Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНА РОБОТА

Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДРІПЗ. 180136.01.02.ПЗ

Виконав студент 2 курсу група ІПЗм-20-1 _____ А.І. Біловол
Підпис Ініціали, прізвище

Керівник д.т.н. проф. _____ О.В. Бармак
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн.наук, доцент _____ Ю. В. Форкун
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

14 грудня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
01 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Біловолу Андрію Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique

Керівник проєкту (роботи) Бармак Олександр Володимирович д.т.н.,проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 25.08.2021 р. № 102
2. Строк подання студентом проєкту (роботи) на кафедру 01.12.2021 р.
3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи досліджуваної проблеми
2. Аналіз методу очищення даних на основі Matching Dependency Technique
3. Розробка програмного засобу
4. Дослідження та оцінка ефективності удосконалення методу очищення даних на основі Matching dependency Technique
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Гурман І. В., доцент	13.12.21	13.12.21
Нормоконтроль	Форкун Ю. В., доцент	29.12.21	3.12.21

7. Дата видачі завдання «01» вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1 Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження;	01.09 - 07.09.2021	
2 Робота над розділом 1 дипломної роботи – вивчення літературних та Інтернет-джерел; аналіз відомих моделей, методів та засобів за темою роботи; висновки до розділу та постановка задач дослідження	08.09 - 25.09.2021	
3 Робота над розділом 2 дипломної роботи – розробка моделей, методів та алгоритмів вирішення задачі; висновки до розділу	26.09 - 10.10.2021	
4 Робота над науковими публікаціями	11.10 - 20.10.2021	
5. Робота над розділом 3 дипломної роботи – проектування архітектури системи для вирішення задачі, розробка вимог.	11.10 - 26.10.2021	
6 Робота над розділом 4 дипломної роботи – Опис оцінки та дослідження; висновки до розділу	27.10 - 15.11.2021	
7 Оформлення пояснювальної записки згідно вимог чинних стандартів	16.11 - 30.11.2021	
8 Попередній захист дипломної роботи	17.11.2021	
9 Перевірка роботи на наявність плагіату; норм контроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 - 04.12.2021	
10 Підготовка до захисту дипломної роботи	05.12 - 08.12.2021	

Студент _____ А.І. Біловол
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ О. В. Бармак
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique».

Автор роботи: Біловол Андрій Ігорович.

Керівник роботи: Бармак Олександр Володимирович.

Пояснювальна записка: 119 с., 33 рис., 10 табл., 3 дод., 22 джерел.

ОЧИЩЕННЯ ДАНИХ, ВІДПОВІДНІСТЬ ЗАЛЕЖНОСТЕЙ, ГІБРИДНИЙ АЛГОРИТМ, PYTHON.

Об’єкт дослідження – виявлення та відновлення брудних даних в процесі очищення даних.

Предмет дослідження – метод очищення даних на основі техніки відповідності залежності.

Мета дипломної роботи – виявлення та виправлення помилкових записів у структурованому наборі даних та вдосконалення методу очищення даних на основі відповідності залежностей та створення засобу, необхідного для застосування та реалізації техніки.

У роботі використані наступні методи дослідження та апаратура:

– експеримент, порівняння, абстрагування, аналіз;

– інструментальні засоби проектування, програмування та оцінки;

– персональний комп’ютер.

У процесі дипломного проектування було досліджено галузь якості даних, проблеми виявлення та відновлення даних. Виявлено що методика відповідності залежності є дуже перспективною для процесу очищення даних, на основі робіт з машинного навчання та методів вивчення очищення даних. Областю застосування було обрано адреси, які будь-який користувач, що проживає в будь-якій країні, буде вводити в Інтернеті.

На основі виявлених проблем методу, проаналізовано ряд алгоритмів пошуку залежностей. На їх основі було запропоновано гібридний метод пошуку відповідних залежностей. Втілена техніка машинного навчання, яка дозволила

використовувати «брудні дані», як дійсні, для перевірки інших записів для того самого адресу.

Проведені емпіричні дослідження доводять що процес очищення набору даних став легшим, оскільки впроваджена методика та вдосконалення алгоритму пошуку, та втіленню техніки машинного навчання для перевірки вхідних даних, ефективніше виявляє нечисті набори даних, а потім дає відповідні можливі рекомендації.

Підпис

1.12.2021
Дата

ABSTRACT

Master’s thesis: « Improving the method and means of data cleaning based on Matching Dependency Technique».

Author: Bilovol Andrii.

Head of research: Barmak Oleksandr.

Master’s thesis consists of: 119 p., 33 pc., 10 tb., 3 add., 22 srs.

DATA PURIFICATION, MATCHING DEPENDENCY, HYBRID ALGORITHM, PYTHON.

The object of research is the detection and recovery of dirty data in the process of data cleaning.

The subject of research is a method of data purification based on the dependence compliance technique.

The purpose of the thesis is to identify and correct erroneous entries in the structured data set and improve the method of data cleaning based on the compliance of dependencies and create the tools necessary for the application and implementation of technology.

The following research methods and equipment are used in the work:

– experiment, comparison, abstraction, analysis;
– tools for design, programming and evaluation;
– personal computer.

In the process of diploma design the field of data quality, problems of data detection and recovery were studied. It was found that the method of compliance of dependence is very promising for the process of data cleaning, based on machine learning and methods of studying data cleaning. The scope has chosen the addresses that any user living in any country will enter on the Internet.

Based on the identified problems of the method, a number of algorithms for finding dependencies are analyzed. Based on them, a hybrid method of finding matching dependencies was proposed. Implemented machine learning technique, which allowed the use of "dirty data" as valid, to verify other records for the same address.

Empirical studies show that the process of cleaning the data set has become easier, as the method and improvement of the search algorithm, and the implementation of machine learning techniques to verify input data, more effectively detects dirty data sets, and then gives appropriate recommendations.

Signature

01.12.2021
Date

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

MD – Matching Dependency

ОЦ – Обмеження цілісності

FD – Functional Dependencies

CFD – Conditional Functional Dependencies

ЛЧ – Ліва частина класифікаторів подібності

ПЧ – Права частина класифікаторів подібності

ВСТУП

Необхідність в автоматизації та діджиталізації торкнулась майже всіх сфер людського життя і ця потреба лише продовжує зростати. Проблеми з якістю зустрічаються в окремих наборах даних, такі як файли та бази даних, наприклад, через помилки в написанні під час введення даних, відсутність інформації або інші недійсні дані.

Орфографічні помилки, неправильний вибір значення та інші причини можуть призвести до таких помилок. Найпоширеніші введені значення будь-якою людиною в наш час - це їхні імена, номери телефонів та адреси під час реєстрації на будь-якому веб-сайті, де ім’я людини не можна стандартизувати. Одна і та ж назва може бути написана кількома способами відповідно до звуку. Наприклад, "вулиця Хрещатик" - це назва вулиці у Києві, яка може бути написана як "вулиця Хрищатик" або "вулиця Крещатик", оскільки вона буде звучати подібно до оригінальної вулиці.

У сучасному цифровому суспільстві від людей часто вимагається вводити адресу свого будинку чи офісу у формах, доступних в Інтернеті. Нерідкі випадки, коли люди вводять деякі незначні помилки, такі як неправильно написані адреси або неправильні поштові індекси. Такі помилки, допущені користувачем, можуть бути досить проблематичними, коли автоматизовані системи повинні обробляти їх запити. Наприклад, якщо людина замовляє щось в Інтернеті, вказавши неправильний поштовий індекс у введеній адресі, ця помилка може призвести до затримки доставки товару або ще гірше, що товар може залишитися недоставленим.

Щоб уникнути таких ситуацій, ці системи часто використовують техніку машинного навчання під назвою «Відповідність залежностей», яка виявилася корисною у складанні рекомендацій щодо виправлення будь-якого неправильного значення у вхідних даних.

Відповідність залежностей нещодавно було введене як декларативні правила для очищення даних та вирішення сутності. Застосування

відповідності залежності від екземпляра бази даних визначає значення деяких атрибутів для двох кортежів за умови, що значення деяких інших атрибутів достатньо схожі. Дана технологія має зрозумілу проблематику і результати, що дозволяє використовувати її у комплексі разом з іншими технологіями.

Отже, існує потреба в розробці та вдосконаленні методу очищення даних на основі техніки відповідності залежностей з використанням машинного навчання, який буде реалізований у вигляді програмного засобу для очищення наборів даних, які можуть містити помилки в адресах.

Актуальність теми роботи полягає у потребі розробці вдосконаленого методу очищення даних на основі техніки відповідності залежностей та засобу який його реалізує.

Мета дослідження – виявлення та виправлення помилкових записів у структурованому наборі даних та вдосконалення методу очищення даних на основі відповідності залежностей та створення засобу, необхідного для застосування та реалізації.

Поставлена мета досягається розв’язанням таких основних задач:

- дослідити методи очищення даних;
- описати уже існуючі алгоритми очищення даних, виділити наявні проблеми та шляхи покращення;
- охарактеризувати структуру предметної області та базову модель методу на основі відповідності залежностей;
- на основі проведених досліджень визначити основні напрямки покращення методу очищення даних на основі залежностей;
- підвести підсумки про необхідність розробки засобу;
- провести практичну перевірку ефективності запропонованих нововведень у порівнянні з оригінальним методом;
- оцінити ступінь виконання поставлених завдань.

Об’єкт дослідження – виявлення та відновлення брудних даних в процесі очищення даних.

Предмет дослідження – метод очищення даних на основі техніки відповідності залежності.

Наукова новизна отриманих результатів: удосконалено техніку пошуку відповідних залежностей за рахунок гібридного алгоритму пошуку залежностей, що дозволило зробити цю техніку ефективнішою, а також удосконалено метод очищення даних на основі відповідних залежностей за рахунок застосування більш розвиненої техніки машинного навчання, що дозволило використовувати погані дані, як дійсні дані для перевірки інших записів для тієї ж адреси.

Практична значимість отриманих результатів полягає у розробленні програмного засобу очищення даних, який ілюструє як ми можемо уникнути помилок у адресах введеним користувачем системи.

За темою і результатами дипломної роботи опубліковані тези доповіді на всеукраїнській науково-практичній конференції [1].

1ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області і виявлення наявних проблем та завдань

Виявлення та відновлення брудних даних є однією з постійних проблем в аналітиці даних, і невиконання цього може призвести до неточної аналітики та ненадійних рішень.

Люди стали свідками значного зменшення повсякденних фізичних зусиль за допомогою комп'ютерів. Іншими словами, ми з кожним днем стаємо все більш залежними від машин. У попередню епоху в школі, державному секторі, приватному секторі чи лікарні всі дані/записи раніше були фізичними або на паперах/файлах. Було б цілком природно помилятися, зберігаючи ці дані на файлі або папері. У сучасному світі всі рукописні та фізичні файли або перетворилися, або щодня перетворюються на м'які копії. Не було б несподіванкою очікувати помилок у результатах після того, як люди ввели дані в систему. Орфографічні помилки, неправильний вибір значення та інші причини можуть призвести до таких помилок.

Підприємствам стає все легше зберігати та отримувати великі обсяги даних. Ці набори даних можуть сприяти покращенню прийняття рішень, розширеній аналітиці та все більше надавати навчальні дані для машинного навчання. Однак якість даних залишається серйозною проблемою, а брудні дані можуть призвести до неправильних рішень та ненадійного аналізу. Приклади поширених помилок включають пропущені значення, помилки друку, змішані формати, повторювані записи однієї і тієї ж реальної сутності та порушення бізнес-правил. Аналітики повинні враховувати вплив брудних даних, перш ніж приймати будь-які рішення, і, як наслідок, очищення даних стало ключовою сферою дослідження бази даних [2, 3].

За останні кілька років спостерігається сплеск інтересу як промисловості, так і наукових кіл до проблем очищення даних, включаючи нові абстракції, інтерфейси, підходи до масштабованості та статистичні методи. Одним з ключових диференціюючих факторів є те, як визначити

помилку даних (тобто виявлення помилок). Кількісні методи, що значною мірою використовуються для виявлення викидів, використовують статистичні методи для виявлення ненормальної поведінки та помилок (наприклад, "заробітна плата, що відрізняється на три стандартні відхилення від середньої зарплати, є помилкою"). З іншого боку, якісні методи використовують обмеження, правила, шаблони для виявлення помилок (наприклад, «не може існувати двох співробітників одного рівня, той, хто знаходиться в Києві, заробляє менше, ніж той, хто не в Києві»). Як тільки помилки будуть виявлені, відновлення можна виконувати за допомогою сценаріїв, людського ресурсу чи експертів, або за допомогою їх поєднання.

Очищення даних також є однією з найважливіших проблем в управлінні, оскільки брудні дані часто призводять до неточних результатів аналізу даних і неправильних бізнес-рішень. Для виявлення помилок даних, правила якості даних або обмеження цілісності було запропоновано декларативний спосіб опису правильних екземплярів даних. Будь-яка підмножина даних, яка не відповідає визначеним правилам, вважається помилковою, що також називають порушенням. Були введені різні види методів відновлення даних з різними цілями, де алгоритми використовуються для виявлення підмножин даних, які порушують оголошені обмеження цілісності, і навіть для пропонування оновлення бази даних таким чином, щоб новий екземпляр бази даних відповідав цим обмеженням. Хоча деякі з цих алгоритмів спрямовані на мінімальну зміну бази даних, інші залучають експертів або бази знань для перевірки ремонту, запропонованого автоматичними повторюваними алгоритмами.

Ми зосередимося на аспекті узгодженості даних якості реляційних даних. Для забезпечення узгодженості даних часто використовуються правила якості даних. Використовують обмеження цілісності для вираження правил якості даних. Будь-яка частина даних, яка не відповідає заданому набору обмежень цілісності, вважається помилковою, також відомою як порушення цих обмежень. Дедуплікацію даних можна розглядати як застосування ключового обмеження, визначеного для всіх атрибутів реляційної схеми,

оскільки два повторюваних кортежі можна розглядати як порушення ключового обмеження. У цьому контексті очищення даних є вправою з виявлення помилок і, можливо, модифікації бази даних, щоб дані відповідали набору правил якості даних, виражених різними мовами. Ми розглянемо методи виявлення невідповідності даних, а також методи усунення невідповідності даних.

Оскільки очищення даних в основному складається з двох етапів: виявлення помилок і виправлення помилок, буде розглянуто різноманітні методи виявлення помилок, а також різні методики усунення помилок та розглянемо їх класифікацію.

1.2 Аналіз методів виявлення аномалій або помилок

Приклад методів для якісного виявлення помилок. Враховуючи брудний екземпляр бази даних, першим кроком є виявлення аномалій або помилок. Кожна техніка має вирішити три основні питання: “Що виявити”, “Як виявити” та “Де виявити”. Рисунок 1.1 ілюструє систематику якісного виявлення помилок.

Рисунок 1.1 – Класифікація якісних методів виявлення помилок

Вид помилки (Що виявити?). Якісні методи виявлення помилок можна класифікувати відповідно до того, який тип помилок фіксується. Іншими словами, які мови використовуються для опису закономірностей або обмежень екземпляру правових даних. Велика кількість робіт використовує обмеження цілісності, дробову логіку першого порядку, для фіксації правил якості даних, яким має відповідати база даних, включаючи функціональні залежності [4] та обмеження відмови [5] . Хоча повторювані записи можна вважати порушенням обмеження цілісності (ключового обмеження), усвідомлюється велика кількість робіт, які зосереджуються на цій проблемі, і обговорюють її як окремий тип помилки від інших типів обмежень цілісності.

Розробка таких обмежень цілісності або шаблонів вручну вимагає великого досвіду в галузі, і займає багато часу, методи автоматичного виявлення є істотними і були запропоновані для різних обмежень цілісності [6]. Класифікують методи відкриття обмежень цілісності та підходи, що керуються схемами, та підходи, керовані екземплярами.

Обмеження цілісності, які зазвичай оголошуються як частина схеми бази даних, все частіше використовуються як правила якості даних, або шляхом перевірки дійсності даних під час додавання або оновлення, або шляхом очищення брудних даних у різні моменти під час конвеєр обробки. Обмеження якості використовуються для виявлення порушень даних і автоматичного їх усунення відповідно до певної функції витрат, наприклад, мінімізація кількості змін до вихідних даних. Для управління якістю даних були запропоновані традиційні типи обмеження цілісності, такі як ключові обмеження, перевірки та функціональні залежності. Було виявлено, що умовні функціональні залежності, розширення функціональних залежностей, є більш загальними та виразними, оскільки вони можуть охопити багато проблем якості даних реального життя, які звичайні функціональні залежності не можуть охопити. Не дивно, що чим виразніша мова обмеження цілісності, тим складніше її використовувати в автоматизованих алгоритмах очищення даних або навіть у перевірці узгодженості. Отже, необхідно досягти балансу між

виразною силою ОЦ, щоб мати справу з більш широким простором бізнес-правил з одного боку, і розробкою ефективних алгоритмів очищення та виявлення з іншого боку. Оскільки власники даних часто не є експертами з якості даних, автоматичне виявлення обмеження цілісності є надзвичайно корисною функцією під час завантаження процесу очищення.

Дедуплікація даних, також відома як виявлення дублікатів, зв'язування записів, узгодження записів або розділення сутності, відноситься до процесу ідентифікації кортежів в одному або кількох відносинах, які посилаються на одну і ту ж реальну сутність. Ця тема широко висвітлювалася в багатьох опитуваннях: деякі спрямовані на надання широкого огляду всіх етапів дедуплікації даних, деякі зосереджені на дизайн метрик подібності, деякі обговорюють аспект ефективності дедуплікації даних, а деякі зосереджуються на тому, як консолідувати кілька записів.

Рисунок 1.2 ілюструє типовий приклад дедуплікації даних.

