МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ МЕТОДУ ВИЯВЛЕННЯ ПОШТОВИХ АДРЕС У ДОВГИХ ТЕКСТАХ»**

Виконала: студентка 4 курсу, групи 6.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Є. О. Мержвинська
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук, к.т.н. Добровольський Г. А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к ф.-м.н. Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук, д.т.н., професор

_____ Чопоров С.В.
(підпис)

" 30 " січня 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Мержвинській Єлизаветі Олександрівні
(прізвище, ім'я та по-батькові)

1. Тема роботи Реалізація методу виявлення поштових адрес у довгих текстах

Керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація методу виявлення поштових адрес у довгих текстах

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

66. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 30.01.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	27.01.2023	
2.	Збір вихідних даних.	27.02.2023	
3.	Обробка методичних та теоретичних	27.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	27.04.2023	
5.	Розробка третього розділу.	27.05.2023	
6.	Оформлення та нормоконтроль	05.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2023	

Студент _____ Є. О. Мержвинська
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г. А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація методу виявлення поштових адрес у довгих текстах»: 37 с., 4 рис., 2 табл., 20 джерел, 1 додатків.

ГЕОКОДУВАННЯ, МАШИННЕ НАВЧАННЯ, МЕТОД РУХОМОГО ВІКНА, ОБРОБКА ПРИРОДНОЇ МОВИ, РОЗБІР АДРЕСИ, РОЗПІЗНАВАННЯ ТЕКСТУ.

Об’єкт дослідження – задача виявлення та розбору поштових адрес у довгих текстах.

Мета роботи: визначення ефективності застосування методу рухомого вікна разом з методами розбору поштових адрес для пошуку адрес у довгому тексті.

Метод дослідження – аналітичний.

Для досягнення поставленої мети був розроблений метод виявлення адрес, що базується на методі рухомого вікна та на методах розбору поштових адрес з функціоналом бібліотеки Libpostal.

Метод протестований на даних різних форматів та мов. Отримані результати свідчать про високу ефективність застосування методу рухомого вікна разом з методами розбору поштових адрес бібліотеки Libpostal у довгих текстах. Застосування цих методів дозволяє точно розпізнавати поштові адреси навіть у випадках, коли адреси мають різні формати, включають додатковий текст або містять помилки.

Розроблений підхід до виявлення поштових адрес у довгих текстах може бути використаний для поліпшення точності та швидкості обробки текстової інформації. Його застосування є особливо актуальним у сферах обробки природної мови, пошуку інформації та електронної комерції. Використання цього підходу сприятиме автоматизації процесів індексації та категоризації текстів, покращенню пошукових систем.

SUMMARY

Bachelor’s qualifying thesis «Implementation of the Method for Postal Addresses Recognition in Long Texts»: 37 pages, 4 figures, 2 tables, 20 references, 1 supplements.

GEOCODING, MACHINE LEARNING, MOVING WINDOW METHOD, NATURAL LANGUAGE PROCESSING, ADDRESS PARSING, TEXT RECOGNITION.

Object of the study – the task of detecting and parsing postal addresses in long texts.

Aim of the study: to determine the effectiveness of using the moving window method in combination with methods of parsing postal addresses for finding addresses in long texts.

Method of research – analytical.

To achieve the set goal, a method for detecting addresses has been developed, which is based on the moving window method and address parsing methods with the functionality of the Libpostal library.

The method has been tested on data of various formats and languages. The obtained results demonstrate the high efficiency of applying the moving window method together with the address parsing methods of the Libpostal library in long texts. The application of these methods allows for accurate recognition of postal addresses even in cases where the addresses have different formats, include additional text, or contain errors.

The developed approach to detecting postal addresses in long texts can be used to improve the accuracy and speed of processing textual information. Its application is particularly relevant to the fields of natural language processing, information retrieval, and e-commerce. The utilization of this approach will contribute to the automation of text indexing and categorization processes, as well as enhance search systems.

ЗМІСТ

ВСТУП

Задача виявлення поштових адрес у довгих текстах набуває все більшої актуальності в різних сферах, зокрема в обробці природної мови, пошуку інформації та електронній комерції. З зростанням кількості цифрових даних і збільшенням попиту на точну обробку текстової інформації, можливість видобування та аналізу поштових адрес з великих обсягів тексту стала критично важливим завданням.

Задача розбору поштових адрес полягає у встановленні відповідності між вхідним рядком та елементами, що описують місцезнаходження: країна, місто, область, поштовий індекс, вулиця, номер будинку.

Наприклад, є рядок "Arnulfstr. 19 D-80335 Munich", де

- вулиця: "Arnulfstr",
- будинок: "19",
- поштовий індекс: "D-80335",
- місто: "Munich".

Існує багато методів [Методи виявлення адрес] для розбору рядків, про які відомо, що вони є поштовою адресою. Незважаючи на наявність існуючих методів виявлення адрес, точна ідентифікація поштових адрес у довгих текстах залишається проблемою через наступні причини:

- формат поштових адрес сильно відрізняється від однієї країни до іншої;
- адреса може бути написана в нестандартному форматі;
- в адресі може бути відсутні один або декілька елементів;
- адреса може мати додатковий текст, наприклад, ім’я одержувача, назва будівлі;
- адреса може містити орфографічні помилки, нестандартні скорочення.

Усі перераховані вище фактори ускладнюють вилучення адрес.

