13]: nlp = stanza.Pipeline('ru', processors='tokenize,pos,pos,lemma,depparse', tokenize_pretokenized=True)
           nlp('Привет, как поживаешь, мой друг?')
```

```
2021-06-10 11:01:56 INFO: Loading these models for language: ru (Russian):
=========================
| Processor | Package   |
-------------------------
| tokenize  | syntagrus |
| pos       | syntagrus |
| lemma     | syntagrus |
| depparse  | syntagrus |
=========================

2021-06-10 11:01:56 INFO: Use device: cpu
2021-06-10 11:01:56 INFO: Loading: tokenize
2021-06-10 11:01:56 INFO: Loading: pos
2021-06-10 11:01:57 INFO: Loading: lemma
2021-06-10 11:01:57 INFO: Loading: depparse
2021-06-10 11:01:57 INFO: Done loading processors!
```

Рис. 3.7 — приклад подання речення для побудови синтаксичного дерева

Out[13]:


```
[
  [
    {
      "id": 1,
      "text": "Привет,",
      "lemma": "привести",
      "upos": "VERB",
      "feats": "Aspect=Imp|Mood=Imp|Number=Sing|Person=2|VerbForm=Fin|Voice=Act",
      "head": 0,
      "deprel": "root",
      "misc": "",
      "start_char": 0,
      "end_char": 7
    },
    {
      "id": 2,
      "text": "как",
      "lemma": "как",
      "upos": "ADV",
      "feats": "Degree=Pos",
      "head": 3,
      "deprel": "advmod",
      "misc": "",
      "start_char": 8,
      "end_char": 11
    },
    {
      "id": 3,
      "text": "поживаешь,",
      "lemma": "поживать",
      "upos": "VERB",
      "feats": "Aspect=Imp|Mood=Ind|Number=Sing|Person=2|Tense=Pres|VerbForm=Fin|Voice=Act",
      "head": 1,
      "deprel": "xcomp",
      "misc": "",
      "start_char": 12,
      "end_char": 22
    },
    {
      "id": 4,
      "text": "мой",
      "lemma": "мой",
      "upos": "DET",
      "feats": "Case=Acc|Gender=Masc|Number=Sing",
      "head": 5,
      "deprel": "det",
      "misc": "",
      "start_char": 23,
      "end_char": 26
    },
    {
      "id": 5,
      "text": "друг?",
      "lemma": "друг",
      "upos": "NOUN",
      "feats": "Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing",
      "head": 3,
      "deprel": "obj",
      "misc": "",
      "start_char": 27,
      "end_char": 32
    }
  ]
]
```


Рис. 3.8 — синтаксичне дерево введеного речення

РОЗДІЛ 4. ОЦІНКА АЛГОРИТМУ СИНТАКСИЧНОГО АНАЛІЗУ

Мірою оцінки точності роботи синтаксичного аналізатора було обрано F1-score. Для цієї міри виділяють такі поняття як повнота так влучність.

- Влучність - частка правильно визначених позитивних результатів серед усіх позитивних результатів.
- Повнота - частка правильно визначених позитивних результатів серед усіх прикладів, які повинно було визначити як позитивні.

Маємо таку формулу F1-міри:

$$F_1 = \frac{2}{\text{повнота}^{-1} + \text{влучність}^{-1}} = 2 \cdot \frac{\text{влучність} \cdot \text{повнота}}{\text{влучність} + \text{повнота}} = \frac{\text{ІП}}{\text{ІП} + \frac{1}{2}(\text{ХП} + \text{ХН})}.$$

Отже, ця міра є середнім гармонійним влучності та повноти.

Після тренування пайплайну stanza на навчальних даних з заздалегідь токенізованим текстом та тестуванні на валідаційних даних, алгоритм показав точність у 76%. У більшості провідних синтаксичних аналізаторів російською це значення варіюється між 87% і 94%. Найбільш ймовірною причиною відставання на 10%-15% є не завжди вірний вибір розбору слів при морфологічному аналізі за допомогою pymorphy2. Ми завжди вибираємо перший варіант розбору слова, та згідно з документацію бібліотеки це буде правильним вибором у 72% випадках.

ВИСНОВКИ

Метою роботи було створення системи синтаксичного аналізу для російської мови засобами мови програмування Python.

Для досягнення поставленої мети було виконано наступні завдання:

- Проведено аналіз предметної області
  - Виявлено значення поняття синтаксичного аналізу тексту
  - Проаналізовано явища синтаксичних аналізаторів та підходи до їх створення
  - Розібрано попередню обробку вхідних даних
- Обрано засоби розробки
  - Мова Python
  - Фреймворки обраної мови програмування
- Проведено проектування і розробку системи
  - Побудовано діаграму варіантів використання
  - Побудовано діаграму класів програмного продукту
  - Розроблено графічний інтерфейс користувача та інструкцію користувача
  - Проілюстровано роботу аналізатора
- Оцінено роботу синтаксичного аналізатора:
  - Вибрана міра оцінювання алгоритму