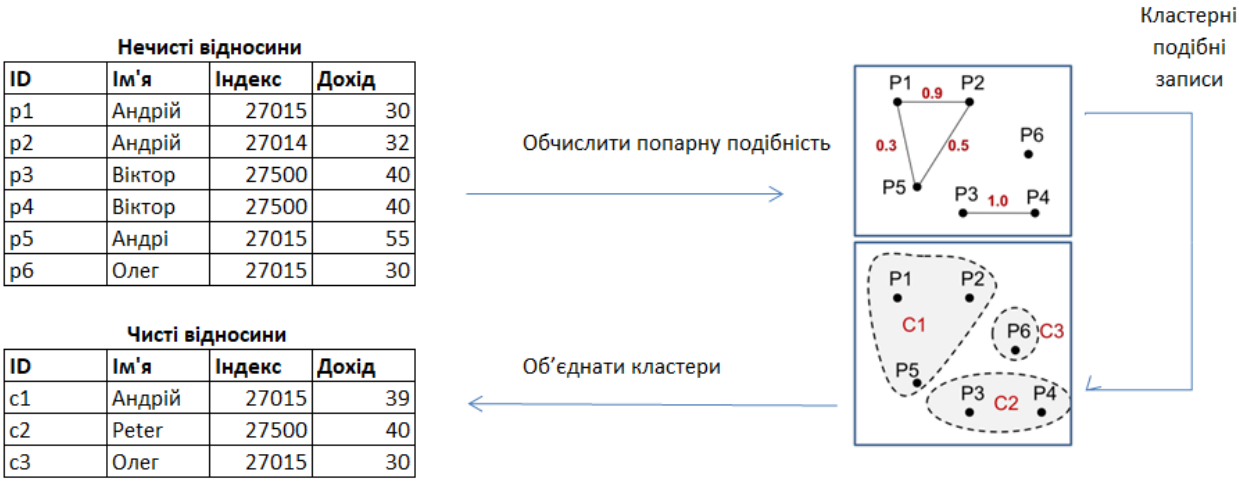


Рисунок 1.2 – Типове завдання дедуплікації

Подібність між парами записів обчислюється і відображається на графіку подібності (верхній правий графік на рисунку 1.2). Відсутні ребра між будь-якими двома записами вказують на те, що вони не є дублікатами. Потім записи об'єднуються в групи на основі графіка подібності. Припустимо, що користувач встановлює поріг рівним 0,5, тобто будь-які пари записів, які

мають подібність більше 0,5, вважаються дублікатами. Хоча записи P1 і P5 мають подібність менше ніж 0,5, вони об'єднані разом через транзитивність. Тобто обидва вони вважаються дублікатами запису P2. Усі записи в одному кластері об’єднуються в один запис в остаточному чистому відношенні.

Автоматизація (як розпізнати?). Класифікація на основі запропонованих підходів відповідно до того, чи залучаються люди до процесу виявлення помилок і як. Більшість методів є повністю автоматичними, наприклад, для виявлення порушень функціональних залежностей , тоді як інші методи включають людей, наприклад, для виявлення дублікатів записів. Для того, чи є два записи повторюваними, зазвичай потрібно нечітко зіставлення, для якого люди іноді можуть досягти кращої точності.

Розглянуто три приклади методик: цілісне очищення даних є прикладом техніки автоматичного виявлення порушень ОЦ, CrowdER використовує гібридний підхід «машина-людина» для підвищення точності дедуплікації даних, а Корлеоне є дедуплікацією даних.

Цілісна очищення даних автоматично виявляє порушення мультитипом ОЦ і захоплює клітини, які, швидше за все, будуть помилковими через повторювані порушення. Два порушення збігаються, якщо вони включають принаймні одну загальну клітину.

CrowdER використовує натовп співробітників, щоб вони допомогли у дедуплікації даних. Мотивація CrowdER полягає в тому, що, хоча автоматичні методи дедуплікації даних вдосконалюються, якість залишається далеко не ідеальною. Тим часом краудсорсингові платформи пропонують більш точний, але дорогий (і повільний) спосіб внести людське розуміння в процес.

Corleone. На відміну від CrowdER, який є гібридним людино-машинним підходом до дедуплікації даних, Corleone [7] є системою дедуплікації даних, яка повністю підключена до краудсорсингу, тобто не потрібно залучати розробників. Corleone є бажаним, оскільки підприємствам регулярно потрібно вирішувати проблеми від десятків до сотень завдань дедуплікації

даних. Залучення розробника до написання правил блокування та функцію відповідності для кожної дедуплікації завдання трудомістке і затратне.

Рівень бізнес-аналітики (де виявити?). Помилки можуть статися на всіх етапах стеку бізнес-аналітики, наприклад, помилки у вихідній базі даних часто поширюються через конвеєр обробки даних. Хоча більшість методів виявлення помилок виявляють помилки у вихідній базі даних, деякі помилки можуть бути виявлені лише набагато пізніше в конвеєрі обробки даних [7], де доступна додаткова семантика та бізнес-логіка, наприклад, обмеження щодо загального бюджету можуть бути застосовані лише після сукупності витрат.

Поширення помилок, виявлених у результатах перетворення, до джерел даних має важливе значення як для виправлення цих помилок, так і для запобігання їх повторному виникненню в майбутньому. Методи поширення помилок відрізняються залежно від типу передбачуваних перетворень даних, таких як аналіз причинно-наслідкових зв'язків, агрегація за числовими атрибутами та більш загальні запити SPJA.

Аналіз причинно-наслідкових зв'язків, який намагається обґрунтувати відповідальність джерела за спричинення помилок у результатах запиту, є інтуїтивним способом вирішення вищезгаданої проблеми. Робота в цій області моделює вихідну базу даних як набір змінних $X = \{X_1, X_2, \ldots, X_n\}$ і цільова база даних як інший набір змінних $Z = \{Z_1, Z_2, \ldots, Z_m\}$. Кожна вхідна змінна X_i приймає значення з дискретної або безперервної області. Кожна вихідна змінна Z_j є булевою змінною. Перетворення даних Φ_j для Z_j є булевим виразом над пороговими предикатами у вигляді X_i op c, де $X_i \in X$, op $\in \{<, \leq, =, \neq, \geq, >\}$, а c — постійне значення в область Xi. Наприклад, просте перетворення Φ_1 є $Z_1 = (X_1 > 10) \wedge (X_2 < 3)$. Нехай $\Phi = \{\Phi_1, \ldots, \Phi_m\}$ позначають m перетворень для m вихідних змінних. Подивіться, що Φ приймає вхідний вектор x значень для X і обчислює вихідний вектор z значень для Z. Нехай ˆz — основні значення істинності для Z. Враховуючи X, Z, Φ, x, z,ˆz, джерела помилок визначаються шляхом ранжирування

вхідних змінних X відповідно до того, наскільки кожна змінна сприяє помилці в z, що також називається відповідальністю X_i.

На рисунку 1.3 наведено зразок методів виявлення, які охоплюють усі категорії таксономії.

Методи:	Вид помилки (Що виявити?)		Автоматизація (як розпізнати?)		Рівень бізнес-аналітики (де	
	дублікати	обмеження цілісності	автоматичний	керується людиною	джерело	ціль
Functional dependencies value modification	✓		✓		✓	
Holistic data cleaning	✓		✓		✓	
CrowdER		✓		✓	✓	
Corleone		✓		✓	✓	
Causality Analysis	✓			✓		✓
Scorpion	✓			✓		✓
DBRx	✓		✓			✓

Рисунок 1.3 – Зразок методів виявлення помилок.

1.3 Аналіз методів та алгоритмів відновлення даних

Приклад засобів усунення помилок та відновлення даних. У приклад візьмемо екземпляр реляційної бази даних I схеми R і набір вимог до якості даних, виражених різними способами, відновлення даних відноситься до процесу пошуку іншого екземпляра бази даних I ', який відповідає набору вимог до якості даних. Ця проблема була широко вивчена, і на рисунку 1.4 зображена таксономія методів відновлення даних. Подібно до виявлення помилок, є три основні питання, які необхідно вирішити кожній техніці: "Що відремонтувати?", "Як відремонтувати?" та "Де відремонтувати?".

Відновлення цілі (Що відремонтувати?). Алгоритми ремонту роблять різні припущення щодо даних та правил якості: довіряючи оголошеним обмеженням цілісності, а отже, лише дані можна оновлювати для усунення помилок, повністю довіряючи даним та дозволяючи послабити обмеження , наприклад, для вирішення еволюції схеми та застарілих бізнес-правил, і

нарешті, вивчення можливості зміни як даних, так і обмежень [8]. Більшість методів виправляють дані лише стосовно одного типу помилок, тоді як інші нові методи розглядають взаємодію між кількома типами помилок і забезпечують цілісне відновлення даних.

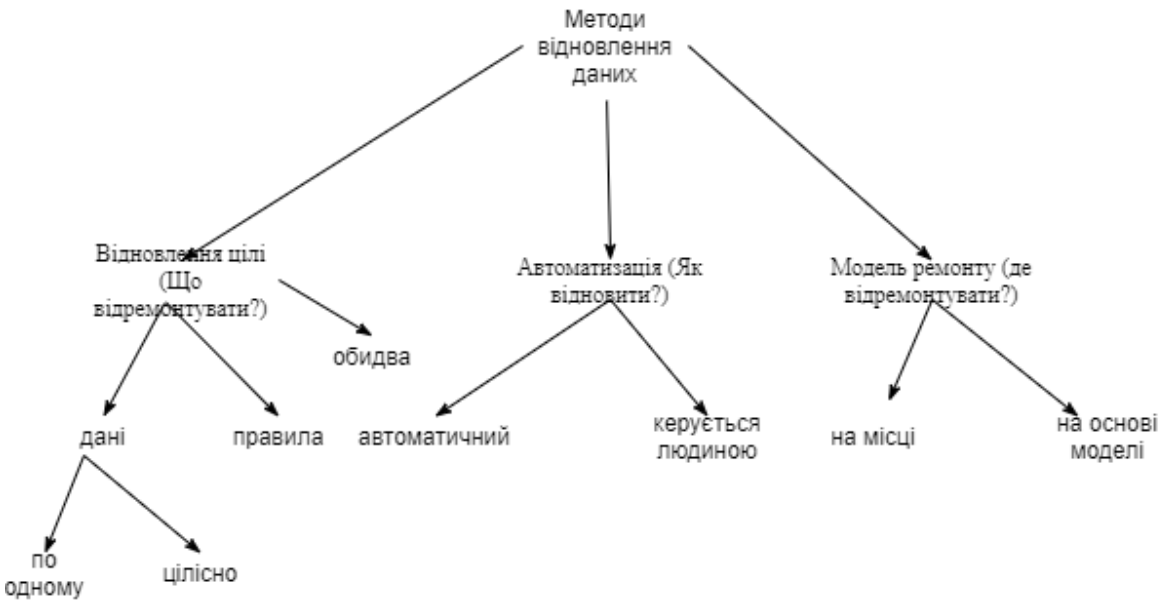


Рисунок 1.4 – Зразок методів відновлення даних.

Лише відновлення даних. Методи відновлення даних у цій категорії припускають, що існує набір обмеження цілісності Σ, визначених у схемі бази даних R, і будь-який екземпляр бази даних I з R повинен відповідати цим обмеженням. Запропоновані підходи спрямовані на зміну мінімальної кількості комірок у базі даних.

Комплексний ремонт. Методи відновлення лише даних роблять різні припущення про драйвер процесу відновлення. Було розглянуто різні типи помилок, які викликають виявлення аномалій даних. Для кожного типу помилок, таких як повторювані записи, пропущені значення та порушення функціональних залежностей, пропонується кілька алгоритмів виправлення. Наприклад, алгоритми розділення об’єктів, є прикладами методів, спрямованих на дублікати. Аналогічно, порушення обмежень умовних функціональних залежностей розглядалися в кількох пропозиціях, або

автоматично, або шляхом залучення людей до циклу. Більшість доступних рішень для відновлення даних належать до цієї категорії. Вони вирішують один тип помилок, або для забезпечення теоретичних гарантій якості, або для можливості масштабованої системи. Однак аномалії даних рідко виникають через один тип помилок, в одному наборі даних часто спостерігаються численні проблеми з якістю даних, такі як відсутні значення, друкарські помилки, наявність повторюваних записів і порушення бізнес-правил. Ці різнорідні типи помилок взаємодіють і конфліктують в одному наборі даних, і їх незалежне оброблення призведе до втрати можливості правильно визначити фактичні помилки в даних. Розглянуто пропозиції, які мають більш цілісний погляд на процес очищення даних, підходи до цілісного очищення [9]. Цілісні алгоритми виправлення розглядають порушення, що надходять від різних типів обмеження цілісності одночасно, і пропонують оновлення для відновлення основних даних. Наприклад, Chu et al.[9] розглянути широкий спектр обмежень цілісності, включаючи FD, CFD і DC, якщо порушення обмеження цілісності можуть бути закодовані як гіперребро в гіперграфі конфлікту. NADEEF, система очищення даних з відкритим вихідним кодом, яка надає користувачам інтерфейс для визначення власних правил якості даних, також використовує методи з для цілісного вирішення порушень. Geerts та ін.[10] розглядають усі обмеження, які можуть бути виражені як залежності, що породжують рівність. Fan[12] об’єднує відновлення даних на основі умовних функціональних залежностей та узгодження записів на основі відповідності залежностей і показує, що ці дві задачі вигідні один з одним, якщо вони поєднані разом.

Виправлення даних і правил. Методи очищення в цій категорії припускають, що дані та правила можуть бути забрудненими одночасно. Для екземпляра бази даних I та набору функціональних залежностей Σ таких, що I` ≠ Σ, нам потрібно знайти інші I` і Σ` , такі, що I` |= Σ`. На рисунку 1.5 показана таблиця з функціональними залежностями, в якій зазначено, що ім’я та прізвище визначають дохід. Є три порушення функціональних залежностей,

тобто перший і другий кортеж, третій і четвертий кортеж, а також п’ятий і шостий кортеж. Якщо функціональна залежність має бути повністю довіреним, необхідно змінити три клітинки, як показано в нижній лівій таблиці на рисунку 1.5. Якщо дані повністю довірені, до лівої сторони функціональних залежностей додаються два атрибути, показані в нижній середній таблиці на рисунку 1.5. Якщо функціональна залежність і дані мають однакову надійність, виправлення полягає у зміні лише одного значення комірки та додаванні одного атрибута до лівої сторони, як показано в нижній правій таблиці на рисунку 1.5.

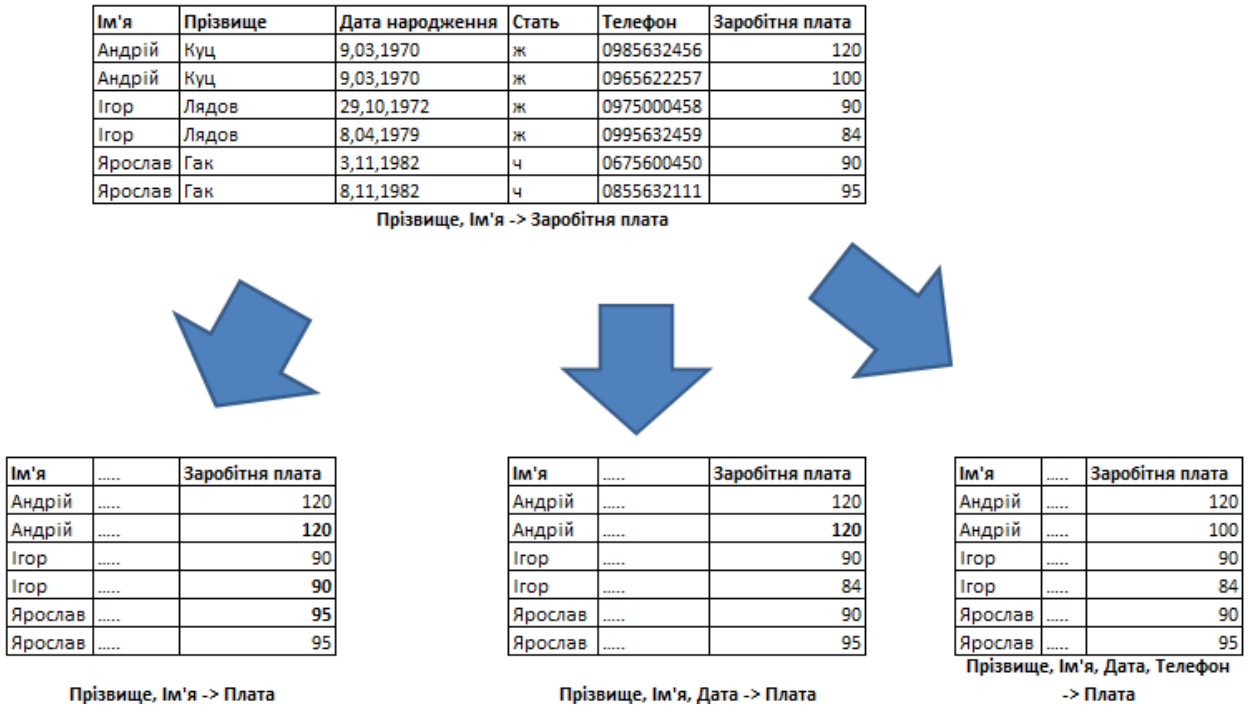


Рисунок 1.5 – відносна довіра до функціональних залежностей і даних

Автоматизація (Як відновити?). Класифікуються запропоновані підходи щодо інструментів, що використовуються в процесі ремонту. Більш конкретно, класифікуються сучасні підходи до ремонту залежно від того, чи залучаються люди і як. Деякі методи є повністю автоматичними, наприклад, шляхом зміни бази даних, так що відстань між вихідною базою даних I та зміненою базою даних I' мінімізується відповідно до певної функції витрат. Інші методи залучають людей до процесу ремонту або для перевірки

виправлень, для виправлення виправлень, або для навчання моделей машинного навчання для виконання автоматичних рішень щодо ремонту [11].

Автоматичний ремонт. Існують численні теоретичні дослідження та опитування щодо вивчення складності відновлення даних, параметризованих різними класами обмежень цілісності, такими як функціональні залежності, умовні залежності і обмеження відмов, і різних операцій відновлення, таких як значення оновлення та видалення кортежів. Автоматичні методи відновлення даних, спрямовані на оновлення бази даних таким чином, щоб відстань між вихідною базою даних I та модифікованою базою даних I` була мінімізована. За відсутності обґрунтованої істини головна гіпотеза, яка стоїть за цільовою функцією мінімальності, полягає в тому, що більшість бази даних чиста, і, таким чином, потрібно виконати лише відносно невелику кількість оновлень порівняно з розміром бази даних. На рисунку 1.6 показано кілька прикладів різних уявлень про мінімальний ремонт.