Метою даної роботи є визначення ефективності застосування методу рухомого вікна разом з методами розбору поштових адрес бібліотеки Libpostal у довгому тексті. Основним завданням є встановлення ефективності даного методу та демонстрація його потенціалу для вирішення задач пошуку адрес у тексті.

1 МЕТОДИ ВИЯВЛЕННЯ АДРЕС

Існують різні техніки, алгоритми та методи, які використовуються для знаходження, розпізнавання та перевірки різних типів адрес.

1.1 Розпізнавання іменованих сутностей

Розпізнавання іменованих сутностей [1] – це підзавдання вилучення інформації, яке знаходить та класифікує іменовані сутності, згадані в неструктурованому тексті, за попередньо визначеними категоріями, такими як імена, організації, місця розташування, медичні коди, кількість, грошові значення, відсотки тощо. Існують різні підходи до створення таких систем, такі як системи на основі словників, системи на основі правил.

1.1.1 Системи на основі словників

Система на основі словників використовує словник з великою кількістю слів та словосполучень та їх синонімами. Система перевіряє, чи присутня конкретна шукана сутність з тексту, у словнику. Ідея полягає в тому, щоб знайти та ідентифікувати об'єкти у реченні та класифікувати їх за категоріям. Наприклад, якщо у є речення «01001, Київ, вул. Хрещатик, 22», то «01001» позначений як «индекс», «Київ» позначений як «місто», «вул. Хрещатик» як «вулиця», «22» як «номер будинку».

Одним із недоліків використання цього підходу є необхідність постійного оновлення набору словникових даних для ефективного функціонування моделі.

1.1.2 Системи на основі правил

Такі системи працюють на основі набору заздалегідь встановлених правил. Прикладами таких систем можуть стати: LaSIE-II [2] та FASTUS [3]. Використовуються два основних набори правил:

– правила на основі шаблонів – такі системи зазвичай основані на ретельно розроблених вручну правил або регулярних виразах і часто поєднуються з пошук у спеціалізованих словниках. Регулярний вираз [4] — це послідовність символів, яка визначає шаблон пошуку в тексті. Цей метод полягає у використанні регулярних виразів для ідентифікації шаблонів, типових для поштових адрес;
– контекстні правила – контекстні правила залежать від значення або контексту слова в документі. Наприклад, одне і те ж слово може мати різні значення залежно від контексту, у якому воно використовується. Контекстні правила можуть включати в себе умови, які враховують розташування слова в реченні, залежність від інших слів у реченні або в попередніх реченнях, а також інші параметри.

Існують програмні пакети для виявлення адрес засновані на правилах та шаблонах. Найпопулярніші з них usaddress [5] (бібліотека для вилучення та структурування поштових адрес США), ez-address-parser [6] (бібліотека для вилучення поштових адрес Канади) та PyAP [7] (бібліотеки для різних країн).

1.2 Методи з використанням машинного навчання

Машинне навчання для аналізу тексту передбачає використання алгоритмів машинного навчання та «вузького» штучного інтелекту для розуміння значення текстових документів. При використанні машинного навчання для аналізу тексту, моделі навчаються на тренувальних даних, де вказано, які слова або фрази є шуканими в даному тексті, наприклад, поштовими

адресами. Модель вчиться виявляти закономірності та шаблони, які допомагають їй розпізнавати поштові адреси у тексті. Процес навчання може включати в себе попередню обробку даних, вибір відповідних ознак або характеристик тексту, та використання алгоритмів машинного навчання для тренування моделі. По суті, роль машинного навчання та штучного інтелекту в обробці природної мови та аналітиці тексту полягає в тому, щоб покращити, прискорити й автоматизувати базові функції аналітики тексту, які перетворюють неструктурований текст у корисні дані. Існують такі набори інструментів для роботи з природною мовою за допомогою машинного навчання: OpenNLP (Apache OpenNLP) [8] та Stanford NLP [9].

Наразі існує багато бібліотек програмного забезпечення, які засновані на принципах машинного навчання. Найпопулярніші з них, які використовуються для пошуку поштових адрес: Libpostal [10] та Deepparse [10].

1.3 Змішані методи

Змішані методи виявлення адрес використовують комбінацію різних технік та алгоритмів для виявлення адрес. Цей підхід може включати в себе використання регулярних виразів, машинного навчання, а також різні методи аналізу тексту, такі як метод n-грам [12], метод аналізу синтаксичних залежностей [13], метод рухомого вікна [15].

Однією з переваг змішаного методу виявлення адрес – це можливість комбінувати різні підходи, що дозволяє збільшити точність та надійність виявлення адрес в різних ситуаціях. Комбінування різних підходів до виявлення адрес дозволяє використовувати сильні сторони кожного методу та компенсувати їхні недоліки. Змішаний метод також є більш гнучким та дає можливість настроїти комбінацію різних підходів в залежності від типу тексту.

2 РЕАЛІЗАЦІЯ МЕТОДУ РУХОМОГО ВІКНА ДЛЯ ПОШУКУ АДРЕС У ДОВГИХ ТЕКСТАХ

2.1 Огляд методів, що застосовуються у роботі

Для вилучення адрес в цій роботі буде використовуватися змішаний метод аналізу тексту, а саме поєднання методу рухомого вікна та можливостей бібліотеки Libpostal.

2.1.1 Метод рухомого вікна

Метод рухомого вікна [15] — це техніка, яка використовується в інформатиці, для отримання інформації з послідовних даних, таких як текст.