  - Розглянуто проблеми, що могли знизити результат

В ході роботи, в першому розділі було проведено аналітичний огляд понять синтаксичного аналізу тексту, явища синтаксичних аналізаторів тексту, що дало загальне розуміння стосовно предметної області, яке в подальшому лягло в основу проектування функціоналу системи.

В другому розділі було обрано мову програмування, яка була використана для розробки програмного забезпечення. Було виявлено, що мова програмування Python ідеально підходить під цілі проекту, так як має

велику кількість бібліотечних текстових аналізаторів та функцій для роботи з текстами, що стало вирішальним фактором у виборі мови програмування.

В третьому розділі повністю описано проектування і розробку системи за допомогою UML-діаграм варіантів використання і класів, опису графічного інтерфейсу. Окрім цього сформовано керівництво користувача, яке допоможе рядовим користувачам швидко розібратися і функціоналом програмного продукту.

В четвертому розділі була обрана міра оцінювання точності алгоритму та розглянута проблема, яка ймовірно знизила результат.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://pymorphy2.readthedocs.io/
2. http://www.vestnik.vsu.ru/pdf/analiz/2012/02/2012-02-29.pdf
3. https://ufal.mff.cuni.cz/~straka/papers/2017-conll_udpipe.pdf
4. https://web.stanford.edu/~jurafsky/slp3/14.pdf
5. "Parse". dictionary.reference.com. Retrieved 27 November 2010.
6. Masaru Tomita (6 December 2012). Generalized LR Parsing. Springer Science & Business Media. ISBN 978-1-4615-4034-2.
7. "Grammar and Composition".
8. Christopher D.. Manning; Christopher D. Manning; Hinrich Schütze (1999). Foundations of Statistical Natural Language Processing. MIT Press. ISBN 978-0-262-13360-9.
9. Jurafsky, Daniel (1996). "A Probabilistic Model of Lexical and Syntactic Access and Disambiguation". Cognitive Science. 20 (2): 137–194. CiteSeerX 10.1.1.150.5711. doi:10.1207/s15516709cog2002_1.
10. Klein, Dan, and Christopher D. Manning. "Accurate unlexicalized parsing." Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. Association for Computational Linguistics, 2003.
11. Charniak, Eugene. "A maximum-entropy-inspired parser." Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference. Association for Computational Linguistics, 2000.
12. Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
13. Jia, Robin; Liang, Percy (2016-06-11). "Data Recombination for Neural Semantic Parsing". arXiv:1606.03622 [cs.CL].
14. Berant, Jonathan, and Percy Liang. "Semantic parsing via paraphrasing." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2014.

15. Aho, A.V., Sethi, R. and Ullman, J.D. (1986) " Compilers: principles, techniques, and tools." Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
16. Sikkel, Klaas, 1954- (1997). Parsing schemata : a framework for specification and analysis of parsing algorithms. Berlin: Springer. ISBN 9783642605413. OCLC 606012644.
17. Frost, R., Hafiz, R. and Callaghan, P. (2007) " Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars ." 10th International Workshop on Parsing Technologies (IWPT), ACL-SIGPARSE , Pages: 109 - 120, June 2007, Prague.
18. Frost, R., Hafiz, R. and Callaghan, P. (2008) " Parser Combinators for Ambiguous Left-Recursive Grammars." 10th International Symposium on Practical Aspects of Declarative Languages (PADL), ACM-SIGPLAN , Volume 4902/2008, Pages: 167 - 181, January 2008, San Francisco.
19. Rekers, Jan, and Andy Schürr. "Defining and parsing visual languages with layered graph grammars." Journal of Visual Languages & Computing 8.1 (1997): 27-55.
20. Rekers, Jan, and A. Schurr. "A graph grammar approach to graphical parsing." Visual Languages, Proceedings., 11th IEEE International Symposium on. IEEE, 1995.
21. Zhang, Da-Qian, Kang Zhang, and Jiannong Cao. "A context-sensitive graph grammar formalism for the specification of visual languages." The Computer Journal 44.3 (2001): 186-200.
22. Jill Fain Lehman (6 December 2012). Adaptive Parsing: Self-Extending Natural Language Interfaces. Springer Science & Business Media. ISBN 978-1-4615-3622-2.
23. taken from Brian W. Kernighan and Dennis M. Ritchie (Apr 1988). The C Programming Language. Prentice Hall Software Series (2nd ed.). Englewood Cliffs/NJ: Prentice Hall. ISBN 0131103628. (Appendix A.13 "Grammar", p.193 ff)