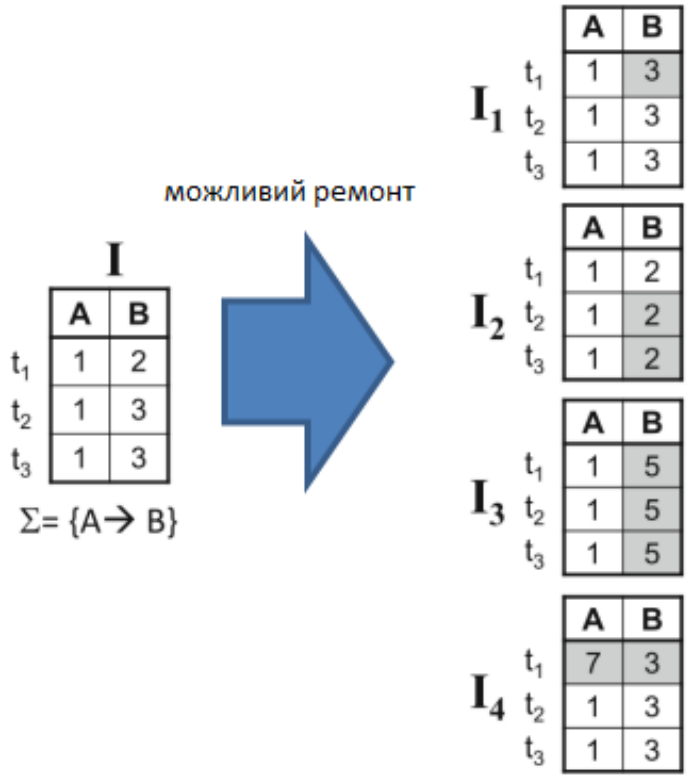


Рисунок 1.6 – Приклади різних видів відновлення.

Ремонт I1 є мінімальним за потужністю, тому що жоден інший ремонт не має менше змінених комірок. Виправлення I2 та I3 є мінімальними, оскільки повернення будь-якої підмножини змінених комірок до значень у I порушить A → B. I3, встановлюється мінімальним, оскільки повернення будь-якої підмножини змінених комірок до значень у I все одно порушить функціональні залежності. Ремонт I4 не є мінімальним, оскільки I4 все ще задовольняє A → B після повернення t1[A] до 1.

Ремонт під керівництвом людини. Методи автоматичного відновлення даних використовують евристики, такі як мінімальний ремонт для автоматичного відновлення даних на місці, і вони часто генерують неперевірені виправлення. Що ще гірше, вони навіть можуть внести нові помилки під час процесу. Часто важко, якщо взагалі неможливо, гарантувати точність будь-яких методів відновлення даних без зовнішньої перевірки за допомогою експертів і надійних джерел даних. Підхід прийнятий більшістю автоматичних алгоритмів очищення даних, мотивував нові підходи, які ефективно залучають людей або експертів до процесу очищення для створення надійних виправлень. Нижче наведено кілька прикладів: AJAX показує, як залучити користувачів до процесу очищення даних, змодельованого як орієнтований граф перетворень даних, Potter's Wheel — це інтерактивна система очищення даних, яка тісно інтегрує перетворення даних і виявлення розбіжностей, Data Tamer — це система контролю даних, яка залучає користувачів з різним досвідом на кількох етапах процесу курації.

AJAX — це фреймворк очищення даних, який розділяє логічний і фізичний рівні очищення даних. Логічний рівень підтримує розробку графіка потоку даних, який моделює перетворення даних, необхідні для очищення даних, тоді як фізичний рівень підтримує реалізації та оптимізацію перетворень даних. AJAX надає декларативну мову, яка є SQL, збагаченим набором конкретних примітивів, для вираження перетворень даних, таких як зіставлення двох записів. AJAX створює виняток, реалізований за допомогою

механізму винятків Java, щоразу, коли процес перетворення даних дає збій, і очікується, що користувачі вручну перевірять та вирішують винятки.

Potter's Wheel — це інтерактивна система очищення даних, яка тісно інтегрує перетворення даних і виявлення невідповідностей. Архітектура Поттерового колеса, бере в якості вхідних даних джерело даних. Дані, зчитувані з джерела вхідних даних, відображаються на масштабованому дисплеї електронних таблиць, що дозволяє користувачам інтерактивно повторно сортувати дані в будь-якому стовпці та прокручувати зразок представлення даних. Онлайн-пересортування підтримується онлайн-переупорядочувачем, який підтримує гістограму в стовпці сортування. Виявлення розбіжностей виконується на фоні нещодавно перетворених даних, і аномалії позначаються, як тільки вони виявлені. Користувачі досліджують ці позначені аномалії та вказують відповідні перетворення, які записуються механізмом трансформації. Зазначене перетворення застосовується до записів, які вже відображаються на екрані, а також тих записів, які використовуються для виявлення невідповідностей. Після того, як користувач буде задоволений послідовністю перетворень, Potter's Wheel може скомпілювати її як оптимізовану програму, яку можна застосувати до поточного набору даних або викликати в інших наборах даних.

Data Tamer — це система контролю даних, яка очищає та перетворює великомасштабні джерела даних на рівні підприємства. Data Tamer інтегрує схему та екземпляри цих джерел за допомогою серії вправ на відображення, дедуплікацію та зв'язування. Основна технологічна інновація Data Tamer — це автоматизація процесу керування даних із залученням різних ролей експертів із даними, включаючи власників даних, розпорядників даних, науковців та кураторів даних. Ця тісно поєднана конструкція машини та людини забезпечує практичне, наскрізне керування в масштабі від сотень до тисяч різнорідних наборів даних. Data Tamer є прикладом залучення користувачів до керування процесом очищення на кількох рівнях у стеку курації та на кількох рівнях деталізації.

Модель відновлення (де відремонтувати?). Класифікація запропонованих підходів відбувається на основі того, чи вони змінюють базу даних на місці, або створюємо модель для опису можливих ремонтів. Більшість запропонованих методів відновлюють базу даних на місці, таким чином руйнуючи базу даних. Для ремонту на місці часто будується модель, яка описує різні способи відновлення базової бази даних. На запити відповідають ці моделі, використовуючи, наприклад, вибірку з усіх можливих ремонтів та інші ймовірнісні механізми відповіді на запити [8].

Методи відновлення даних у цій категорії не створюють жодного відновлення для екземпляра бази даних, натомість вони створюють простір для можливого ремонту.

Імовірнісна дедуплікація. Бескалес та ін. вивчають проблему моделювання та запиту можливих ремонтів у контексті виявлення дублікатів, що є процесом виявлення записів, які посилаються на ту саму реальну сутність. На рисунку 1.7 показано вхідне відношення, що представляє вибіркові дані перепису, які, можливо, містять повторювані записи.

ID	Ім'я	Індекс	Дохід
p1	Андрій	27015	30
p2	Андрій	27014	32
p3	Віктор	27500	40
p4	Віктор	27500	40
p5	Андрі	27015	55
p6	Олег	27015	30

Невизначена кластеризація

Можливе виправлення

x1	x2	x3
{p1}	{p1,p2}	{p1,p2,p3}
{p2}	{p3,p4}	{p3,p4}
{p3,p4}	{p5}	{p6}
{p5}	{p6}	
{p6}		

Рисунок 1.7 – Імовірнісне виявлення дублікатів.

Алгоритми виявлення дублікатів генерують кластеризацію записів, де кожен кластер являє собою набір дублікатів, які в кінцевому підсумку об’єднуються в один репрезентативний запис на кластер. Підхід одноразового виявлення дублікатів визначає записи як дублікати або не повторювані на основі заданих специфікацій очищення. Отже, результатом є одна

кластеризація (відновлення) вхідного відношення (наприклад, будь-який з трьох можливих ремонтів, показаних на рисунку 1.7). Однак у імовірнісному підході виявлення дублікатів це обмеження пом’якшено, щоб урахувати невизначеність у виборі справжніх дублікатів. Результатом є набір кількох можливих кластеризацій.

На рисунку 1.8 наведено зразок методів відновлення даних, які охоплюють усі категорії таксономії.

Методи:	Відновлення цілі (Що відремонтувати?)				Автоматизація (Як відновити?)		Модель ремонту (де відремонтувати?)	
	дані - по одному	дані - цілісно	правила	обидва	автоматичний	керується людиною	на місці	на основі моделі
Functional dependencies value modification	✓				✓		✓	
FDs hypergraph	✓				✓		✓	
CFDs value modification	✓				✓		✓	
Holistic data cleaning		✓			✓		✓	
LLUNATIC		✓			✓		✓	
ecord matching and data repairing		✓			✓		✓	
NADEEF		✓			✓		✓	
Generate optimal tablaux			✓		✓		✓	
Unified repair				✓	✓		✓	
Relative trust				✓	✓		✓	
Continuous data cleaning				✓		✓	✓	
Potter’s Wheel	✓					✓	✓	
GDR	✓					✓	✓	
KATARA	✓					✓	✓	
DataTamer	✓					✓	✓	
Editing rules	✓					✓	✓	
Sampling FDs repairs	✓				✓			✓
Sampling Duplicates	✓				✓			✓

Рисунок 1.8 – Зразок методів відновлення даних.

1.4 Очищення даних із статистичної перспективи

Оскільки аналітика стає дедалі складнішою, важливо розуміти статистичні наслідки очищення даних. Розглянуто підходи, які або використовують методи машинного навчання для підвищення точності чи ефективності, також розглянуто вплив очищення на наступні числові запити.

Зосереджуються на дедуплікації (помилка дублювання), виправленні відсутніх і неправильних значень (помилка атрибута) та видалення помилкових або невідповідних даних (релевантна помилка). Спираючись на таксономію, представлену в попередній частині, буде розглянуто нещодавно запропоновані алгоритми та системи очищення даних на основі їх зв’язку зі статистикою.

Очищення даних зі статистикою. Існує кілька методів підвищення ефективності або точності алгоритмів очищення даних за допомогою статистичних методів, таких як машинне навчання.

Активне навчання в краудсорсингу. Краудсорсинг широко застосовується в промисловості для очищення даних . В академічних колах зростає консенсус щодо того, що натовп важко масштабувати, і в результаті в кількох останніх роботах використовується активне машинне навчання для визначення пріоритетності запитів до натовпу. Основна ідея полягає в тому, щоб сформулювати вхід людини для очищення даних як мітки для методики навчання під наглядом (наприклад, SVM або Random Forest), а Active Learning — це клас алгоритмів, які вибирають найбільш інформативні мітки.

Інші статистичні методи: Було також кілька робіт, які використовують статистичні методи для більш точного очищення забрудненої бази даних. Проект Eracer показав, як очищення даних про брудні відносини можна поставити як проблему навчання, що складається з двох кроків: спочатку вивчення графічної моделі для представлення відношення та алгоритм передачі повідомлень для вирішення невідповідностей .

Очищення даних для статистичного аналізу. У той час як більші набори даних можуть сприяти навчанню більш складних моделей машинного навчання, систематичні помилки даних, тобто пошкодження, які впливають на певні записи непропорційно, можуть зробити навчання моделей ненадійним. Неодноразово було виявлено, що проблеми машинного навчання дуже чутливі до брудних даних, навіть якщо використовуються надійні методи , а висока розмірність цих моделей призводить до неінтуїтивних ефектів при навчанні

після деяких типів процедур очищення даних. Наприклад, в одному прикладі передбачення шахрайства, просто об’єднання неузгоджених атрибутів перед навчанням моделі SVM підвищило істинну позитивну ймовірність виявлення з 62% до 91%. У цьому розділі підручника вивчається зв’язок між очищенням даних і подальшою аналітикою, а також проводяться опитування, які намагаються проаналізувати вплив очищення даних на аналітику.

Зведені запити: SampleClean зазначає, що для таких агрегатів, як сума, кількість і середнє значення, віддача очищення даних зменшується, і часто достатньо очистити невеликі вибірки даних, щоб оцінити результати з високою точністю. Ця проблема була додатково розширена для вивчення сукупних запитів на матеріалізовані уявлення.

Машинне навчання: SampleClean було розширено для вивчення очищення даних, яке передує навчанню моделі машинного навчання в системі під назвою ActiveClean . ActiveClean використовує методи вибору найцінніших даних і методи поступового оновлення моделей машинного навчання на основі нових чистих даних. Одним із цікавих висновків цієї роботи є те, що прогресивне очищення даних і навчання моделі не змінюються очікуваним чином. Припустимо, k N записів очищено, але всі інші брудні записи збережені в наборі даних. Агрегати над сумішами різних популяцій даних можуть призвести до помилкових зв’язків через добре відоме явище, яке називається парадоксом Сімпсона . Деякі з цих проблем не є очевидними в одновимірній аналітиці, як-от сума, підрахунок і середнє значення, але вони можуть призвести до незначних упереджень у високовимірному статистичному аналізі.

Адаптивний аналіз даних: нещодавно статистичне співтовариство вивчало деякі з цих проблем у галузі під назвою «Адаптивний аналіз даних. Такі поняття, як перевірка множинних гіпотез і швидкість помилкових відкриттів дуже актуальна для розробки інструментів аналізу . У цих роботах розглядається проблема помилкових відкриттів, коли аналітики виявляють неіснуючі тенденції в наборі даних через статистичну випадковість.

1.5 Висновки та постановка задачі

В розділі було здійснено деталізований аналіз літературних джерел та класифікації методів очищення даних, щоб визначити стан проблеми дипломної роботи.

Також у розділі досліджено методи виявлення та їх відновлення. Результати аналізу продемонстрували, що методика відповідності залежності є дуже перспективною для процесу очищення даних, оскільки результати були більш точними та ефективними [4].

Після огляду предметної області та здійснення детального аналізу було визначено, що проблеми, вирішувані в даній дипломній роботі є й досі актуальними. Виявлення та виправлення брудних даних є однією з багаторічних проблем у аналітиці даних, а їх невиконання може призвести до неточної аналітики та прийняття ненадійних рішень.

Було обрано адреси, які будь-який користувач, що проживає в будь-якій країні, буде вводити в Інтернеті. Тут дані зовнішніх джерел - це авторизовані та надійні дані, які використовувалися для перевірки та перевірки вхідних даних користувача. Відповідно до назви, дані введення користувача - це дані, введені будь-яким користувачем. Було взято до уваги назву вулиці, місто, область, країну та поштовий індекс для процесу очищення даних. Наприклад, якщо користувач при введенні адреси помилився у поштовому індексі, то цей алгоритм, виявить його, перевіривши його за допомогою зовнішнього джерела даних, і порекомендує всі можливі поштові індекси, які відповідають іншим значенням адреси, тобто , назва вулиці, місто, область та країна.

Іншою проблемою, є те, що на сьогоднішній день більшість систем очищення даних створені лише для англійської мови. Розробка системи очищення даних українською мовою потребує великих масивів українських даних. Доцільно використати для цього доступні відкриті дані. Дослідження поєднують розробку засобу та вдосконаленню методу виявлення помилок і усунення помилок для очищення даних на основі відповідності залежностей.

Метою є – виявлення та виправлення помилкових записів у структурованому наборі даних та вдосконалення методу очищення даних на основі відповідності залежностей та створення засобу, необхідного для застосування та реалізації.

Об’єкт дослідження – виявлення та відновлення брудних даних в процесі очищення даних.

Предмет дослідження – метод очищення даних на основі техніки відповідності залежності.

Для вирішення завдання необхідним є:

− охарактеризувати структуру предметної області та базову модель методу на основі відповідності залежностей;

− на основі проведених досліджень визначити основні напрямки покращення методу очищення даних на основі залежностей;

− підвести підсумки про необхідність розробки засобу;

− провести практичну перевірку ефективності запропонованих нововведень у порівнянні з оригінальним методом;

− оцінити ступінь виконання поставлених завдань.

2. АНАЛІЗ МЕТОДУ ОЧИЩЕННЯ ДАНИХ НА ОСНОВІ MATCHING DEPENDENCY TECHNIQUE

2.1 Метод очищення даних на основі відповідності залежностей

Концепція відповідних залежностей (Matching Dependency) нещодавно була запропонована для визначення правил відповідності для ідентифікації об’єктів. Подібно до функціональних залежностей (Functional Dependencies), MD також можна застосовувати до різних програм якості даних, таких як виявлення порушень.

Розглянувши недоліки та можливі напрями покращення існуючих рішень, переглянувши роботи з машинного навчання та методи вивчення очищення даних. Методика відповідності залежності здається дуже перспективною для процесу очищення даних, оскільки результати були більш точними та ефективними. Іншим методам вдалося виправити часткові помилки [4].

Відповідно до назви, метод відповідності залежності вимагає джерела даних, яке може бути використано як джерело з автентифікацією. Ці автентифіковані дані допомагають перевірити правильність введення, зіставляючи їх разом для процесу очищення. Метод бере в якості вхідного даних брудний набір даних. Щоб краще зрозуміти це, візьмемо приклад. Ми маємо зовнішнє автентифіковане джерело даних адрес для Кам’янець-Подільського, Хмельницької області, як показано у таблиці 2.1.

Таблиця 2.1 – Зовнішня автентифікована інформація, адрес

Автентифіковане джерело			
Адреса	Місто	Область	Індекс
проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
провулок Амбулаторний, 12	Кам'янець-Подільський	Хмельницька	32341
вулиця Заводська 3	Кам'янець-Подільський	Хмельницька	32307
вулиця Вишнева 8	Кам'янець-Подільський	Хмельницька	32302

Набори даних, введені користувачем, наведені у таблиці 2.2.

Таблиця 2.2 – Набір забруднених даних введений користувачем

Найменування фірми	Адреса	Місто	Область	Індекс
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Avto	проспект Грушевського, 9	**Кам'янець-Подольський**	Хмельницька	32301

Набори даних користувачів у таблиці 2 містять погані вхідні дані, які можна чітко побачити, порівнявши ці набори даних із зовнішнім джерелом даних у таблиці 1. Поштовий індекс у рядках 2 та 3 здається неправильним, назва підприємства та місто у рядку 4 також здаються неправильними. Першим кроком у робочому процесі методу є виявлення клітин у наборі даних користувача з потенційно неточними значеннями. Цей процес поділяє кортежі даних на шумні(забруднені) та чисті клітини. У нас є певні залежності, які можуть бути корисними в процесі очищення даних користувача за допомогою зовнішніх даних. Ці залежності показані на рисунку 2.1.