У контексті текстових даних вікно — це сегмент тексту фіксованого розміру, який зазвичай складається з кількох слів або символів. Метод рухомого вікна передбачає послідовне переміщення цього вікна по тексту для вилучення необхідної інформації, аналізу чи обчисленню над цією інформацією. Вікно переміщується в міру надходження нових даних. Спочатку аналіз проводяться над першими n екземплярами, після алгоритм переходить до наступних n екземплярів з заданим кроком. Крок вказує, наскільки елементів переміщується вікно під час переходу до наступного кроку. Наприклад, якщо крок ковзного вікна дорівнює 1, вікно переміщається на один елемент за кожен крок. Вибір кроку рухомого вікна залежить від конкретного аналізу та вимог до результатів. Такий процес продовжується до закінчення даних чи при виконанні деяких умов, наприклад, знайдений шуканий елемент, обчислений деякий параметр. Довжина вікна можна регулювати відповідно до конкретного завдання.

Для пошуку адреси за допомогою такого методу вікно повинно переміщуватися по тексту та вилучати з нього відповідну інформацію з кожного вікна, таку як назва вулиці, номер будинку, місто, індекс, штат.

2.1.2 Бібліотека Libpostal

Libpostal — бібліотека для розбору/нормалізації поштових адрес, написана мовою C.

Бібліотека призначена для автоматичної обробки та нормалізації адрес представлених різними мовами та в різних форматах.

Основні функції Libpostal:

– розбір адреси на її складові (вулиця, місто, регіон тощо),
– нормалізація адреси (приведення адрес до стандартного формату, транслітерація).

Libpostal використовує методи машинного навчання, весь процес тренування моделей відкритий. Автори бібліотеки [16] відзначають, що в роботі бібліотеки використовується алгоритм навчання усередненого персептрона.

Базовий алгоритм [17] використовує просту процедуру навчання, керовану помилками, тобто якщо поточні ваги передбачають правильну відповідь, вони не змінюються. Якщо припущення неправильне, тоді для кожної функції один додається до ваги правильного класу, а один віднімається від ваги прогнозованого/неправильного класу.

Оскільки оновлення ваги дуже рідкісні та відбуваються лише тоді, коли модель робить помилку, навчання відбувається дуже швидко.

Алгоритм усередненого перцептрон [18] є модифікованою версією базового алгоритму, який використовується для багатокласової класифікації. Основна відмінність між цими двома алгоритмами полягає у способі оновлення ваг моделі.

У звичайному персептроні, який базується на принципі спрямованого пошуку помилок, ваги оновлюються тільки в разі помилкової класифікації. Якщо модель правильно класифікує приклад, ваги залишаються без змін. Це може призводити до перенавчання, коли модель добре вивчає тренувальні дані, але погано працює на нових даних.

Усереднений персептрон розв'язує проблему перенавчання шляхом усереднення ваг з кількох ітерацій тренування. Після завершення тренування, ваги з усіх ітерацій усереднюються. Це сприяє поліпшенню стабільності моделі та допомагає уникнути перенавчання, оскільки надлишкові коливання ваг відсікаються.

Усереднення ваг дозволяє отримати більш узагальнену модель, яка має кращі показники работи на нових даних.

Бібліотека навчається на даних реального світу з проекту OpenStreetMap. OpenStreetMap некомерційний веб-картографічний проект зі створення силами спільноти учасників — користувачів Інтернету докладної, вільної та безкоштовної географічної карти світу. Учасники проекту збирають географічні дані з різних джерел, таких як повітряні фотознімки, GPS-дані та інші відкриті джерела. Вони редагують ці дані, додаючи дороги, будівлі, природні об'єкти, місця та багато іншого. Результатом їхньої спільної роботи є цифрова карта, яка доступна для використання всім користувачам.

Також використовуються для навчання моделі дані з проектів GeoNames, Quattroshapes, Zetashapes.

В роботі Libpostal використовує низку словників для кожної мови, які є простими текстовими файлами та мають в собі загальноприйняті скорочення та абревіатури. Словники компілюються в структуру даних префіксне дерево, після чого використовується швидкий алгоритм пошуку для сканування рядка та вилучення відповідних фраз.

Функції бібліотеки викликаються через інтерфейс, написаний різними мовами програмування. В даній роботі розглядається взаємодія з бібліотекою мовою Python.

Найчастіше використовуються дві функції бібліотеки: libpostal_parse_address, expand_address. Призначення та вхідні, вихідні данні функцій наведені в Таблиця 2.1.

Таблиця 2.1 – Функції бібліотеки Libpostal

	Назва функцій	
	libpostal_parse_address	expand_address
Призначення	призначена для отримання елементів адреси.	призначена для нормалізації адрес.
Вхідні дані	рядок з адресою, яку потрібно опрацювати.	рядок з адресою, яку потрібно опрацювати.
Вихідні дані	словник Python, що містить ключі та значення для кожного із вилучених елементів адреси.	список рядків, кожен з яких є одним з можливих варіантів повністю розширеної та нормалізованої адреси на основі вхідного рядка

Приклад коду використання функції libpostal_parse_address наведений на Рисунок 2.1 – Приклад використання функції libpostal_parse_address.

```
from postal.parser import parse_address

text = "Запорізький національний університет, 69600, Запоріжжя, вул. Жуковського, 66"

parsed_addresses = parse_address(text)
print(parsed_addresses)
```

Рисунок 2.1 – Приклад використання функції libpostal_parse_address

У прикладі вхідними даними є адреса "Запорізький національний університет, 69600, Запоріжжя, вул. Жуковського, 66". Після застосування функції libpostal_parse_address, програма повертає результат у вигляді списку кортежів, де кожен кортеж містить два елементи: розпізнаний компонент адреси та його тип.