залежність1: Індекс = Авт.Індекс → Місто = Авт.Місту
залежність2: Індекс = Авт.Індекс → Область = Авт.Область
залежність3: Місто= Авт.Місто ∧ Область = Авт.Область ∧ Адреса = = Авт.Адреса → Індекс = Авт.Індекс

Рисунок 2.1 – Відповідні залежності

Тут відповідні залежності - це можливі сигнали, які допомагають виявляти та виправляти погані вхідні дані, порівнюючи їх із зовнішнім джерелом даних.

Відновленні та очищенні набори вихідних даних, які ми отримуємо за допомогою зовнішніх даних та їх можливих залежностей, мають правильне значення міста у рядку 4 та правильний поштовий індекс у рядках 2 та 3.

2.2 Основи методу очищення даних на основі техніки відповідності залежностей

Останнім часом якість даних стала гарячою темою в спільноті баз даних через величезну кількість «брудних» даних, отриманих з різних ресурсів. Ці дані часто містять дублікати, невідповідності та конфлікти через різні помилки, внесені людьми та машинами . На додаток до витрат на роботу з величезним обсягом даних, виявлення та видалення «брудних» даних вручну, безумовно, є непрактичним, оскільки запропоновані людиною методи очищення можуть знову викликати невідповідності. Тому залежності даних, які широко використовувалися в дизайні реляційної бази даних для встановлення обмежень цілісності, були для виявлення ширших невідповідностей у даних . Наприклад, розглянемо зв’язок контакти на схемі, яка зображена на рисунку 2.2.

Контакти (ІД, Ім’я, Код країни, Індекс, Місто, Вулиця)

Рисунок 2.2 – Зв’язок схеми контакти

Наступна функціональна залежність (FD) визначає обмеження, що для будь-яких двох кортежів у Контактах, якщо вони мають однаковий поштовий індекс, ці два кортежі також мають те саме місто. Останнім часом функціональні залежності були поширені на умовні функціональні залежності (CFD), тобто FD з умовами, які мають більшу виразну силу. Основна ідея цих розширень полягає в тому, щоб зробити FD, які спочатку застосовувалися для всієї таблиці, дійсними лише для набору кортежів. Наприклад, наступний CFD визначає, що лише за умови коду країни = 44, якщо два кортежі мають однаковий індекс, то вони також повинні мати ту саму вулицю, що зображено на рисунку 2.3.

Fd : [Індекс]->[Місто]

cfd : [Індекс, Код країни = 44]->[Вулиця]

Рисунок 2.3 – Умовні функціональні залежності

Ці обмеження залежностей можна використовувати для виявлення порушень даних [12]. Наприклад, ми можемо використовувати наведений вище FD для виявлення порушень у екземплярі контактів у таблиці 2.3. Для кортежів t5 і t6 з однаковими значеннями індексу = 21 вони мають різні значення міста, які потім виявляються як порушення наведений вище FD.

Таблиця 2.3 – Приклад відношення Контактів R

Номер страхування	Ім'я	Код країни	Індекс	Місто	Вулиця	
584	Чуб Олег	44	606	Київ	Центральна №2	t1
584	Чуб Олех	44	606	Київ	Центральна №2	t2
584	Чуб Оле	44	606	Київ	Центральна #2	t3
265	Андрій Гав	1	21	Львів	Центральна №3	t4
265	А. Гав	1	21	Львів	Центральна #3	t5
939	І. А. Гав	1	21	Київ	Центральна #3	t6

Хоча функціональні залежності (та їх розширення з умовами) дуже корисні для визначення невідповідності даних і виправлення «брудних» даних, вони перевіряють узгодження значень зазначеного атрибута на основі точної відповідності. Наприклад, за допомогою наведеного вище CFD, кортежі, які мають код країни = 44 і однакове значення в атрибуту індексу, будуть перевірені, щоб перевірити, чи вони точно відповідають значенням вулиці. Очевидно, це суворе обмеження точної відповідності обмежує використання FD і CFD, оскільки інформація в реальному світі часто має різні формати представлення. Наприклад, кортежі t2 і t3 в таблиці контактів будуть виявлені як «порушення» CFD, оскільки вони мають «різні» значення вулиці, але узгоджуються з індекс і код країни = 44. Однак «Центральна №2». і

«Центральна #2» є точно «одна й та сама» вулиця в реальному світі з різними форматами представлення. Щоб адаптувати залежності до цього реального сценарію, тобто бути толерантними до різних форматів представлення, Фан [12] запропонував нову концепцію залежностей даних, названу “Matching dependency”. Неформально залежність збігу націлена на нечіткі значення, такі як текстові атрибути, і визначає залежність між двома наборами атрибутів відповідно до їхньої якості збігу, виміряної деякими операторами збігу, такими як евклідова відстань і косинусна подібність.

Знову ж таки, у прикладі контактів на рисунку 2.4 ми можемо мати MD, які стверджують, що для будь-яких двох кортежів із контактів, якщо вони узгоджуються з атрибутом вулиці (відповідність, наприклад, косинусна подібність, для атрибута вулиця перевищує поріг 0,8), тоді відповідний атрибут місто також має збігатися (тобто подібність для міста більше, ніж відповідний поріг 0,7).

$$md1 : ([\text{вулиця}] > [\text{місто}], < 0.8, 0.7 >).$$

Рисунок 2.4 – Відповідні залежності

Подібно до методів, пов’язаних із FD, MD також можна застосовувати у багатьох завданнях. Наприклад, при очищенні даних ми також можемо використовувати MDs для виявлення неузгоджених даних, тобто дані не відповідають обмеженню (правилу), визначеному відповідними залежностями. Наприклад, згідно з наведеним вище прикладом на рисунку 2.4, для будь-яких двох кортежів t_i і t_j, які мають подібність більше 0,8 на атрибуті вулиця, вони також повинні бути зіставлені на атребуті місто (подібність ≥ 0,7). Якщо їхня подібність міста менше 0,7, то має бути щось не так в t_i і t_j , тобто невідповідність. Таку невідповідність текстових атрибутів неможливо виявити за допомогою функціональних залежностей та розширень на основі точної відповідності.

На додаток до визначення місцезнаходження неузгоджених даних, ідентифікація об’єкта, ще одна важлива робота для очищення даних, також може використовувати MDs як правила збігу. Наприклад, згідно з рисунком 2.5:

md2 : ([Ім’я, Вулиця] > [номер страхування], < 0.9, 0.9, 1.0 >).

Рисунок 2.5 – Кортеж залежностей

Якщо два кортежі мають високу схожість за назвою та вулицею (обидві подібності більші за 0,9), то ці два кортежі, ймовірно, позначають одну й ту ж людину в реальному світі, тобто має той самий код страхування. Насправді, з огляду на екземпляр бази даних, існує величезна кількість MD, які можна виявити, якщо ми встановимо різні пороги подібності для атрибутів. Зауважте, що якщо всі пороги встановлені на 1.0, MD мають ту ж семантику, що й традиційні FD, іншими словами, традиційні залежності є окремими випадками відповідних.

Далі буде розглянуто відповідні залежності та представлено основи, необхідні для виявлення MD. Розглянуто тривіальність та і мінімальні властивості MD, визначено частковий порядок на відповідні залежності. Розглянуто реляційний екземпляр показаний у таблиці 2.4. Щоб виявити всі MD в цьому реляційному екземплярі, використано три збіги стовпців, тобто кожен стовпець зіставляється з самим собою, з деякими специфічними для стовпців, але для цього прикладу нерелевантними показниками подібності. Приклад реляційного відношення екземплярів зображений на таблиці 2.4.

Таблиця 2.4 – Приклад відношення

id	A	B	C
1	a1	b1	c1
2	a2	b2	c2
3	a3	b2	c3
4	a4	b3	c2

Усі попарні подібності наведено в таблиці 2.5. Виділено три основні поняття: ідентифікатори записів (1, 2, 3, 4), значення стовпців (a_i , b_i , c_i) та подібності (0,0, 0,5, 0,7, 0,8, 1,0).

Таблиця 2.5 – Таблиця подібності

Записи	A	B	C
(1,2)	0,0	0,8	0,0
(1,3)	1,0	0,8	0,5
(1,4)	1,0	1,0	0,0
(2,3)	0,7	0,0	1,0
(2,4)	0,7	0,0	1,0
(3,4)	0,7	0,0	1,0

Основною ознакою відповідних залежностей, що відокремлює їх від функціональних залежностей, є їх здатність класифікувати значення як подібні чи відмінні. Два значення класифікуються як подібні за допомогою міри подібності та межі рішення. Міра подібності ≈ — це функція, яка визначає подібність двох значень за шкалою від 0,0 до 1,0, де 1,0 вказує на максимальну подібність (рівність), а 0,0 — на максимальну несхожість.

Межа рішення λ (або ρ) приймає рішення щодо двох значень, якщо вони подібні чи відмінні: подібність, що перевищує чи дорівнює λ, вважається подібною, а нижча подібність — неподібною. Отже, λ = 0,0 класифікує будь-які два значення як подібні. Межі рішення MD часто називають порогами MD. Міра подібності з межею рішення утворює так званий класифікатор подібності. Окрім введення поняття подібності для порівняння записів, MD також розширює визначення FD на два відношення: враховуючи два (потенційно однакові) відношення R і S і набір мір подібності ≈, MD визначає залежність від набору стовпців відповідає $C = \{C_1, C_2, ..., C_m\}$ над R і S, де $C_i = (A_i, B_i, \approx_i) \in R \times S \times \approx$. Набір збігів стовпців C зазвичай визначає однозначне відображення атрибутів у R на атрибути в S на основі заданого відображення схеми (або ідентичності, якщо R = S). Проте MD за визначенням можуть відповідати однаковим парам стовпців кілька разів з різними показниками

подібності або взагалі не відповідати певним стовпцям. Таким чином, виявлення відповідних залежностей можна розпочати з усіх можливих збігів стовпців, тобто C = R × S × ≈, і дозволити алгоритму виявлення автоматично знайти відповідні. Хоча це може бути корисно для інтеграції даних і використання відповідності схем, велика кількість можливих комбінацій значно впливає на продуктивність виявлення. Тому C визначається як підмножина, залежна від домену, з усіх можливих збігів стовпців, тобто C ⊂ R × S × ≈, і показуємо його як вхідний параметр процесу виявлення. Якщо r = s, C зазвичай відображає всі стовпці собі. Якщо r ≠ s, методи зіставлення схем можуть створювати збіги стовпців у C [13]. Зазвичай, домен збігу стовпця, природно, передбачає функцію подібності, таку як відстань редагування для рядків, відстань до маркера для тексту, числова відстань для чисел, тимчасова відстань для часу, календарна відстань для дат або евклідова відстань для координат. Якщо функція подібності для певного збігу стовпців неочевидна, ми визначаємо кілька збігів стовпців, по одному для кожної можливої функції подібності. Два поняття, класифікатор подібності та збіги стовпців, призводять до наступного формального визначення MD:

Дано набір збігів стовпців C ⊆ R × S × ≈ за відношеннями R і S та мірами подібності ≈, де Ci = (Ai, Bi, ≈i). Тоді MD φ визначається за формулою (1):

$$\bigwedge_{i=1}^{m} R[A_i] \approx_i, \lambda_i S[B_i] \rightarrow R[A_j] \approx_j, \rho_j S[Bj_j]. \quad (1)$$

Ліва частина (Left-hand side, ЛЧ) є поєднанням класифікаторів подібності стовпців. Кожен класифікатор подібності стовпців представлений у вигляді відповідності стовпця C_i з певною межею рішення $\lambda_i \in [0{,}0, 1{,}0]$. Множину меж рішення ЛЧ позначаємо як $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$.

Права частина (Right-hand side, ПЧ) є класифікатором подібності з одним стовпцем для деякого $j \in [1,m]$. Він складається з стовпця збігу C_j з мірою подібності ≈j і межі рішення $\rho_j \in [0{,}0, 1{,}0]$.

Для двох реляційних екземплярів r |= R та s |= S пара записів (r_k ,s_l) ∈ r × s відповідає ЛЧ X MD, якщо ці записи класифікуються як подібні для кожного збігу стовпців із присвоєнням λ. Більш формально це виражено формулою (2):

$$(r_k , s_l) | = X \equiv \bigwedge_{i=1}^{m} (r_k[A_i] \approx {}_i s_i[B_i]) \geq \lambda_i \qquad (2)$$

Аналогічно, пара записів (r_k ,s_l) ∈ r × s відповідає ПЧ Y MD, якщо ці записи класифікуються як подібні для кожного збігу стовпців із присвоєнням ρ, або формально що зображено формулою (3):

$$(r_k , s_l)| = Y \equiv (r_k [A_j] \approx {}_j s_l[B_j]) \geq \rho_j \qquad (3)$$

Таким чином на формулі(4), MD φ виконується для двох випадків r |= R і s |= S тоді і тільки

$$\forall(r_k , s_l) \in (r \times s) : (r_k , s_l) | = X \rightarrow (r_k , s_l) Y . \qquad (4)$$

Відповідно до цього визначення, кожен MD містить усі можливі збіги стовпців C на своїх ЛЧ, тобто ЛЧ завжди має розмір |C| = м. Однак для MD є релевантними лише ті C_i ∈ C з λ_i > 0,0, оскільки вони можуть класифікувати два записи як різнорідні і, отже, фактично звужують область застосування MD. Збіг стовпця C_i ∈ C з λi = 0,0 відповідає будь-яким двом значенням і, отже, не має значення для валідності MD. З цієї причини опускається кожен λ_i = 0,0 у специфікаціях MD і перерахуємо лише ті стовпці ЛЧ з λ_i > 0,0.

Оскільки MD φ повністю характеризується λ і ρ_j, якщо стовпець відповідає C, іноді MD позначають як φ(λ, ρj). Наприклад, записано $animal_{0,75}$ → $diet_{0,75}$, що означає, що якщо подібність двох записів у стовпці збігу, представленого твариною, більше або дорівнює 0,75, то подібність у стовпцях дієти буде щонайменше 0,75. Зауважте, що MD мають, як і FD, кілька збігів

стовпців ЛЧ, але збігається лише один стовпець ПЧ. Тим не менш, можливо згрупувати MD з ідентичними збігами стовпців ЛЧ, як φ(λ, ρ), оскільки вони відповідають однаковим парам записів.

Мета відкриття MD полягає в тому, щоб знайти всі φ(λ, ρ_j) або, іншими словами, знайти всі дійсні призначення для λ і пов'язаного ρ_j . Можливі межі рішення для збігів стовпців ЛЧ мають бути отримані з даних, що означає, що будь-яка існуюча подібність між двома значеннями атрибутів визначає один λ_i ∈ λ. Оскільки межі рішення випливають із даних, їх називають природними кордонами прийняття рішень. Для кожного набору присвоєнь λ відповідна максимальна межа рішення ПЧ ρj, з якою MD все ще виконується, відповідає мінімальній подібності записів збігу в їхніх атрибутах ПЧ. Іншими словами, усі записи, які збігаються на їхніх ЛЧ через λ, також повинні збігатися на їхніх ПЧ, тому нам потрібно встановити ρ_j як мінімум, як найменшу подібність відповідних записів на цій стороні, щоб сформувати правильний MD. Щоб проілюструвати це на прикладі, розглянемо MD $A_{0.7}B_{0.8} \rightarrow C_{0.5}$, що вірно для прикладу у таблиці 2.4. ЛЧ відповідає парам записів (1,3) і (2,3), мінімальна подібність яких в атрибуті C дорівнює 0,5. Отже, $\rho_j = 0{,}5$ є межею рішення для атрибуту ПЧ C. Більше значення ρj зробить MD недійсним. Крім того, менше значення λ для A або B змусить MD відповідати додатковим парам записів, які також роблять недійсним MD. З цієї причини $A_{0.7}B_{0.8} \rightarrow C_{0.5}$ є саме найбільш специфічним MD, який підтверджується даними і, отже, має бути виявлений.

Кількість кандидатів MD, які потрібно розглянути алгоритму виявлення, надзвичайно велика: нехай існує m збігів стовпців і t_i можливих меж рішення для кожного збігу стовпців C_i. Тоді існують можливі ЛЧ (=λ присвоєнь), які можуть бути згенеровані. Це число є експоненційним як за кількістю збігів стовпців, так і за межами можливих рішень. Крім того, ПЧ може мати t_j різні присвоєння для кожного збігу стовпця ρ_j ПЧ C_j . Оскільки кількість меж рішення t_i залежить від кількості рядків як у r, так і в s (природні межі рішення), існує до |r|·|s| можна перевірити. За допомогою правил мінімального скорочення можливо трохи контролювати складність пошуку, але воно

залишається NP-складним — значно важчим, зокрема, ніж відкриття FD, де $t_i \in \{0, 1\}$ і $|C| = |R|$. Усі існуючі підходи для виявлення відповідних залежностей, включаючи посилання [14] та [15], використовують попередньо визначені межі рішення, які фіксують число t як константу. Хоча це обмеження значно зменшує складність виявлення MD, воно також робить неможливим для алгоритмів виявлення всіх MD. З фіксованими межами рішення, скажімо, 0,0, 0,2, 0,4, 0,6, 0,8 і 1,0, ми не знайдемо цей MD. Цей MD менш точний, оскільки обидва пороги можна встановити на 15% вище без порушення залежності. Залежно від того, наскільки дрібно вибрані пороги, виявлені МД є більш-менш точними. Отримавши пороги з даних, відкриття завжди знаходить точні залежності, а отже, і всі залежності.

Тривіальні та мінімальні MD (відповідні залежності). За умови призначення λ для меж рішення ЛЧ, тоді всі пари записів, що відповідають цьому ЛЧ, мають подібність щонайменше λ_i у відповідному стовпці збігаються з C_i . Будь-які відповідні залежності з межею рішення ПЧ $\rho j \leq \lambda j$ мають бути істинним, незалежно від даних, тобто ми можемо просто генерувати всі такі тривіальні MD, навіть не перевіряючи дані, отже MD $\varphi(\lambda, \rho_j)$ є тривіальним якщо і тоді $\lambda_j \geq \rho_j$.

Це узагальнює визначення тривіальних FD, якщо ПЧ $\subseteq$ ЛЧ: Якщо атрибут з індексом j присутній по обидва боки FD, обидва ρ_j і λ_j дорівнюють 1,0, що також робить FD тривіальним. Якщо атрибут j є частиною лише ПЧ, то відповідне λ_j дорівнює 0,0 і FD є нетривіальним.

Усі MD з межею рішення ПЧ 0,0 за визначенням тривіальні. Однак MD з ненульовою межею рішення ЛЧ для збігу стовпця ПЧ не обов’язково тривіальні. Наприклад, MD $A_{0.5}B_{0.6} \rightarrow A_{0.6}$ є нетривіальним, оскільки його межа рішення ПЧ для атрибута A є суворішою, ніж межа рішення ЛЧ для A: A-значення можуть збігатися на ЛЧ, але не на ПЧ. Подальші приклади для тривіальних і нетривіальних МД наведені на рисунку 2.6. МД $\varphi(\lambda, \rho j)$ не перетинається, якщо $\lambda j = 0,0$. Отже, всі MD на рисунку 2.6 не перетинаються. Будь-який неперехідний MD з ρ_j 0,0 є нетривіальним.

Кількість можливих MD в реляційному наборі даних зростає експоненціально з кількістю рядків і стовпців. Однак MD не є незалежними один від одного, тобто ми можемо зробити висновок про дійсні MD з уже знайдених MD (мінімальне скорочення). Отже, тепер визначають частковий порядок для MD таким чином, що дійсні MD також визначають усі їхні наступні MD як дійсні.

Тривіальні	Нетривіальні
$A_{0.5} \rightarrow A_{0.5}$	$A_{0.5} \rightarrow A_{0.6}$
$A_{0.5} \rightarrow A_{0.4}$	
$A_{0.5}B_{0.6} \rightarrow A_{0.5}$	$A_{0.5}B_{0.6} \rightarrow A_{0.6}$
$A_{0.5}B_{0.6} \rightarrow A_{0.4}$	

Рисунок 2.6 – Приклади для тривіальних і нетривіальних MD

Ми починаємо із замовлення всіх можливих ЛЧ. Кожному набору меж рішення ЛЧ відповідає набір пар записів. Збільшення будь-якої з цих меж рішення також збільшує вибірковість відповідного класифікатора, тому він відповідає лише підмножині раніше збіганих пар записів. На формулі (5) стверджується , що λ включає λ' і тоді:

$$\lambda \leq \lambda' \equiv \ \forall i \ \in [1, m] : \lambda_i \leq \lambda'_i. \qquad (5)$$

Іншими словами, λ — це узагальнення λ', а λ' — спеціалізація λ. Враховуючи, що φ(λ, ρ_j) є дійсним MD, ми знаємо, що будь-який інший MD φ` (λ` , ρ_j`) з λ ≤ λ` відповідає підмножині пар записів, які відповідають φ. Якщо для φ(λ, ρ_j) і φ` (λ’ , $\rho_j$`) також істинно $p_j \leq \rho_j$, MD φ` також має бути дійсним, оскільки $\rho_j$` відповідає принаймні тим самим значенням, що й ρ_j, отже, може не вводити нові пари значень, що порушують значення на ПЧ. На формулі (6) стверджується, що MD φ включає іншу MD φ` і тоді:

$$\varphi(\lambda, \rho_j) \leq \ \varphi'[\ (\lambda', \rho'_j)] \equiv \ \lambda \leq \ \lambda' \wedge \ \rho_j \ \geq \ p_j'. \qquad (6)$$

Знову ж таки, φ є узагальненням φ` , а φ` є спеціалізацією φ. Якщо відповідність залежностей справедливо, всі його спеціалізації також мають місце, і ми можемо зробити висновок про них. Відношення підрахунку визначає частковий порядок на множині всіх MD Ф.

Нижче наведено три приклади дійсних відношень підрахунку. Інтуїтивно, підвищення межі рішення ЛЧ λ_i (1) і (2) зображені на рисунку 2.7 або зменшення межі рішення ПЧ ρ_j (3) генерує дійсні MD з існуючих MD. Будь-яке інше маніпулювання межами рішення породжує незрівнянних кандидатів MD.

$$(1) \quad A_{0.5} \rightarrow C_{0.6} \leq A_{0.6} \rightarrow C_{0.6}$$
$$(2) \quad A_{0.5} \rightarrow C_{0.6} \leq A_{0.6}B_{0.1} \rightarrow C_{0.6}$$
$$(3) \quad A_{0.5} \rightarrow C_{0.6} \leq A_{0.5} \rightarrow C_{0.5}$$

Рисунок 2.7 – Дійсні відношення

Оскільки багато MD можна вивести з інших, ми можемо обмежити відкриття лише тими MD, які неможливо вивести. Такі відповідні відношення називають мінімальними. Тут відповідна залежність $\varphi \in \Phi$ мінімальна тоді й тоді, коли $\varphi` \in \Phi : \varphi` \neq \varphi \wedge \varphi` \leq \varphi$.

Це визначення є узагальненням визначення мінімальних FD. Як і у FD discovery, набір мінімальних залежностей набагато менший за набір усіх залежностей і все ще визначає повний набір залежностей. Таким чином цей підхід до відкриття виявляє лише мінімальні MD.

Як і у функціональних залежностях, набір мінімальних залежностей набагато менший, ніж набір усіх залежностей, і все ще визначає повний набір залежностей. Хоча концепція відповідних залежностей наведена в [13], автори не обговорювали, як знайти корисні залежності у даних(відповідні

залежності). Таким чином, цей підхід до відкриття виявляє лише мінімальні відповідні залежності.

2.3 Алгоритми виявлення відповідних залежностей у даних

Для виявлення залежностей даних пропонується дві стратегії пошуку: обхід решітки та висновок із пар записів. Перший моделює простір пошуку як решітку набору потужностей комбінацій атрибутів, щоб класифікувати кандидатів у решітці як правдиві або хибні залежності, різні стратегії обходу та правила обрізання мають на меті зменшити кількість перевірок кандидатів. Остання стратегія порівнює всі (необхідні) пари записів у пошуках незалежностей, з яких вона отримує всі істинні залежності. Утримуючись від окремих перевірок кандидатів, ця стратегія легше контролює розмір простору пошуку, але витрати на порівняння записів квадратичні щодо кількості записів. Для того щоб вдосконалити пошук відповідних залежностей необхідним є розглянути запропоновані стратегії пошуку.

2.3.1 Обхід решітки

Алгоритм обходу решітки є підходом знизу вгору, в ширину, який базується на алгоритмі виявлення Tane [16]. Починаючи з найзагальніших MD в корені решітки зображних на рисунку 2.8, алгоритм крокує вгору, рівень за рівнем, перевіряючи всіх кандидатів, які він передає. Під час обходу алгоритм обрізає всі немінімальні кандидати MD.

Частковий порядок MD, який розглянутий вище, описує решітку набору потужностей класифікаторів стовпців, тобто збіги стовпців та межі їх вирішення. Точніше, він описує одну решітку набору потужності для кожного можливого збігу стовпців ПЧ, оскільки MD з різними збігами стовпців ПЧ є незалежними. Подібно до виявлення інших типів залежностей, решітка слугує

структурою для систематичного проходження простору пошуку всіх можливих кандидатів у MD.

Решітка для будь-якого співпадання стовпця ПЧ C_j починається з кореня MD $\emptyset \rightarrow C_{j,1.0}$. Тут $\emptyset$ являє собою призначення межі рішення λ з $\lambda_i = 0{,}0$ для всіх i, тобто всі межі рішення встановлені на 0,0, а $C_{j,1,0}$ — відповідність стовпця ПЧ для решітки C_j з максимальною межею рішення $\rho j = 1{,}0$. Щоб створити прямий наступник цього (або будь-якого іншого) вузла решітки, ми або збільшуємо один λ_i, або зменшуємо ρ_j. Щоб збільшити або зменшити межу рішення, використовуються природні межі рішення відповідного збігу стовпців, тобто відсортований список подібних значень, які фактично присутні у відповідних стовпцях. Це гарантує, що будуть створені лише потенційно мінімальні кандидати MD. Таблиця подібності, яка представлена в таблиці 2.5 перелічує ці природні межі рішення для кожного атрибута прикладу відношення. Для прикладу на рисунку 2.8 зображено повну решітку набору потужності, кандидатів MD для відповідності стовпця ПЧ.

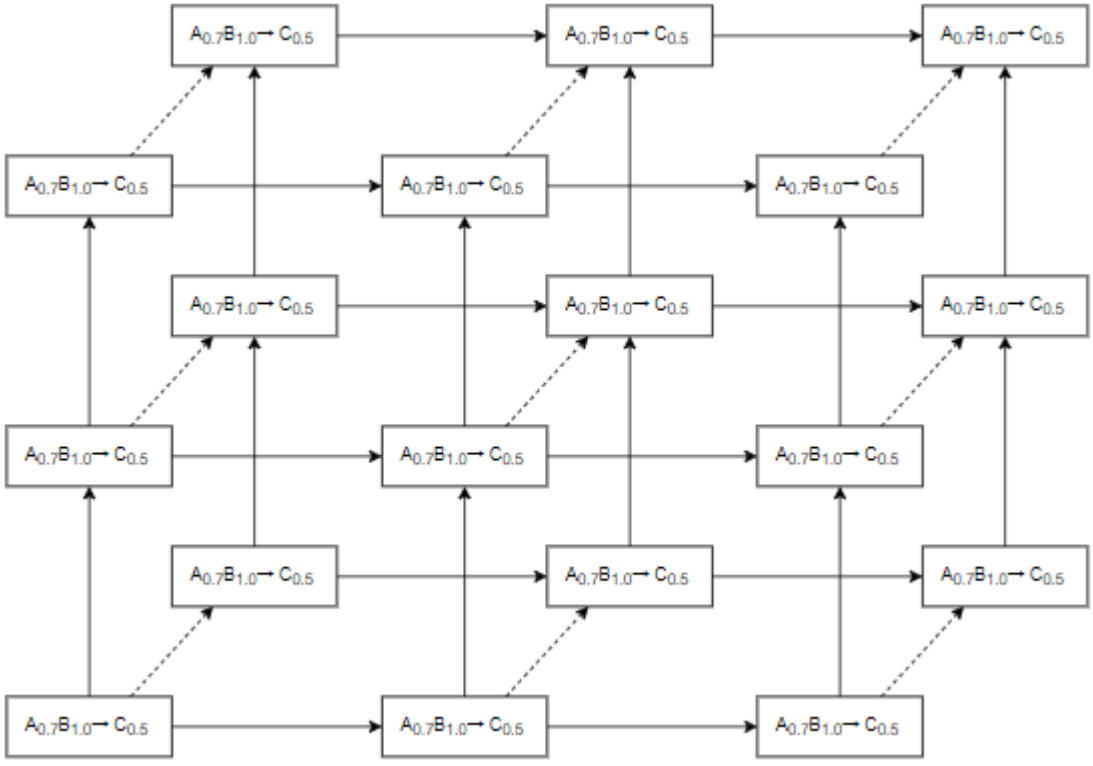


Рисунок 2.8 – Решітка простору пошуку

Ще існує одна така решітка для A і B. Кожна відповідність стовпця створює один вимір, а довжина кожного виміру визначається кількістю можливих меж прийняття рішень. Стрілки позначають прямі відношення спеціалізації, тобто вони змінюють лише одну межу рішення на один природний крок. Ми опустили всі тривіальні та нерозбірні MD для простоти, тим не менш, ці МД зазвичай є частиною решітки. На рисунку 2.8 також представлена послідовна класифікація всіх кандидатів MD. Дійсні MD знаходяться вгорі, тоді як недійсні MD – внизу. Єдині два мінімальні MD для стовпця ПЧ, які відповідають C, $B_{1.0} \rightarrow C_{1.0}$ і $A_{0.7}B_{0.8} \rightarrow C_{0.5}$, виділені посередині. Оскільки мінімальним MD ніколи не передує дійсний MD, ми можемо пройти решітку від кореня вгору, обрізати всі спеціалізації дійсних MD і таким чином виявити всі мінімальні MD. Хоча два MD з різних решіток, тобто з різним ПЧ, не пов'язані з порядком, вони можуть мати ідентичний ЛЧ і відповідати однаковим парам записів.

За Tane[16] глибина MD-кандидата визначається, як довжина найкоротшого шляху від кореня до MD. Кроки у вимірі ПЧ не додають довжини шляху, тому всі MD з ідентичними ЛЧ мають однакову глибину. Наприклад, MD $A_{0.7}B_{0.8} \rightarrow C_{0.5}$ має глибину два. Усі вузли з різними ЛЧ, але однаковою глибиною незалежні один від одного, тобто не узагальнюють і не спеціалізують один одного. Пізніше ми використовуємо це спостереження, щоб розпочати відкриття.

Зауважимо, що концептуально існує одна решітка для кожного збігу стовпців ПЧ, але алгоритм обходить їх одночасно, обробляючи всі MD з однаковими ЛЧ одночасно з міркувань ефективності. Рисунок 2.9 з алгоритмом пояснює обхід більш детально.

Алгоритм починається з ініціалізації решітки з найбільш загальними MD рядок 2. Наразі глибина MD вважається його рівнем, пізніше ми повторно визначаємо рівень MD з іншою функцією, щоб покращити розпаралелювання. Після отримання поточного рівня з решітки алгоритм обробляє кожного кандидата MD на цьому рівні, рядок 5. Мета полягає в тому, щоб визначити

всі максимальні межі рішення ПЧ, для яких поточний ЛЧ формує дійсні MD. Спочатку знімаємо кандидата з решітки рядок 6. Перед перевіркою MD ми отримуємо максимальну межу рішення ПЧ, яка існує в решітці для MD з більш загальним ЛЧ, ніж поточний рядок 7. Алгоритм зображений на рисунку 2.9.

```
Дані: Реляційні екземпляри r, s. Стовпець відповідає C.
Мінімальні межі рішення Rhs ρmin. Мінімальна
підтримка minSup.
Результат: Набір усіх мінімальних МД Ф.
1   m ← |C|;
2   Φ ← {ϕ(∅, (1.0, j))|j ∈ [1,m]};
3   l ← 0;
4   while l ≤ getMaxLevel(Φ) do
5       foreach ϕ(λ, ρ) ∈ getLevel(l, Φ) do
6           Φ ← Φ \ ϕ;
7           λ_lower ← getLowerBoundaries(λ, Φ);
8           ρ', σ ← validate(λ, ρ, r, s, C, λ_lower, ρ_min);
9           if σ < minSup then
10              markUnsupported(λ);
11          else
12              foreach ρ'_j ∈ ρ' do
13                  if ρ'_j > λ_j∧
14                      ρ'_j ≥ ρ_min[j]∧
15                      ρ'_j > λ_lower[j] then
16                          add(ϕ(λ, ρ'_j) , Φ)
17              foreach i ∈ [1,m] do
18                  if canSpecializeLhs(λ, i) then
19                      λ' ← specializeLhs(λ, i);
20                      if isSupported(λ' ) then
21                          foreach ρ_j ∈ ρ do
22                              if ρ_j > λ'_j then
23                              addIfMin(ϕ(λ', ρ_j), Φ);
24      l ← l + 1;
25  return Φ;
```

Рисунок 2.9 – Псевдокод алгоритму обхід решітки

Щоб знайти новий мінімальний MD з поточним ЛЧ, його межа рішення ПЧ має бути вищою за відповідну нижню межу. Щоб отримати ці нижні межі, алгоритм повинен перевірити всі MD в решітці за допомогою більш загального ЛЧ. Враховуючи нижні межі, алгоритм тепер визначає максимальні межі рішення ПЧ для поточного кандидата, тобто перевіряємо його рядок 8: ідентифікуємо усі збігаючі пари записів, визначаємо їх мінімальну

подібність у збігах стовпців ПЧ і обчислюємо їх підтримку . Визначивши межі підтримки та максимального ПЧ, алгоритм виділяє два випадки, використовуючи мінімальний поріг підтримки: (a) ЛЧ підтримується або (b) не підтримується. Якщо він не підтримується, ми позначаємо ЛЧ як непідтримуваний і зупиняємо обхід для цього кандидата рядок 10. Якщо це підтримано, ми продовжуємо висновок щодо нових кандидатів у MD.

Спочатку ми перевіряємо для кожної межі рішення ПЧ, чи вона призводить до цікавого, нетривіального та мінімального MD. Якщо це так, ми додаємо MD до решітки рядок 16. Алгоритм також може негайно випромінювати цей MD в результаті. Далі ми робимо висновок про спеціалізації, які ще не охоплені жодним мінімальним MD. Він створює ці спеціалізації, збільшуючи кожну межу рішення ЛЧ λi до наступного можливого значення рядок 19. Стара межа рішення ПЧ зберігається, а спеціалізація додається до решітки, якщо вона мінімальна рядок 23. У процесі виявлення алгоритм позначив деякі кандидати MD як непідтримувані рядок 10 і перевірив, чи підтримуються певні нові кандидати рядок 20. Щоб зробити це ефективно, алгоритм зберігає непідтримуваний ЛЧ λ в додатковій решітці та перевіряє підтримку нового ЛЧ λ за допомогою пошуку узагальнення в цій решітці. Як і основна решітка-кандидат, ця додаткова решітка реалізована у вигляді дерева префіксів, але замість того, щоб анотувати межі рішення ПЧ, коментується, чи підтримується ЛЧ чи ні.

Після обробки кожного MD поточного рівня, алгоритм переходить до наступного рівня. Коли всі рівні оброблені, тобто коли решітка не містить більше кандидатів у вищих рівнях, усі мінімальні MD були знайдені.

2.3.2 Висновок із пар записів

Другим підходом відкриття є висновок з пар записів: систематично виводяться незалежності з порівнянь пар записів і використовують їх для

отримання всіх дійсних залежностей. Ця методика була вперше запропонована алгоритмом Fdep [17]. Висновок дуже ефективний щодо розміру простору-кандидата, оскільки для того, щоб позначити залежність як недійсну, достатньо знайти одну недійсну пару записів. Алгоритм пошуку зображений на рисунку 2.10.