Отримані вихідні дані для даного прикладу будуть наступними: [('запорізький національний університет', 'house'), ('69600', 'postcode'), ('запоріжжя', 'city'), ('вул. жуковського', 'road'), ('66', 'house_number')].

Значення "house" вказує на те, що перший компонент адреси є назвою будівлі (Запорізький національний університет), "postcode" - поштовий індекс (69600), "city" - назва міста (Запоріжжя), "road" - назва вулиці (вул. Жуковського), "house_number" - номер будинку (66).

Приклад коду використання функції expand_address наведений на Рисунок 2.2 – Приклад використання функції expand_address.

```
from postal.expand import expand_address

text = "Запорізький національний університет, 69600, Запоріжжя, вул. Жуковського, 66"

parsed_addresses = expand_address(text)
print(parsed_addresses)
```

Рисунок 2.2 – Приклад використання функції expand_address

У даному прикладі, вхідні дані містять адресу: "Запорізький національний університет, 69600, Запоріжжя, вул. Жуковського, 66". Після застосування функції expand_address, програма повертає список розширених варіантів цієї адреси.

Отримані вихідні дані мають наступний вигляд: ['запорізькии національнии університет 69600 запоріжжя вулиця жуковського 66', 'zaporizkyy natsionalnyy universytet 69600 zaporizhzhya vulytsya zhukovskoho 66', 'zaporizkij nacionalnij universitet 69600 zaporizza vulytsya zukovskogo 66']. Цей список містить розширені та нормалізовані варіанти початкової адреси, представлені у різних форматах транслітерації.

Така функція дозволяє розширити абревіатури та назви в адресах до їхньої канонічної форми та представити адресу у різних форматах та мовах.

2.2 Реалізація методу

Для вилучення адрес було розроблено комплекс функцій, в тому числі и функція на основі методу рухомого вікна.

Повний текст програми наведений в main.py. Для зручності програма поділена на модулі. Модуль main.py є основним файлом програми, який координує взаємодію з модулем address_parser.py. Файл address_parser.py є модулем, який містить функціональність для виявлення поштових адрес.

Мета програми полягає у виявленні адрес в довгих текстах та представлення їх в зручному форматі для подальшого використання.

Результатом роботи програми є асоціативний масив, який містить мітки елементів адреси та відповідні індекси знаходження цих елементів у тексті. Такий формат є зручним, оскільки він надає легкий доступ до окремих елементів адреси. Наприклад, працювати тільки з назвами вулиць та номерами будинків, не витрачаючи ресурси на аналіз надлишкової інформації. Також цей формат дозволяє отримати доступ до тексту, що оточує знайдену адресу, що є корисним, якщо потрібно виявити контекст, в якому згадана адреса.

В програмі будуть використовуватися мітки елементів задані в бібліотеці Libpostal. Перелік міток та їх значення наведені у Таблиця 2.2 – Мітки елементів адреси та їх призначення

Таблиця 2.2 – Мітки елементів адреси та їх призначення

Мітки адреси	**Символ маскування**	**Опис**
category	a	Категорія, наприклад, "ресторани"
near	n	Фрази, такі як "у", "поруч"
house_number	h	Номери будівлі
road	r	Назви вулиці

unit	u	Квартира, офіс, ділянка або інші допоміжні позначення

Продовження таблиці 2.2

level	l	Номер поверху
staircase	s	Нумеровані / позначені буквами сходи
entrance	e	Нумерований / позначений буквами вхід
po_box	o	Скринька відділення поштового зв'язку
postcode	p	Поштові індекси
suburb	b	Зазвичай неофіційна назва району
city_district	d	Зазвичай райони або округи в межах міста, є офіційною назвою
city	c	Міста, села, будь-які населені пункти
island	i	Названі острови
state_district	q	Зазвичай адміністративний підрозділ другого рівня або округ
state	y	Адміністративний підрозділ першого рівня
country_region	g	Неформальний підрозділ країни
country	t	Назва країни
world_region	w	Позначення регіонів світу

2.2.1 Етапи роботи програми

Процес роботи програми можна розбити на такі етапи:

а) зчитування оброблюваного тексту з файлу;

б) попередня обробка тексту;

в) пошук адрес за допомогою Libpostal;

г) маскування значень

г) пошук індексів початку та кінця елементів адреси в тексті;

д) злиття інтервалів, що перекриваються;

е) запис отриманих результатів в файл.

Опис кожного етапу наведений нижче:

Зчитування оброблюваного тексту з файлу

Для зручності вхідними даними для програми є текстовий файл, в якому знаходяться тексти, із яких необхідно вилучити адреси.

Попередня обробка тексту

Функція "process_text" призначена для обробки тексту шляхом видалення знаків пунктуації, табуляцій, розривів рядків та інших символів.

У функції використовується модуль re [19]. Модуль Python "re" забезпечує підтримку регулярних виразів у мові Python. Після передачі тексту у якості аргументу до функції, відбувається видалення всіх знаків пунктуації, табуляцій, розривів рядків та інших символів за допомогою регулярного виразу "[^\w\s]". Далі, за допомогою регулярного виразу "\s+", здійснюється заміна множинних пробілів на одиночні пробіли.

Попередня обробка тексту, яка здійснюється за допомогою функції "process_text", є необхідною перед подальшим аналізом тексту. Така обробка допоможе зекономити час та ресурси алгоритму при подальшому аналізі. Видалення зайвих символів дозволяє зменшити обсяг тексту, з яким потрібно працювати.

Крім того, видалення зайвих символів допомагає зосередитися на значущих елементах тексту. Наприклад, виключення знаків пунктуації може запобігти помилковому визначенню слів як адреси. Це допомагає знизити кількість неправильно розпізнаних елементів під час аналізу тексту.

Пошук адрес за допомогою Libpostal

На цьому етапі застосовується функція parse_address з бібліотеки Libpostal, яка дозволяє вилучити та розбирати адреси з тексту. Результатом роботи є список кортежів, де кожен кортеж містить два елемента. Перший елемент у кожному кортежі – це конкретна частина адреси, а другий елемент – опис цієї частини адреси.

Наприклад, кортеж містить інформацію про місцезнаходження будинку в Лондоні, вказано номер будинку, назва вулиці, передмістя, місто: [('100-106', 'house_number'), ('leonard st', 'road'), ('shoreditch', 'suburb'), ('london', 'city').

Маскування значень

Значення отримані на попередньому кроці будуть масковані з використанням символів для позначення певних елементів адреси. Маскування проводиться за словником, який містить коди для різних компонентів адреси,

коди приведені в *Таблиця 2*. Кожен компонент адреси відповідає певному символ. Текст, що не є адресою позначається символом ‘0’.

Застосовуючи процес маскування, кожен елемент списку перетворюється на відповідної довжини послідовність символів. Повертаючись до прикладу з попереднього етапу результат маскування буде виглядати так.

Вхідні дані: [('100-106', 'house_number'), ('leonard st', 'road'), ('shoreditch', 'suburb'), ('london', 'city')].

Вихідні дані: nnnnnnn0rrrrrrrrrr0sssssssss0cccccc0

Пошук індексів початку та кінця елементів адреси в тексті

До маскованого рядка застосовується метод рухомого вікна, що реалізований в функції "find_addresses". Функція виділяє поточне вікно з рядка та знаходить унікальні символи в цьому вікні. Якщо кількість унікальних символів у вікні дорівнює або перевищує 3, то будемо вважати, що була знайдена поштова адреса (докладно в підрозділі 2.2.2). Для знайдених символів функція створює словник для збереження початкових та кінцевих індексів кожної групи унікальних символів.

В кінці функція видаляє ключ '0' з кожного словника у списку, щоб очистити записи, що не є адресами.

Функція повертає список словників, які містять початкові та кінцеві індекси кожного унікального символу, знайденого в кожному вікні.

Вхідні дані: nnnnnnn0rrrrrrrrrr0sssssssss0cccccc0

Вихідні дані: , [{'house_number': [0, 8]}, {'road': [9,19]}, {'suburb': [20, 31]}, {'city': [32, 39]}].

Злиття інтервалів

Далі використовується функція merge_intervals, яка призначена для об'єднання перекриваючих інтервалів у списку словників. Ця функція отримує на вхід список, який містить словники з інтервалами. Інтервали представлені у вигляді пар індексів, які повертає метод рухомого вікна. Метою функції є об'єднання таких інтервалів, щоб усунути перекриття та дублікати. Результатом роботи функції є новий список словників з об'єднаними діапазонами. Для

досягнення цієї мети функція реалізує алгоритм об'єднання інтервалів, псевдокод якого приведений на Рисунок 2.3 – Приклад використання функції expand_address

```
Функція злиття_інтервалів(arr):
    Для кожного і від 0 до довжини(arr):
        ключі = отримати список ключів з arr[i]

Для кожного j від і + 1 до довжини(arr):
     # Перевірка, чи присутній ключ в іншому елементі arr[j]
    Якщо ключ присутній в arr[j]:
        інший_початок = перший елемент arr[j][ключ]
        інший_кінець = другий елемент arr[j][ключ]

        # Перевірка, чи перекриваються інтервали
        Якщо початок <= інший_кінець і інший_початок <= кінець:
            # Об'єднання інтервалів
            початок = мінімальне значення між початком і іншим_початком
            кінець = максимальне значення між кінцем і іншим_кінцем

            # Оновлення значення інтервалу в arr[i]
            arr[i][ключ] = [початок, кінець]
    # Створення нового списку, що містить лише непорожні елементи
    злитий_список = []

          Повернути злитий_список
```

Рисунок 2.3 – Приклад використання функції expand_address

Алгоритм проходить через кожний елемент списку arr та перевіряє, чи є значення для певного ключа списком довжиною 2, тобто чи має значення індекс кінця та початку. Далі, якщо значення відповідає умові, алгоритм знаходить

початкове та кінцеве значення інтервалу та порівнює його з іншими елементами списку arr. У разі перекриття інтервалів, алгоритм зливає їх, оновлюючи початкове та кінцеве значення.

Запис отриманих результатів в файл

Заради зручності, результати роботи записуються у файл, що є особливо важливим при роботі з довгими текстами.

2.2.2 Особливості використання методу рухомого вікна для виявлення адрес

Мінімально поштова адреса складається з двох елементів: номер будинку та назва вулиці. Також у вікні може бути текст, який не є адресою, який теж враховується як елемент. Тому, якщо кількість елементів у вікні дорівнює або перевищує 3, то вважається, що це поштова адреса. Коли ми знаходимо адресу, функція рухомого вікна зберігає інформацію про його знаходження у тексті.