Дані: Реляційні екземпляри r, s. Стовпець відповідає C.
Мінімальні межі рішення Rhs ρ_min .
Результат: Набір усіх мінімальних МД Ф.

```
1   m ← |C|;
2   Φ ← {ϕ(∅, (1.0, j))|j ∈ [1,m]};
3   foreach (r_k ,s_l ) ∈ r × s do
4       sim ← calculateSimilarity(r_k, s_l, C);
5       violated ← findViolated(sim, Φ);
6       foreach ϕ(λ, ρj) ∈ violated do
7           Φ ← Φ \ ϕ;
8           if sim[j] ≥ ρ_min[j] ∧ sim[j] > λ_j then
9               ρ'_j ← sim[j];
10              addIfMin(ϕ(λ, ρ'_j), Φ);
11          foreach i ∈ [1,m] do
12              if canSpecializeLhs(λ, i , sim[i]) then
13                  λ' ← specializeLhs(λ, i , sim[i]);
14                  if ρ_j > λ'_j then
15                      addIfMin(ϕ(λ', ρ_j), Φ);
16  return Φ;
```

Рисунок 2.10 – Псевдокод алгоритму висновку із пар записів

Щоб зрозуміти, яких кандидатів MD конкретна пара записів робить недійсними, ми розглянемо два записи r і s та їх конкретне значення подібності $sim_i \in sim$ у кожному стовпці збігаються з $C_i \in C$. Кандидат MD φ з межами рішення ЛЧ λ відповідає парі записів, якщо для всіх $\lambda_i \in \lambda$ справедливо, що $\lambda_i \leq sim_i$, тобто, якщо всі межі рішення ЛЧ менші або дорівнюють спостережуваній подібності в парі записів. Тепер ми можемо зробити

висновок про недійсність φ наступним чином, якщо кандидат MD φ збігається з деякою парою записів (r,s) і його межа рішення ПЧ ρ_j більша за спостережувану подібність, тобто $\rho_j > sim_j$, тоді цей запис порушує φ.

Припустимо, Ф — це множина всіх кандидатів у MD, які ми вважаємо істинними і що нам дана пара записів (r,s), яка ще не розглядалася. Ми можемо зменшити Ф, виключивши всі $\varphi \in Ф$, які мають підмножину кордону рішення ЛЧ ($\forall\lambda_i \in \lambda : \lambda_i \leq sim_i$), але більшу межу рішення ПЧ ($\rho_j > sim_j$). Наприклад, пара записів (1, 3) у таблиці 2.5 збігається та робить недійсними шість кандидатів MD $\emptyset \rightarrow C_{1.0}$, $A_{0.7} \rightarrow C_{1.0}$, $A_{1.0} \rightarrow C_{1.0}$, $B_{0.8} \rightarrow C_{1.0}$, $A_{0.7}B_{0.8} \rightarrow C_{1.0}$ і $A_{1.0}B_{0.8} \rightarrow C_{1.0}$, які показані синім на малюнку 1. Після того, як кандидат MD був визнаний недійсним якоюсь парою записів, ми можемо зробити висновок, що цей MD і всі узагальнення є недійсними. Проте спеціалізації MD все ще можуть бути дійсними і потенційно також мінімальними. Це ті MD, які найближчі до попереднього MD, але ще не порушені. Щоб зменшити Ф, ми видалимо кожне недійсне φ і додамо його найближчу, все ще діючу та мінімальну спеціалізацію. решітка простору пошуку кандидатів MD. Алгоритм описує весь процес висновку MD: Аналогічно підходу обходу решітки, ми спочатку ініціалізуємо решітку найзагальнішими MD рядок 2. Потім алгоритм повторює всі пари записів у вхідному відношенні рядок 3. Пізніше ми вводимо стратегію вибірки, яка уникає порівняння всіх пар записів. Пізніше ми вводимо стратегію вибірки, яка уникає порівняння всіх пар записів. Для кожної вибраної пари записів (r_k ,s_l) алгоритм обчислює подібність «sim» для всіх стовпців збігів C (рядок 4). Завдяки цій схожості алгоритм шукає в решітці всі порушені MD, як пояснювалося вище рядок 5. Будь-який порушений MD спочатку видаляється з решітки рядок 7, а потім використовується для визначення нових кандидатів MD, які є спеціалізаціями поточного MD, або знижуючи межу рішення ПЧ рядки 8–10, або збільшуючи деяку межу рішення ЛЧ рядки 11–15.

Оскільки MD з тим же ЛЧ, що і струм, може триматися для більш слабкого ПЧ, ми зменшуємо межу рішення ПЧ до подібності записів у

відповідних стовпцях. Якщо ця спеціалізація мінімальна, вона додається в решітку (рядок 10). Для створення інших спеціалізацій ми спеціалізуємо ЛЧ, тому поточна пара записів, що порушує, більше не відповідає їй. Для цього ми збільшуємо межу рішення для кожного збігу стовпців ЛЧ до наступного можливого значення вище подібності записів у відповідних стовпцях рядок 13. Якщо новий кандидат MD, що складається зі спеціалізованих ЛЧ і старих ПЧ, мінімальний і нетривіальний, алгоритм додає його до решітки рядок 15.

Зберігається межа рішення ПЧ, оскільки з іншої пари записів було зроблено висновок, що MD на цьому шляху повинен використовуватися, щоб бути мінімальним. Якщо вища межа рішення ПЧ фактично можлива, це визначається іншим MD на іншому шляху через решітку. Алгоритм завершується після обробки всіх пар записів, Ф тоді містить усі допустимі мінімальні MD. Для ілюстрації цього підходу ми використовуємо дані, наведені в таблицях 2.4 і 2.5, і робимо висновок про мінімальні MD згідно з рисунком 2.8. Початковий кандидат MD – $\emptyset \rightarrow C_{1.0}$. Пара (1, 2) відповідає цій MD, і ми, таким чином, робимо висновок про її спеціалізації: оскільки (1, 2) має подібність 0,0 у стовпцях C, не випливає нетривіальний MD з ∅ як ЛЧ. Однак ми робимо висновок $A_{0.7} \rightarrow C_{1.0}$ і $B_{1.0} \rightarrow C_{1.0}$. (1, 3) порушує перше, і ми робимо висновок $A_{0.7} \rightarrow C_{0.5}$. $A_{0.7}B_{1.0} \rightarrow C_{1.0}$ також є висновком, але не мінімальним, (1, 4) порушує $A_{0.7} \rightarrow C_{0.5}$, і, отже, $A_{0.7}B_{0.8} \rightarrow C_{0.5}$ є висновком. З цього моменту жодна пара записів більше не визнає кандидатів недійсними, і виявляються мінімальні MD $B1.0 \rightarrow C1.0$ і $A0.7B0.8 \rightarrow C0.5$.

2.4 Гібридний підхід до виявлення відповідних залежностей

Розглянувши дві стратегії пошуку, їх переваги та недоліки, а також їх структури даних, алгоритм пошуку можна вдосконалити поєднавши їх у гібридний алгоритм. Алгоритм замість того, щоб відразу вибирати всі записи, компонент вибірки вибирає пари записів послідовно. З кожною обраною парою записів компонент висновку виводить не відповідні залежності, які

використовуються для спеціалізації кандидатів MD в решітці. Коли вибірка стає неефективною, вона перемикається на компонент обходу, який вибирає кандидатів з решітки (по рівнях, знизу вгору). Ці кандидати перевіряються компонентом перевірки та відповідно оновлюються в решітці. Після кожного раунду перевірки вдосконалений алгоритм продовжує використовувати стратегію висновку. Щоб підвищити ефективність цієї фази, стратегія обходу збирає пари записів як рекомендації щодо висновку для стратегії висновку. За допомогою обміну кандидатами MD та рекомендаціями для висновків обидві фази підтримують один одного.

Висновок із пар записів. У порівнянні з його основою у підрозділі 2.3.1, алгоритм висновку з пар записів змінюється лише незначно при використанні в об'єднаному алгоритмі. Перші дві зміни прості: по-перше, не ініціалізується набір мінімальних MD Ф з кожним викликом методу, а повторно використовується та ділиться одним і тим же набором MD з фазою обходу решітки. По-друге, додається ще один параметр, який переносить рекомендації висновку з фази обходу до цієї фази висновку. Ці рекомендації перевіряються, перш ніж ми входимо в основний цикл у рядку 3 алгоритму, який зображений на рисунку 2.7. Третя зміна підраховує кількість порівнянь пар записів (r_k ,s_l) рядок 3 і кількість уточнених MD φ, які насправді можуть взяти з Ф рядок 7. Частка цих двох чисел, тобто уточнення/порівняння, з часом зменшується, оскільки необхідно виявити все менше уточнень. Тому він відображає ефективність пошуку на основі висновків і запускає перемикання фази після обробки деякої пари записів, якщо вона падає нижче певного порогу. Автори схожого алгоритму пошуку функціональних залежностей[18] показали, що для виявлення гібридних функціональних залежностей будь-який поріг ефективності від 0,1% до 10% добре працює, оскільки ефективність швидко падає [19]. Це також відбувається при виявленні відповідних залежностей, отже, починаємо з порогового значення 1%. З кожним переходом у фазу висновку об'єднаний алгоритм вдвічі зменшує цей поріг, щоб знову стати ефективним. Четверта і остання зміна полягає в тому, що алгоритм

уникає обробки зайвих пар записів, які є парами записів з тим же набором подібності, що й раніше оброблені пари записів. Оскільки надлишкові пари створюють однакові недійсні, алгоритм запам’ятовує унікальні набори подібностей, які були оброблені, щоб не обробляти їх знову.

Обхід решітки. Стратегія обходу решітки, що використовується, є розширенням порівневого алгоритму обходу знизу вгору у підрозділі 2.3.2. Розширення в першу чергу спрямовані на те, щоб дозволити цьому алгоритму перемикатися на стратегію висновку та від неї в гібридних налаштуваннях, але також внесені деякі оптимізації, які не залежать від гібридної стратегії виконання.

Фаза висновку відповідає за ініціалізацію набору мінімальних MD Ф, який є решіткою простору пошуку. Оскільки Ф вже обчислюється фазою висновку на основі пар записів, яка завжди виконується першою, ми видаляємо рядок 2 в алгоритмі зображеному на рисунку 2.9 і робимо Ф параметром. Як згадувалося раніше, об’єднаний алгоритм перемикається від фази обходу до фази висновку щоразу, коли підтверджено один рівень кандидатів.

На відміну від [18], ми не вимірюємо ефективність фази обходу і перемикаємо більш активно, тому що якщо перемикач не потрібен, фаза висновку все одно швидко повертається. Щоб реалізувати перемикач, ми замінюємо цикл while у рядку 4 оператором if і перемикаємося на фазу висновку в кінці алгоритму. Щоб підтримати фазу висновку після перемикання, фаза обходу збирає, можливо, багато рекомендацій висновку, які є парами записів (r_k ,s_l), які визнали недійсними певного кандидата MD. Ці пари записів раніше не порівнювалися, бо інакше кандидата б не перевіряли. Цілком імовірно, що вони так само впливають на інших кандидатів, що робить їх перевірку перспективною. Оскільки функція перевірки в рядку 8 знаходить ці пари записів природним чином, алгоритм може просто зібрати їх. Зібрані пари записів потім передаються на фазу висновку як пропозиції порівняння з кожним перемиканням фаз.

Функція getLevel() у рядку 5 алгоритму на основі обходу решітки витягує всі кандидати MD певного рівня з решітки простору пошуку. Оскільки

всі отримані кандидати незалежні один від одного, ми можемо перевірити їх паралельно, тобто розпаралелювати цикл for у рядку 5 до кількох потоків. Решітка на рисунку 2.8, наприклад, має 2 кандидати MD відстані один (див. ЛЧ $A_{0.7}$ і $B_{0.8}$) і 4 кандидати MD потужності один (див. ЛЧ $A_{0.7}$, $A_{1.0}$, $B_{0.8}$ і $B_{1.0}$) – зверніть увагу, що враховуються лише ЛЧ. Всі ПЧ обчислюються динамічно. З потужністю як функцією рівня один рівень може містити кандидатів, які спеціалізуються на інших, наприклад $A_{0.7} \rightarrow C_{1.0}$ і $A_{1.0} \rightarrow C_{1.0}$. Щоб виявити лише мінімальні MD, нам потрібно перевірити всі узагальнення перед їх спеціалізацією.

Тому алгоритм спочатку аналізує кожен рівень на предмет спеціалізацій, перевіряючи кандидатів по парах, щоб потім відкласти перевірку спеціалізацій. Отже, у нашому прикладі ЛЧ $A_{0.7}$ закінчується до $A_{1.0}$, а $B_{0.8}$ закінчується перед $B_{1.0}$. Тим не менш, будь-яку іншу комбінацію ЛЧ, наприклад $A_{1.0}$ і $B_{0.8}$, можна оцінювати паралельно. Незважаючи на накладні витрати на аналіз спеціалізації на кожному рівні, виграш від розпаралелювання переважає ці витрати. Багато кандидатів, наприклад $A_{1.0} \rightarrow C_{1.0}$ і $B_{0.8} \rightarrow C_{1.0}$ на рисунку 2.8, для прикладу, незалежні один від одного, але мають різну глибину.

Перш ніж перевіряти паралельно набір усіх (без спеціалізації) кандидатів MD, їх потрібно групувати за їхніми ЛЧ λ. Це дає нам набір ПЧ ρ, який об’єднаний гібридний алгоритм може перевірити одночасно: Алгоритм спочатку отримує всі пари записів, що відповідають λ, а потім перевіряє всі ρ щодо цих записів. Іншими словами, він обчислює максимальні межі рішення ПЧ ρ для кордонів рішення ЛЧ λ, щоб алгоритм обходу міг перевірити ці ρ. Алгоритм зображений на рисунку 2.11 детально показує цей алгоритм перевірки. Він надає три різні підходи до перевірки залежно від потужності MD, яка становить 0, 1 або більше.

MD з потужністю 0 мають усі межі рішення ЛЧ встановлені на 0,0 рядок 2. Вони відповідають усім парам записів і, отже, мають опору $\sigma |r| \cdot |s|$. Межі рішення ρ цих MD є мінімальною схожістю у відповідних збігах

стовпців ПЧ і можуть бути отримані за допомогою одного лінійного сканування індексу подібності рядок 3.

Для MD з потужністю 1, рядок 4 необхідно враховувати лише один стовпець ЛЧ, що відповідає C_i. Отже, запропонований алгоритм починає з ітерації всіх значень та їх позицій r` у r, використовуючи індекси списку позицій π[r] цього стовпця, тобто з індексом i, рядок 5. Для кожного r-значення алгоритм збирає всі позиції s' значень s, які подібні до значення r щодо межі рішення ЛЧ λ_i, використовуючи індекс подібності, тобто він вибирає записи, які мають подібність більше або дорівнює єдина межа рішення ЛЧ рядок 6. Оскільки всі записи в r` і s` збігаються попарно, |r` |·|s` | додається до опори σ (рядок 7). Потім гібридний алгоритм всі записи r_k і s_l у r` × s` з їхньою ПЧ подібністю та зберігає мінімальну подібність у ρ`. Ці мінімальні подібності становлять максимальні межі рішення ПЧ для підтверджених кандидатів MD рядок 9.

Якщо вхідні MD мають потужність > 1, рядок 10, алгоритм не може безпосередньо прочитати значення r зі свого індексу списку позицій. Тому він групує записи r за їх комбінаціями значень ЛЧ на льоту, перетинаючи індекси списку позицій усіх збігів стовпців ЛЧ рядок 11. Кожний кластер в результуючому індексу списку позицій визначає репрезентативна комбінація значень rep, яку вимагає алгоритм, щоб знайти подібні комбінації значень у s. Щоб знайти всі позиції s` записів s, які подібні в усіх збігах стовпців ЛЧ, алгоритм збирає для кожного окремого стовпця ЛЧ відповідність позицій подібних записів. Цей стовпець відповідає за індексом подібності та перетинає всі ці набори, рядки 12–14. Набори r` і s` відповідних позицій r- і s-записів потім використовуються для оновлення ρ` і σ так само, як у випадку потужності 1 рядки 15–17.

Навіть із змінами, поясненими вище, усі відповідні залежності, виявлені вдосконаленим алгоритмом який є об’єднанням двох запропонованих, є дійсними та мінімальними. З цієї причини пошук залежностей завершується на фазі обходу решітки, коли всі кандидати

відповідних залежностей були оброблені, що дає Ф як кінцевий результат. Запропонований гібридний алгоритм пошуку зображений на рисунку 2.11.

Дані: межі рішення λ, ρ . Plis r π [r]. Реляційні екземпляри r, s. Стовпець відповідає C. Максимальна Lhs і мінімальна Rhs межі λ_нижня, ρ_min. Індекси подібності simIndex.
Результат: максимальні межі рішення ρ'. Підтримка σ .

```
1   ρ', σ ← ρ, 0;
2   if |λ| = 0 then
3       return getMinSims(ρ , simIndex), |r|·|s|;
4   if |λ| = 1 then
5       foreach value,r' ∈ π[r][i] do
6           s' ← getSimRecs(value, λi , simIndex[i]);
7           σ ← σ + |r'|·|s'|;
8           foreach rk ,sl ∈ r' × s' do
9               ρ' ← computeMaxRhs(rk,sl, ρ', λ, λlower , ρmin);
10  else
11      foreach rep,r' ∈ groupByValue(π[r], λ, C) do
12          s' ← getSimRecs(rep[j], λj , simIndex[j]);
13          foreach λi ∈ λ \ λj do
14              s' ← s' ∩ getSimRecs(rep[i], λi , simIndex[i]);
15          σ ← σ + |r'|·|s'|;
16          foreach rk ,sl ∈ r' × s' do
17              ρ' ← computeMaxRhs(rk,sl, ρ', λ, λlower, ρmin);
18  return ρ', σ;
```

Рисунок 2.11 – Псевдокод об’єднаного алгоритму

2.5 Засіб перевірки орфографії автентифікованих даних за допомогою машинного навчання

Щоб вдосконалити виправлення даних у техніці Matching Dependency, пропонується попередню перевірку орфографії у вхідних даних, яка допоможе використовувати «брудні дані» для більш ефективного виведення рекомендацій. З цієї причини було запропоновано використання машинного навчання, яке допоможе усувати помилки у вхідних даних.