Важливий аспект при використані методу є розмір вікна, над яким будуть проведені операції. Визначення оптимального розміру вікна у методі рухомого вікна вимагає балансу між двома факторами: точністю аналізу та обчислювальною складністю. Занадто маленький розмір вікна може призвести до упущення важливої інформації. З іншого боку, занадто великий розмір вікна може сприяти збільшенню обчислювальної складності.

Для застосування методу рухомого вікна до довгих текстів з метою виявлення поштових адрес необхідно враховувати особливості таких адрес. Поштові адреси зазвичай мають певний формат, що включає такі елементи, як ім'я вулиці, номер будинку, поштовий індекс і країна. Припустимо, що вікна довжиною 100 символів буде достатньо для ефективного пошуку.

Крім того, слід врахувати також обчислювальні обмеження та продуктивність системи при виборі розміру вікна. Великі вікна можуть вимагати більше обчислювальних ресурсів для обробки текстових даних, особливо під час роботи з довгими текстами.

3 ПЕРЕВІРКА ЯКОСТІ МЕТОДУ

3.1 Відомості про вхідні дані

Вхідними даними методу виявлення поштових адрес є довгий текст, який потенційно містить адреси.

Під поняттям «довгий текст» в цій роботі мається на увазі, що довгий текст - це текстовий документ або фрагмент, який характеризується значним обсягом слів, речень або символів. Наприклад, статті, книги, наукові роботи, блогові пости, корпоративні звіти, документи та інші види текстів, які містять велику кількість інформації.

Довгий текст характеризується не тільки своїм обсягом, але й структурою, такий текст може включати багато розділів, підрозділів, абзаців та розділових знаків. Це може ускладнювати розуміння і обробку таких текстів, оскільки важко виявити ключову інформацію, встановити логічну структуру та зв'язки між різними частинами.

Тестовими вхідними даними є набір текстів, з'єднаних в один файл, взятих з веб-сторінок ресторанів, що входять в рейтинг "La Liste 2023" [20]. В рейтинг входять ресторани с різних країн тому, тексти на веб-сторінках представлені на різних мовах. Так у тестовому файлі є тексти англійською, французкою, іспанською, італійською, японською, нідерландською, німецькою. Важливим є перевірка методу саме на даних різними мовами. Проблема, пов'язана з використанням різних мов у тексті, полягає у варіації форматів поштових адрес. Кожна мова може мати власні конвенції і правила для вираження адрес, що ускладнює розробку універсального методу виявлення.

Як приклад, розглянемо порівняння форматів адрес в Німеччині та Японії. У Німеччині при написанні адреси використовується порядок, де спочатку зазначається назва вулиці, номер будинку, поштовий індекс та місто.

У Японії адреси починаються з поштового індексу, за яким слідує адресна стрічка, в якій зазначаються від великих до менших поділів (областей) місцезнаходження.

3.2 Проблеми, виявлені під час аналізу якості методу

Під час перевірки було помічено, що метод здатний виявити більшість поштових адрес, однак, він також робить невірне класифікування в деяких випадках, основним чином через недосконалу роботу функції parse_addresses з бібліотеки Libpostal. Ця функція часто помічала різні рядки з цифрами, наприклад, номери телефонів, дати або позначення часу, як частину адреси. 93,22 відсотків всіх неправильно визначених рядків були цифрами або виразами, що містять цифри.

Ця проблема може бути вирішена функцією, яка виконує попередню обробку тексту. З тексту необхідно вилучити частини, які розроблений метод виявлення адрес може помилково визначити частиною адреси. Для цього була реалізована функція, яка шукає цифрові вирази і видаляє їх з тексту. У програмі пошук таких рядків реалізований за допомогою регулярних виразів, написаних з використанням модуля re [19]. Регулярні вирази описують різні формати написання номерів телефонів, дат та позначень часу. Саме ці категорії метод найчастіше помилково визначає як адресу. Регулярні вирази використані в функції приведено на Рисунок 3.1 – Регулярні вирази для видалення числових виразів

3.3 Точність та повнота методу

Оцінювання якості було проведено за допомогою метрик точність та повнота. Комбінація цих показників дає повнішу картину про продуктивність моделі і допомагає визначити її сильні та слабкі сторони.

```
# позначення часу
\d{1,2}:\d{2}
\d{1,2}\s\d{2}
\d{1,2}:\d{1,2}\s(?:AM|PM)
\b\d{3,4}\b

# номери телефонів
\+\d{1,3}\s?\d{1,3}\s?\d{1,4}\s?\d{1,4}
\d{4,}
\+\d{2}\s?\(?\d\)?\s?\d{2}\s?\d{2}\s?\d{2}\s?\d{2}
\+\d{1,3}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}
\d{1,4}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}
\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}
\d{1,4}\s?ext\s?\d{1,4}
\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}
\(\d{1,4}\)\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}

# роки (наприклад, "1999" або "2023")
\b(?:1[789]\d{2}|20[01]\d|202[0-9])\b

# позначення дат (наприклад, "2 August" або "Oktober 23")
\d{1,2}\s(?:January|February|March|April|May|June|July|August|September|Octobe
r|November|December)
```

Рисунок 3.1 – Регулярні вирази для видалення числових виразів

Точність та повнота є двома ключовими метриками оцінки ефективності класифікатора в задачах обробки інформації. Вони дозволяють визначити його здатність правильно класифікувати об'єкти.

Точність визначає відсоток істино-позитивних результатів відносно суми істино-позитивних та хибно-позитивних результатів. Вона показує, наскільки часто класифікатор правильно визначає позитивний клас, уникнувши помилкових віднесень до цього класу.

Повнота визначає відсоток істино-позитивних результатів відносно суми істино-позитивних та хибно-негативних результатів. Вона показує, наскільки часто класифікатор визначає всі екземпляри позитивного класу.

Істино-позитивне рішення - це кількість екземплярів, які були правильно визнані класифікатором як позитивний клас.

Хибно-позитивне рішення - це кількість екземплярів, які були помилково визнані класифікатором як позитивний клас, тоді як насправді вони належать до негативного класу.

Хибно-негативне рішення - це кількість екземплярів, які були помилково визнані класифікатором як негативний клас, тоді як насправді вони належать до позитивного класу.

За отриманими даними (істино-позитивне рішення = 94, хибно-негативне рішення = 8, хибно-позитивне рішення = 33), обчислимо значення точності та повноти:

$$\text{Точність} = \frac{94}{94+33} \approx 0{,}7402,$$

$$\text{Повнота} = \frac{94}{94+8} \approx 0.9216.$$

Повнота становить близько 0.9216, а точність - близько 0.7402.

Таким чином, для обраних тестових даних і розміру вікна 100 символів, повнота становить приблизно 92.16%, а точність становить приблизно 74.02%.

3.4 Подальші кроки у вдосконаленні методу

Один зі способів покращення точності та повноти полягає в розширенні процесу передобробки тексту. Додавання більшої кількості регулярних виразів для різних форматів написання елементів, таких як номери, дати і т.д., може сприяти точнішому визначенню адресів.

Крім того, розроблення шаблонів пошуку для різних мов може допомогти у визначенні елементів з різними мовними особливостями. Наприклад, врахування особливостей граматики та правил написання дат і чисел для кожної мови може поліпшити точність визначення таких елементів.

Інша причина, що може призводити до помилок з боку бібліотеки Libpostal, - це неправильне кодування тексту. Деякі символи, відображені в неправильній системі кодування, можуть впливати на правильність розпізнавання та класифікацію елементів. Тому важливо контролювати кодування тексту і, за необхідності, коригувати її для усунення можливих проблем.

ВИСНОВКИ

У даній роботі було реалізовано метод виявлення поштових адрес у довгих текстах. Оцінювання якості методу було проведено за допомогою метрик точність та повнота. Використання цих метрик дозволило об'єктивно оцінити якість роботи методу та визначити його ефективність у виконанні поставленої задачі класифікації поштових адрес у тексті.

У результаті тестування методу на тестових даних та при розмірі вікна 100 символів були отримані наступні показники: точність класифікації становить 74.02%, а повнота - 92.16%.

Отже, на основі проведених оцінювань можна зробити висновок, що запропонований метод класифікації тексту демонструє задовільну точність та високу повноту. Однак, для подальшого покращення результатів необхідно додати нові методи передобробки тексту, зокрема, додати більше регулярних виразів для різних форматів написання елементів та розробити шаблони пошуку для різних мов. Також варто звернути увагу на контроль кодування тексту, щоб уникнути можливих помилок, пов'язаних з бібліотекою Libpostal.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jerry R. Hobbs, Ellen Riloff. Information Extraction. Handbook of Natural Language Processing. CRC Press. Florida. 2010. 511 p.
2. K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, H. Cunningham, Y. Wilks. Description of the LaSIE-II System as Used for MUC-7. Proceedings of a Conference Held in Fairfax. Virginia. 1998.
3. Hobbs, Jerry R., Douglas E. Appelt, John Bear, David J. Israel, Megumi Kameyalna and Mabry Tyson. FASTUS: A System for Extracting Information from Text. Human Language Technology. The Baltic Perspectiv. 1993.
4. Jurafsky, D., Martin, J. H.. Speech and Language Processing. Third Edition. Charter 2 : Regular expressions, text normalization, edit distance. 2023, 4 p.
5. Репозиторій "usaddress" на платформі GitHub. URL: https://github.com/datamade/usaddress (дата звернення: 20.05.2023).
6. Репозиторій "ez-address-parser" на платформі GitHub. URL: https://github.com/zehengl/ez-address-parser (дата звернення: 20.05.2023).
7. Репозиторій Pyap (Python address parse) на платформі GitHub. URL: https://github.com/vladimarius/pyap (дата звернення: 20.05.2023).
8. Apache OpenNLP Developer Documentation. Version 2.2.0. URL: https://opennlp.apache.org/docs/2.2.0/manual/opennlp.html#tools.namefind.recognition (дата звернення: 20.05.2023).
9. Stanford NLP Documentation. URL : https://nlp.stanford.edu/software/ (дата звернення: 20.05.2023).
10. Репозиторій Libpostal на платформі GitHub. URL : https://github.com/openvenues/libpostal (дата звернення: 20.05.2023).
11. Deepparse Documentation. URL: https://deepparse.org/ (дата звернення: 20.05.2023).
12. Jurafsky, D., Martin, J. H.. Speech and Language Processing. Third Edition. Charter 3 : N-gram Language Models. 2023. 31 p.

13. Steven Bird, Ewan Klein, Edward Loper. Natural Language Processing with Python. Charter 8 : Analyzing Sentence Structure. 2009. 310 p.

14. Jerry R. Hobbs, Ellen Riloff. Information Extraction. Handbook of Natural Language Processing. CRC Press. Florida. 2010. 533p.

15. Mayur Datar, Rajeev Motwani. Data Streams: Models and Algorithms. Chapter 8 : The Sliding Window Computation Model and Results. 2007. 152 p.