Метою цього впровадження є побудова моделі, яка може приймати кортежі з орфографічними помилками, як вхідні дані та виводити той самий набір, але з виправленими помилками в значеннях.

Наприклад, якщо назва вулиці — «вул. Хрещатик», а користувач ввів назву вулиці як «вулеця Крищатик», тоді техніка повинна знайти її як хороші вхідні дані та використовувати, як дійсні дані для перевірки інших записів для тієї самої адреси.

Одним з найважливіших аспектів машинного навчання є робота з хорошими, чистими даними. Модель на базі якої пропонується використовувати перевірку для цієї задачі є модель «seq2seq», яка представлена у роботі[20]. Sequence-to-sequence моделі (seq2seq) – це моделі глибокого навчання, які досягли великих успіхів у таких завданнях, як машинний переклад, сумаризація тексту, анотація зображень та інші. Вона була налаштована для роботи з вхідними даними, які використовуються у дипломній роботі, тому було змінено архітектуру та додані параметри для аналізу результатів.

Основна увага у цьому засобі буде присвячена тому, як підготувати дані для моделі, а також декілька інших особливостей моделі буде описано у програмній реалізації цього засобу. У цьому втіленні буде використовуватися Python 3 і TensorFlow 1.1. Дані які будуть використовуватися описані у підпункті 3.2 .

Щоб оцінити модель, на що вона здатна, наведено приклади які вона має виявляти та використовувати для рекомендацій:

- у вхідній адресі:
  - а) «вулеця Крищатик», як «вул. Хрищатик»;
  - б) «провулок Ніжний», як «пров. Ніжний» та навпаки.
- у місті:
  - а) «Киев», як «Київ».
- у області:
  - а) «обл. Київська», як «Київська область»;

б) «Хмельнецька область», як «Хмельницька область».

Таким чином, ми отримуємо вдосконалений метод та засіб очищення даних на основі відповідності залежностей, який буде перевіряти не тільки вхідні дані користувача, але і автентифіковані дані. Такий підхід дозволить більш точно давати рекомендації та пришвидшить очищення даних.

2.6 Алгоритм очищення даних на основі відповідності залежностей

Початок побудови алгоритму базуємо на зразках наборів даних із зовнішнього джерела даних та даних введення користувача. Де дані зовнішніх джерел - це авторизовані та надійні дані, які будуть використовуватися для перевірки вхідних даних користувача. Відповідно до назви, дані введені користувачем - це дані, введені будь-яким користувачем. У цієї реалізації методу береться до уваги назва вулиці, місто, область, країну та поштовий індекс для процесу очищення даних. Наприклад, якщо користувач при введенні адреси помилився у поштовому індексі, то цей алгоритм, швидше за все, виявить його, перевіривши його за допомогою зовнішнього джерела даних, і порекомендує всі можливі поштові індекси, які відповідають іншим значенням адреси, тобто , назва вулиці, місто, область та країна.

Реалізований алгоритм виконує такі кроки:

− алгоритм має приймати введені користувачем дані та автентифіковані дані у форматі двовимірного списку, де кожен список (всередині списку) має адресу;

− інший двовимірний список створюється для окремого елемента адреси (для назви вулиці/міста/області/поштового індексу) окремо від автентифікованих даних, який містить відсортований елемент адреси разом із вихідним індексом;

− введені користувачем дані виконуються в циклі для отримання однієї адреси за раз, щоб перевірити їх справжність. У середині циклу створюється новий список вихідних індексів із зовнішніх даних для окремого елемента

адреси введення. Цей список отримується з функції пошуку, яка приймає вхідний елемент адреси та двовимірний список елемента зовнішньої адреси разом з їх індексами;

− функція гібридного запропонованого пошуку продовжує пошук надісланого елемента вхідної адреси у списку елементів зовнішньої адреси та записує всі індекси елемента зовнішньої адреси у список, коли знайдено відповідність залежностей;

− після отримання всіх окремих списків вихідних індексів для вхідних адресних елементів проводиться перевірка загальних індексів для всіх списків. Якщо серед списків знайдено загальний індекс, він перевіряє правильність введення адреси, саме за допомогою втіленої техніки машинного навчання. В іншому випадку адреса введення містить помилки;

− щоб рекомендувати правильну адресу для негативного випадку, усі окремі списки вихідних індексів додаються як елемент у новий список. Тепер створено новий словник, щоб усі індекси були ключами, а їх кількість-значеннями з цього двовимірного списку;

− використовуючи функцію найбільшого значення, найвище значення зі словника отримується та зберігається у новій змінній. Потім цикл на ключах словника виконується за умовою, де значення витягується зі словника за допомогою його ключа. Якщо отримане значення дорівнює максимальному значенню функції, ключ зберігається у списку для рекомендацій;

− нарешті, у списку рекомендацій запускається цикл для вилучення вихідних індексів автентифікованих даних для надання рекомендацій щодо конкретних вхідних даних. Ці результати друкуються як рекомендації.

Псевдокод для реалізованого алгоритму вдосконаленого методу наведено на рисунку 2.12 нижче:

```
Set Data = list()
Set avtentData = list()
Set indexValue = list()
StreetData = sorted list of street names with their index from avtentData
CityData = sorted list of city names with their index from avtentData
RegionData = sorted list of region names with their index from avtentData
ZipData = sorted list of zip codes with their index from avtentData
for each data from the Data do
    resultStreet = hybridSearch(StreetData, first element of data)
    Set indexValue = list()
    resultCity = hybridSearch(CityData, first element of data)
    Set indexValue = list()
    resultRegion = hybridSearch(RegionData, first element of data)
    Set indexValue = list()
    resultZip = hybridSearch(ZipData, first element of data)
    Set indexValue = list()
    ResultsTensorFlow = list(resultStreet, resultCity,resultRegion)
    for each result from the ResultsTensorFlow do
        if spellingMistake = 0 then
            Increment ResultsTensorFlow[index] by 1
        else
            Set CountOfMisstake[index] = ResultsTensorFlow[index]
        end if
    end for
    commonIndex = set(resultStreet) & set(resultCity) &
    set(resultRegion) & set(resultZip)
    Set CountOfCommonIndices = dict()
    if not commonIndex then
        ResultsListAll = list(resultStreet, resultCity,resultRegion, resultZip)
        for each ResultListAvt from the ResultsListAll do
            for each index from the ResultListAvt do
                if index in keys of CountOfCommonIndices then
                    Increment CountOfCommonIndices[index] by 1
                else then
                    Set CountOfCommonIndices[index] = 1
                end if
            end for
        end for
        Set FrequentIndexMost  = maximum of values of CountOfCommonIndices
        cleaningGuide = get the most frequent indices from
        FrequentIndexMost
        for each recommendation from the cleaningGuide do
            print avtentData[recommendation]
        end for
    end if
end for
```

Рисунок 2.12 – Псевдокод методу очищення даних на основі відповідних залежностей

2.7 Висновоки

У цьому розділі було розглянемо метод очищення даних на основі нещодавно запропонованої техніки відповідності залежностей. Було розглянуто основи методу, а саме алгоритми виявлення залежностей та відновлення даних.

Також розглянуто визначення відповідних залежностей та представлені основи, необхідні для виявлення MD. Вивчено поняття тривіальності і мінімальні властивості MD, та розглянуто частковий порядок MD для структурування простору пошуку.

Проаналізовано дві стратегії пошуку: обхід решітки та висновок із пар записів. Хоча обидва алгоритми вже є повністю автономними рішеннями для виявлення MD, але вони також мають свої слабкі сторони, які було мінімізовано за рахунок втілення вдосконаленого алгоритму пошуку, який поєднував ці дві стратегії.

Підсумовуючи вище зазначене, ми змогли зробити висновки та вдосконалити метод очищення даних на основі залежностей за рахунок:

– було втілено вдосконалений гібридний алгоритм пошуку залежностей, який використовує переваги обох методів пошуку та пом’якшує слабкі сторони;

− запропоновано використання машинного навчання для збільшення ефективності методу за рахунок вдосконаленого алгоритму перевірки та виправлення вхідних даних для очищення даних.

Наступним етапом розробки дипломного проекту є програмна реалізація програмного засобу на основі вдосконаленого методу очищення даних на основі техніки відповідних залежностей.

3. РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

3.1 Розроблення вимог до програмного засобу для вирішення задачі

Отже, завдання для розробки програмного проекту полягає в розробці програмного засобу, який би реалізував метод на основі техніки відповідності залежностей та її вдосконалення для очищення даних.

Як було сказано областю реалізації було обрано адреси України, які будь-який користувач, що проживає в будь-де, може вводити в Інтернеті. Цей засіб повинен ілюструвати, як ми можемо уникнути помилок в адресах України, введених будь-яким користувачем системи.

Програмний засіб повинен досягати максимальної точності у виправленні адреси у разі будь-якої помилки, яка включає:

- назва вулиці;
- місто;
- область;
- країну;
- поштовий індекс.

Засіб очищення даних повинен виконувати наступні функції, які повторюють реалізований алгоритм:

- приймати дані введені користувачем та автентифіковані дані;
- створювати відсортовані списки;
- переглядати кортежі адрес, для перевірки на справжність;
- продовжувати перегляд адрес для знаходження відповідностей;
- перевіряти та обробляти відповідні залежності для створення списку рекомендацій щодо очищення;
- виводити список рекомендацій щодо очищення даних.

Основним питанням яке позачергово стоїть перед розробкою програмного засобу – це вибір мови програмування для реалізації. Виходячи з цього питання було здійснено пошук високорівневих мов програмування, які дозволяють втілити програмний засіб та вирішити проблеми реалізації.

Розглянувши різні мови програмування, для реалізації дипломного проєкту було обрано Python як мову програмування, яка є об’єктно-орієнтованою, інтерпретованою мовою високого рівня з динамічною синематекою та зручним синтаксисом.

Одним з основних переваг Python є широкий спектр бібліотек для роботи з штучним інтелектом та масивами даних, які є основними інструментами, доступними для користувачів, які не потрібно буде розробляти для нових програмних продуктах.

Оскільки програмний засіб буде обробляти великі набори даних, а також використовувати машинне навчання, відкриті бібліотеки Python є дуже корисні для обробки на конвертації даних. Приклад деяких з них:

− tensorFlow, комплексна бібліотека з відкритим вихідним кодом для машинного навчання. Він має всеосяжну гнучку екосистему інструментів, бібліотек та ресурсів спільноти, яка дозволяє дослідникам просувати новітні досягнення у галузі машинного навчання;

− numpy бібліотека є одним із найкращих варіантів для виконання обчислень над матричними операціями. Він підтримує багатовимірні масиви. Над масивами можна виконувати велику кількість математичних і логічних операцій;

− python MIDI є бібліотекою, яка надає засоби для обробки, послідовності, запису та відтворення даних MIDI.

Python є простою та ефективною мовою у використанні при обробці великого обсягу даних та у використанні складних алгоритмів, яка має зручний для користувача синтаксис, що ефективно під час написання коду.

Щоб обрати найкращу IDE Python, було переглянуто список деяких популярних IDE Python на основі функціональності, кількості користувачів та інших оцінок, приклад деяких з них: Eclipse, Pycharm, Atom та VS Code.

Eclipse — це інтегроване середовище розробки (IDE), історично розроблене для мови Java. Проте, завдяки системі плагінів або розширень, його можна використовувати з іншими мовами програмування, включаючи

C/C++ та PHP. Pydev — це плагін, який дозволяє використовувати Eclipse як IDE Python, яка також підтримує Jython та IronPython. Pydev використовує передові методи висновку, щоб забезпечити такі елементи, як завершення коду та аналіз коду.

Pycharm — це інтегроване середовище розробки, розроблене JetBrains. Він виділяється серед конкурентів завдяки своїм інструментам продуктивності, таким як швидкі виправлення. Доступний у трьох версіях: версія для спільноти з ліцензією Apache, версія для освітніх (Edu) та запатентована професійна версія. Перші дві версії з відкритим кодом і, отже, безкоштовні, а професійна версія платна.

Atom включає більшість функцій базової IDE. Серед його функцій – підсвічування синтаксису та автозаповнення. Розробники Atom працюють над інтеграцією основних мов програмування, таких як Rust або Go. Atom прогресує у покращенні своєї продуктивності, і розробники надзвичайно уважні до потреб та думок спільноти, прагнучи зробити користувацький досвід більш корисним.

Розроблений компанією Microsoft для Windows, Linux та ОС, VS Code — це розширюваний редактор коду, який не слід плутати з Visual Studio. Насправді VS Code невеликий, але повний, і програмне забезпечення є відкритим вихідним кодом за ліцензією MIT. Ви можете додати нову мову до середовища, наприклад Python. Просто завантаживши та встановши відповідний плагін, щоб адаптувати його до середовища. VS Code доповнено такими функціями, як інтеграція потужного механізму автозаповнення коду (IntelliSense), консолі налагодження та терміналу для запуску команд сервера. VS Code в цілому дуже добре розроблений, і його головна перевага полягає в тому, що він пропонує архітектуру на основі розширення. Оскільки IDE є легкою, її можна розширити, додаючи послідовні компоненти за потреби.

Таким чином, Visual Studio Code є найкращим варіантом, оскільки він має найбільш оптимізоване середовище, з широким функціоналом та має низку необхідних бібліотек для написання коду.

Переваги обраного середовища:

- більше 4700 розширень;
- ротужний механізм керування кодом;
- імпорт на вимогу комбінацій клавіш з інших редакторів Python, таких як Sublime Text або Atom.

Недоліки:

- важко знайти розширення, яке найкраще відповідає вашим потребам, через тисячі доступних розширень.

Для проекту буде використана Visual Studio Code з оновленнями останнього року.

Таким чином, було сформовано вимоги до засобу. Це повиний бути десктопний засіб, який буде видавати рекомендації щодо очищення даних на основі покращеного методу відповідності залежностей.

3.2 Набори даних та їх опрацювання

Введені користувачем дані, які використовуються в цьому проекті, які містять списки адрес, створюються вручну. Незважаючи на те, що ці дані можуть не існувати в реальності, використовується такий самий формат, як і будь-яка існуюча адреса.

У наших зразкових даних ми врахували:

- назва вулиці,
- місто,
- область,
- країна,
- поштовий індекс.

Нижче наведено набори даних, які були використані як нечисті дані які введенні користувачем та зовнішні джерела даних у цьому проекті:

```
my_data =   [['вул. Соборна','Запоріжя','Запорізька область','Україна','69000'],
             ['вул. Радісна','Львів','Львівська область','Україна','79000'],
             ['вул. Надпільна','Черкаси','Черкська область','Україна','18210'],
             ['вул. Надпільна','Черкаси','Черкська область','Україна','18300'],
             ['вул.ФрунЗЕ','Харків','Київська область','Україна','61000'],
             ['ву. Учбова','Чернігів','Чернігівська область','Україна','14000'],
             ['вул. Стеценка','Луганськ','Луганська область','Україна','91050'],
             ['вул. Новий','Ужрод','Закарпатська область','Україна','88000']
]
```

Рисунок 3.1– Введені користувачем дані з неочищеними наборами даних

```
ext_data =  [['вул. Надпільна','Черкаси','Черкська область','Україна','18010'],
             ['вул. Надпільна','Черкаси','Черкська область','Україна','18015'],
             ['вул. Хрещатик','Київ','Київська область','Україна','01000'],
             ['вул. Радісна','Львів','Львівська область','Україна','79000'],
             ['вул. Соборна','Запоріжя','Запорізька область','Україна','69000',],
             ['вул. Північна','Кам'янець-Подільський','Хмельницька область','Україна','32300'],
             ['пр. Миру','Донецьк','Донецка область','Україна','83000'],
             ['вул. Фастівська','Херсон','Херсонська область','Україна','73000'],
             ['вул. Івана Франка','Херсон','Херсонська область','Україна','73100'],
             ['вул.Фрунзе','Харків','Харківська область','Україна','61000'],
             ['вул. Учбова','Чернігів','Чернігівська область','Україна','14000'],
             ['вул.Червона','Рівне','Рівненська область','Україна','33500'],
             ['пров. Ціолковського','Коломия','Івано-Франківська область','Україна','78000'],
             ['вул. Шпака','Вінниця','Вінницька область','Україна','21000'],
             ['вул. Ясна','Полтава','Полтавська область','Україна','37000'],
             ['вул. Стеценка','Луганськ','Луганська область','Україна','91000'],
             ['вул. Польова','Полтава','Полтавська область','Україна','37000'],
             ['вул. Підлипка','Харків','Харківська область','Україна','61500'],
             ['вул. Нова','Ужгород','Закарпатська область','Україна','88000'],
             ['вул. Ніжинська','Суми','Сумська область','Україна','40000']]
```

Рисунок 3.2 – Зовнішні дані, які мають автентифіковані набори даних адрес

Після виконання алгоритму на наведених вище наборах даних ми отримуємо наступні результати рекомендацій щодо очищення даних, які зображенні на рисунку 3.3:

```
PS E:\ХНU\Магістратура\Диплом\Program>  e:; cd 'e:\ХНU\Маг?стратура\Дип
Data\Local\Programs\Python\Python310\python.exe' 'c:\Users\Andriy\.vsco
11.1422169775\pythonFiles\lib\python\debugpy\launcher' '51606' '--' 'e:
test_20.py'
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул.ФрунЗЕ','Київська область','Харків','Україна','61000']
Рекомендовані дані:
['вул.Фрунзе','Харківська область','Харків','Україна','61000']
```

Рисунок 3.3 – Згенерований результат під час роботи алгоритму

Оскільки це невеликий набір результатів, можна побачити вручну, що результати мають два хибнопозитивних результатів і нуль хибних негативних результатів. Рекомендації, які мають хибнопозитивні та хибнонегативні результати, також містять відповідний результат. Результати аналізу наведені в таблиці 3.1:

Таблиця 3.1: Дані аналізу оцінки 20 записів

Розмір зовнішніх даних	20
Розмір тестових даних	8
Кількість помилок у тестових даних	6
Тип помилок	Назва вулиці/місто/область/країна/поштовий індекс
Кількість рекомендацій	6
Кількість помилкових спрацьовувань	2
Кількість помилкових негативних результатів	0
Витрачений час	0,1 секунда

Помилкові спрацювання – це кількість вхідних записів, які мають більше однієї рекомендації. Помилково-негативні результати – це кількість вхідних записів, які є дійсними, але автентифіковані дані не містять інформації про ці записи.