16. Statistical NLP on OpenStreetMap. URL: https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-b9d573e6cc86 (дата звернення: 20.05.2023).

17. Christopher M. Bishop. Pattern Recognition and Machine Learning. Chapter 4 : Linear Models for Classification. 2006. 192 p.

18. Michael Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing. EMNLP. 2002.

19. Official Python Documentation v.3.11.3. re – Regular expression operations. URL : https://docs.python.org/3/library/re.html (дата звернення: 20.05.2023).

20. La Liste 2023. URL: https://www.laliste.com/en/laliste/world (дата звернення: 20.05.2023).

ДОДАТОК А

Повний текст програми

main.py

```
from address_parser import process_addresses

input_file = 'text.txt'
output_file = 'result.txt'

process_addresses(input_file, output_file)
```

address_parser.py

```
from postal.parser import parse_address
import re

MARK = {
    'house': '0',
    'category': 'a',
    'near': 'n',
    'house_number': 'h',
    'road': 'r',
    'unit': 'u',
    'level': 'l',
    'staircase': 's',
    'entrance': 'e',
    'po_box': 'o',
    'postcode': 'p',
    'suburb': 'b',
```

```
    'city_district': 'd',
    'city': 'c',
    'island': 'i',
    'state_district': 'q',
    'state': 'y',
    'country_region': 'g',
    'country': 't',
    'world_region': 'w'
}


def process_addresses(input_file, output_file):
    with open(input_file, 'r', encoding='utf-8') as file:
        text = file.read()

    try:
        processed_text =  process_text(text)

        remove_num_patterns = remove_patterns(processed_text)

        parsed_addresses = parse_address(remove_num_patterns)

        masked_addresses = mask_values(parsed_addresses, MARK)

        window_size = 100
        found_addresses = find_addresses(masked_addresses, window_size)

        merged_intervals = merge_intervals(found_addresses)

        with open(output_file, 'w', encoding='utf-8') as file:
```

```
            file.write(str(merged_intervals))

        except Exception as e:
            print(f"An error occurred: {str(e)}")


    def remove_patterns(text):

        text = re.sub(r'\d{1,2}:\d{2}', '', text)# 12:00
        text = re.sub(r'\d{1,2}\s\d{2}', '', text)# 12 00
        text = re.sub(r'\d{1,2}:\d{1,2}\s(?:AM|PM)', '', text)# 12:00 PM
        text = re.sub( r'\b\d{3,4}\b', '', text)# 1200

        text = re.sub(r'\+\d{1,3}\s?\d{1,3}\s?\d{1,4}\s?\d{1,4}', '', text)
        text = re.sub(r'\d{4,}', '', text)
        text = re.sub(r'\+\d{2}\s?\(?\d\)?\s?\d{2}\s?\d{2}\s?\d{2}\s?\d{2}', '', text)
        text = re.sub(r'\+\d{1,3}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}', '', text)
        text = re.sub(r'\d{1,4}\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}', '', text)
        text = re.sub(r'\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}', '', text)
        text = re.sub(r'\d{1,4}\s?ext\s?\d{1,4}', '', text)
        text = re.sub(r'\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}-\d{1,4}', '', text)
        text = re.sub(r'\(\d{1,4}\)\s?\d{1,4}\s?\d{1,4}\s?\d{1,4}', '', text)

        text = re.sub(r'\b(?:1[789]\d{2}|20[01]\d|202[0-9])\b', '', text)

        text                                                                    =
re.sub(r'\d{1,2}\s(?:January|February|March|April|May|June|July|August|September|
October|November|December)', '', text)

        return text
```

```
def process_text(text):
    text = re.sub(r'[^\w\s]', '', text)

    text = re.sub(r'\s+', ' ', text)

    return text.strip()



def mask_values(addresses, mask_dict):
    masked_values = ''
    for text, label in addresses:
        if label in mask_dict:
            masked_values += mask_dict[label] * len(text) + '0'
    return masked_values



def find_addresses(text, window_size):
    found_addresses = []
    for i in range(len(text) - window_size + 1):
        window = text[i:i + window_size]
        unique_chars = set(window)
        if len(unique_chars) >= 3:
            address = {}
            for j in range(i, i + window_size):
                if j > i and text[j] == text[j - 1]:
                    address[text[j]] = [address[text[j]][0], j]
                else:
                    address[text[j]] = [j, j]
```

```
            if address not in found_addresses:
                found_addresses.append(address)
            for dictionary in found_addresses:
                dictionary.pop('0', None)

        decrypted_addresses = []
        for address in found_addresses:
            decrypted_address = {next((k for k, v in MARK.items() if v == label), None): interval
                                 for label, interval in address.items() if next((k for k, v in MARK.items() if v == label), None)}
            decrypted_addresses.append(decrypted_address)

        return decrypted_addresses



    def merge_intervals(arr):
        for i in range(len(arr)):
            keys = list(arr[i].keys())

            for key in keys:
                if isinstance(arr[i][key], list) and len(arr[i][key]) == 2:
                    start = arr[i][key][0]
                    end = arr[i][key][1]

                    for j in range(i + 1, len(arr)):
                        if key in arr[j] and isinstance(arr[j][key], list) and len(arr[j][key]) == 2:
                            other_start = arr[j][key][0]
                            other_end = arr[j][key][1]
```

```
                    if start <= other_end and other_start <= end:
                        start = min(start, other_start)
                        end = max(end, other_end)

                        arr[i][key] = [start, end]
                        del arr[j][key]
merged_list = [element for element in arr if element]

return merged_list
```