3.3 Програмна реалізація

Тепер коли отримано усі необхідні проектуючи компоненти, можна переходити до програмної реалізації дипломного проекту. Програмний засіб складається з декількох модулів, які взаємодіють один з одним, щоб в кінцевому результаті рекомендувати адреси.

Засіб складається з таких основних модулів:

- опрацювання вхідної інформації;
- перевірка даних за допомогою машинного навчання;
- модуль пошуку відповідних залежностей;
- виведення рекомендацій щодо очищення даних.

Розглянуто буде два основних модулі, перевірки та пошуку.

Важливим модулем програмного засобу, який буде описаний це перевірка орфографії за допомогою машинного навчання. Дані для тренування

моделі складаються з даних двадцяти наборів, які були аналогічно згенеровані як у підпункті 3.2. Єдина відмінність що будуть використовуватися лише вулиці, області та міста.

Реалізація засобу складається з таких етапів:

- завантаження даних;
- підготовка даних;
- побудова моделі;
- навчання моделі;
- виправлення користувацьких даних.

Завантаження даних. Для початку потрібно зібрати усі назви файлів наборів даних. Завантажити дані, використовуючи імена файлів. Порівняти кількість слів у кожному наборі, та перевірити чи текст виглядає правильно.

Фрагмент коду:

```
def load_data(path):
    """Load a data from its file"""
    input_file = os.path.join(path)
    with open(input_file) as f:
        data = f.read()
    return data
path = './datas/'
data_files = [f for f in listdir(path) if isfile(join(path, f))]
data_files = data_files[1:]
datas = []
for data in data_files:
    datas.append(load_data(path+data))
for i in range(len(datas)):
    print("There{}{}.".format(len(data[i].split()),
data_files[i]))
datas[0][:500]
```

Підготовка даних. По-перше потрібно очистити тест наборів даних. Переконатися, що текст очищений належним чином. Створити словник для перетворення символів у цілі числа. Додати спеціальні маркери до vocab_to_int. Перевірити розмір словникового запасу та всі значення. Створити інший словник для перетворення чисел у відповідні символи. Розділити дані із наборів на окремі області даних: міста, області, вулиці.

Перевірити, чи правильно розділено текст. Перетворити зразки даних на цілі числа. Знайти довжину значень кожного стовпця(місто,область,вулиця). Обмежити дані, які ми будемо використовувати для навчання нашої моделі.

Розділити дані для навчання та тестування. Відсортувати стовбці за довжиною, щоб зменшити заповнення, що дозволить моделі тренуватися швидше. Перевірити, щоб все було вибрано та відсортовано правильно.

Фрагмент коду реалізації частини описаної вище:

```
letters = ['а','б','в','г','ґ ','д','е','є','ж','з','и','і','ї',
'й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ш','щ',
'ь','ю','я']
def noise_maker(adress, threshold):
    noisy_adress = []
    i = 0
    while i < len(adress):
        random = np.random.uniform(0,1,1)

        if random < threshold:
            noisy_adress.append(adress[i])
        else:
             __new___random = np.random.uniform(0,1,1)

            if  __new___random > 0.67:
                if i == (len(adress) - 1):

                    continue
                else:
                    noisy_adress.append(adress[i+1])
                    noisy_adress.append(adress[i])
                    i += 1
            elif  __new___random < 0.33:
                random_letter = np.random.choice(letters, 1)[0]
                noisy_adress.append(vocab_to_int[random_letter])
                noisy_adress.append(adress[i])

            else:
                pass
        i += 1
    return noisy_adress
```

Наступний код буде відображати частину побудови моделі навчання:

```
def   training_decoding_layer(dec_embed_input,   targets_length,
dec_cell, initial_state, output_layer,
                              vocab_size, max_target_length):
    '''Create the training logits'''
```

```
    with tf.name_scope("Training_Decoder"):
        training_helper                                          =
tf.contrib.seq2seq.TrainingHelper(inputs=dec_embed_input,
sequence_length=targets_length, time_major=False)
training_decoder    =    tf.contrib.seq2seq.BasicDecoder(dec_cell,
training_helper, initial_state,output_layer)
training_logits,                                 _               =
tf.contrib.seq2seq.dynamic_decode(training_decoder,
output_time_major=False,
impute_finished=True,
maximum_iterations=max_target_length)
        return training_logits
def  seq2seq_model(inputs,  targets,  keep_prob,  inputs_length,
targets_length, max_target_length,
                   vocab_size, rnn_size, num_layers, vocab_to_int,
batch_size, embedding_size, direction):
    '''Use the previous functions to create the training and
inference logits'''

    enc_embeddings  =  tf.Variable(tf.random_uniform([vocab_size,
embedding_size], -1, 1))
    enc_embed_input    =    tf.nn.embedding_lookup(enc_embeddings,
inputs)
    enc_output,     enc_state     =     encoding_layer(rnn_size,
inputs_length, num_layers,
                                            enc_embed_input,
keep_prob, direction)

    dec_embeddings  =  tf.Variable(tf.random_uniform([vocab_size,
embedding_size], -1, 1))
    dec_input  =  process_encoding_input(targets,  vocab_to_int,
batch_size)
    dec_embed_input    =    tf.nn.embedding_lookup(dec_embeddings,
dec_input)

    training_logits,          inference_logits                   =
decoding_layer(dec_embed_input,
               dec_embeddings, enc_output,
               enc_state,
               vocab_size,
               inputs_length,
               targets_length,
               max_target_length,
               rnn_size,
               vocab_to_int,
               keep_prob,
               batch_size,
               num_layers,
               direction)
return training_logits, inference_logits
```

Тренування моделі з потрібними параметрами настройки:

```
for keep_probability in [0.75]:
    for num_layers in [2]:
        for threshold in [0.95]:
            log_string                                                    =
'kp={},nl={},th={}'.format(keep_probability,
                                                        num_layers,
                                                        threshold)
            model   =   build_graph(keep_probability,   rnn_size,
num_layers, batch_size,
                                learning_rate,    embedding_size,
direction)
            train(model, epochs, log_string)
```

Фрагмент коду, який реалізує етап виправлення вхідних даних:

```
model   =   build_graph(keep_probability,   rnn_size,   num_layers,
batch_size, learning_rate, embedding_size, direction)

with tf.Session() as sess:
    saver = tf.train.Saver()
    saver.restore(sess, checkpoint)
    answer_logits   =   sess.run(model.predictions,   {model.inputs:
[text]*batch_size,  model.inputs_length:  [len(text)]*batch_size,
model.targets_length: [len(text)+1], model.keep_prob: [1.0]})[0]
```

Приклади орфографічних виправлень даних для подальшого використання при побудові рекомендацій зображені на рисунку 3.4.

```
-2021.11.1422169775\pythonFiles\lib\python\debugpy\launcher' '62717' '--' 'e:\ХНУ\Маг?стратура\Диплом\Program\TensorFlow.py'
['вул. Соборн','Запоріжя','Запорізька область','Україна','69000']
['вул. Соборна','Запоріжя','Запорізька область','Україна','69000']

['вул. Радісна','Львів','Львівська область','Україна','79000']
['вул. Радісна','Львів','Львівка область','Україна','79000']

['вулиця Учбова','Чернігов','Чернігівська область','Україна','14000']
['вул. Учбова','Чернігів','Чернігівська область','Україна','14000']
PS E:\ХНУ\Магістратура\Диплом\Program>
```

Рисунок 3.4– Зразок даних, які виправлені та будуть використовуватися

Іншим важливим модулем засобу є модуль пошуку відповідних залежностей з використанням запропонованого вдосконаленого алгоритму.

Фрагмент коду з реалізацією класів:

```
def MD {
```

```
    def MDSite getLhs()
    def MDElement getRhs()

    def boolean isInLhs(int attr) {
        return getRhs().isSet(attr)
    }

    def boolean isRhs(int attr) {
        return getRhs().getId() == attr
    }
}
def HybridMD [MatchingDependencyAlgorithm,
     RelationalInputParameterAlgorithm, StringParameterAlgorithm,
IntegerParameterAlgorithm] {

  static {
    CPSTypeIdResolver.addClassLoader(HyMD.class.getClassLoader())
  }
  def MatchingDependencyResultReceiver resultReceiver
  def Relation leftRelation
  def Relation rightRelation
  def double minThreshold = 0.7
    def   SupportCalculator   supportCalculator   =      __new__
SizeBasedSupportCalculator()
   def  MappingConfiguration  mappingConfiguration  =    __new__
MappingConfiguration()
```

Фрагмент коду програми, який реалізує створення решітки обходу:

```
def MDMapping implements Hashable {
    def List<ColumnConfiguration<?>> mappings
    def hash(Hasher hasher) {
        hasher.putAll(mappings)
    }
                          def           PreprocessingConfiguration
getPreprocessingConfiguration() {
         List<PreprocessingColumnConfiguration<?>> configurations
= mappings.stream()
             .map(ColumnConfiguration::getPreprocessingConfigurat
ion)
            .collect(Collectors.toList())
                                     return                 __new__
PreprocessingConfiguration(configurations);
    }
    def List<ThresholdFilter> getThresholdFilters() {
        return StreamUtils.seq(mappings)
            .map(ColumnConfiguration::getThresholdFilter)
            .toList()
    }
}
```

```
def main execute() throws AlgorithmExecutionException {
    if (leftRelation == null) {
      throw  __new__ AlgorithmConfigurationException("")
      #Потрібне хоча б одне відношення
    }
    if (rightRelation == null) {
                              MDMapping        mappings        =
mappingConfiguration.createMapping(leftRelation)
      Discoverer discoverer = createDiscoverer(mappings)
      discoverer.discover(leftRelation)
    } else {
                              MDMapping        mappings        =
mappingConfiguration.createMapping(leftRelation, rightRelation)
      Discoverer discoverer = createDiscoverer(mappings)
      discoverer.discover(leftRelation, rightRelation)
    }
  }
def setRelationalInputConfigurationValue(String identifier,
                RelationalInputGenerator...    values)    throws
AlgorithmConfigurationException {
    if (Identifier.RELATION.name().equals(identifier)) {
      if (values.length == 0 || values.length > 2) {
        throw  __new__ AlgorithmConfigurationException(
            "");
             #MD Discovery очікує один або два реляційні вхідні
дані
      }
      this.leftRelation = wrap(values[0])
      if (values.length == 2) {
        this.rightRelation = wrap(values[1])
      } else {
        this.rightRelation = null;
      }
    }
  }
def SimilarityIndexBuilder extends Hashable {

        <T> SimilarityIndex create(Dictionary<T> leftDictionary,
Dictionary<T> rightDictionary,
                          SimilarityMeasure<T>   similarityMeasure,
SimilarityComputer<T> similarityComputer,
            double minThreshold, PositionListIndex rightIndex)
    }
```

Повний програмний код розробленої програмної системи наведено в додатку А.

3.4 Висновки

У розділі описано програмну реалізацію засобу, обрано мову програмування для його реалізації, основні бібліотеки, необхідні для його роботи, середовище розробки, описано набір даних для тестування розроблених систем, описано способи обробки даних.

За мову реалізації системи вирішено обрати мову Python, оскільки вона якнайкраще підходить для наших задач та має велику кількість готових бібліотек, які містять більшість необхідних нам алгоритмів та функцій.

Як середовище розробки вирішено обрати Visual Studio Code, оскільки воно забезпечує зручний процес розробки програмного забезпечення, має вбудований широкий функціонал для тестування та відлагоджування як окремих програмних компонентів так і усієї системи у цілому.

Як тестовий набір даних, вирішено обрати набір даних, з п’яти компонентів. Також усі дані було вирішено генерувати за допомогою спеціальних засобів та відкритих даних. Цей самий набір даних буде використовується для тренування моделі машинного навчання для перевірки правильності вхідних даних користувача.

4.ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ УДОСКОНАЛЕННЯ МЕТОДУ ОЧИЩЕННЯ ДАНИХ НА ОСНОВІ MATCHING DEPENDENCY TECHNIQUE

4.1 Генерація тестових даних

Першим завданням для оцінки було мати велику кількість даних для перевірки реалізованої та вдосконаленої техніки. Ці дані мають бути в такому форматі для України: [Назва вулиці, місто, область, країна, поштовий індекс]. Дані, згенеровані в Інтернеті, являли собою повну адресу, яка містила всю адресу замість назви вулиці (показано на Рисунок 4.1). Після застосування генерації ми отримаємо 1000 адрес у списку формату JSON. Дані для дослідження генерувалися за допомогою сервісу генерації тестових даних[21]. Наступним завданням було виправити проблему з адресою, іншими словами, мені потрібно було перетворити адресу вулиці в назву вулиці.


```
[{'address':'9515 Nulla Avenue','country':'Україна',
    'region':'Cherkasy Oblast','city':'Cherkasy','postalZip':'18000'},
{'address':'Ap #639-1134 Diam St.','country':'Україна',
    'region':'Donetsk Oblast','city':'Donetsk','postalZip':'83000'},
{'address':'500-5100 Feugiat. Rd.','country':'Україна',
    'region':'Ivano-Frankivsk Oblast','city':'Ivano-Frankivsk','postalZip':'76000'},
{'address':'238-5339 Tortor. Rd.','country':'Україна',
    'region':'Kharkiv Oblast','city':'Kharkiv','postalZip':'61000'},
{'address':'P.O. Box 578, 646 Nec Av.','country':'Україна',
    'region':'Kyiv City','city':'Kyiv','postalZip':'01000'},
{'address':'392-3219 Quis Rd.','country':'Україна',
    'region':'Lviv Oblast','city':'Lviv','postalZip':'79000'},
{'address':'948 Fermentum Rd.','country':'Україна',
    'region':'Odesa Oblast','city':'Odesa','postalZip':'65000'},
{'address':'Ap #673-8972 Tincidunt Road','country':'Україна',
    'region':'Khmelnytskyi Oblast','city':'Khmelnytskyi','postalZip':'29000'},
{'address':'9149 Id, Road','country':'Україна',
    'region':'Khmelnytskyi Oblast','city':'Kamyanets-Podilsky','postalZip':'32300'},
{'address':'Ap #396-3577 Arcu. Rd.','country':'Україна',
    'region':'Zaporizhzhia Oblast','city':'Zaporizhzhia','postalZip':'69000'}]
```


Рисунок 4.1– Зразок даних, згенерованих онлайн

Далі завантажуємо спеціальний згенерований список із 1000 назв вулиць для України. Список вулиць було взято за допомогою сервісу geographic[22]. Зразок списку показано на Рисунку 4.2:

- "Перемога" селище 3148
- 14-й км Шляху 3115
- 15-й км Житомирського шосе 3179
- 16-й км Ніжинського шосе 2099
- 19-й км Житомирського шосе 3179
- 1-го Травня вул. 3187
- 1-го Травня вул.-Бортничи 2088
- 21-й км Київ-Дніпровс.ПЗЗ 2002
- 23-й км Харківського шосе 2121
- 8-го Березня вул. 4077
- Авдєєнка Генерала вул. 3148
- Автозаводська вул. -
- Автопаркова вул. 2121
- Автотранспортна вул.-Бортничи 2088
- Агітаторська вул. 3035
- Агрегатна вул. 4074
- Аеровокзальна вул.(Ернста Ф) 3151
- Аеродромна вул. 3151
- Аеропорт Жуляни 3036
- Азовська вул. 3037
- Айвазовського пров. 4107

Рисунок 4.2 – Зразок вулиць, згенерованих онлайн

Наступною дією було перетворення наведених вище текстових даних у список даних програмно. Зразок списку показано на малюнку 4.3:

```
['пров. Айвазовського', ' вул. 1-го Травня', ' вул. Берегова',
    'вул. Волочаївська', ' вул. Газова',
    'вул. Газопровідна', 'вул. Нагорна', 'вул. Осіння',
    'вул. Патріотів', ' вул. Радісна',
    ' вул. Садова', 'вул. Салютна', 'вул. Українська']
```

Рисунок 4.3 – Зразок списку назв вулиць, згенерованих в Інтернеті

Набір адресних даних містив цілу адресу, яку було програмно оновлено зі списком назв вулиць, показаним на наведених вище рисунках.

Для даних з 1000 записів, список назв вулиць був послідовно замінений на список з українських адрес. Також було програмно перекладено назву країни, області та міста.

Зразок записів адрес із оновленими полями показано на рисунку 4.4:


```
[{'address':'вул. Надпільна','country':'Україна',
    'region':'Черкаська область','city':'Черкаси','postalZip':'18000'},
{'address':'пр. Миру','country':'Україна',
    'region':'Донецька область','city':'Донецьк','postalZip':'83000'},
{'address':'вул. Богдана Лепкого','country':'Україна',
    'region':'Івано-Франківська область','city':'Івано-Франківськ','postalZip':'76000'}
{'address':'вул. Шевченка','country':'Україна',
    'region':'Харківська область','city':'Харків','postalZip':'61000'},
{'address':'пров. Айвазовського','country':'Україна',
    'region':'Київська область','city':'Київ','postalZip':'01000'},
{'address':'вул. Радісна','country':'Україна',
    'region':'Львівська область','city':'Львів','postalZip':'79000'},
{'address':'вул. Салютна','country':'Україна',
    'region':'Одеська область','city':'Одеса','postalZip':'65000'},
{'address':'вул. Прибузька','country':'Україна',
    'region':'Хмельницька область','city':'Хмельницький','postalZip':'29000'},
{'address':'вул. Північна','country':'Україна',
    'region':'Хмельницька область','city':'Кам'янець-Подільський','postalZip':'32300'},
{'address':'вул. Соборна','country':'Україна',
    'region':'Запорізька область','city':'Запоріжжя','postalZip':'69000'}]
```


Рисунок 4.4 – Зразок автентифікованих даних JSON для оцінки

Дані, показані на рисунку 4.4, розглядалися як автентифіковані дані для оцінки. Для введених користувачем даних, які можуть містити неправильно написані/неправильні значення в наборі, копіюємо випадковий набір даних програмно з автентифікованих даних і зберіг його окремо в іншому файлі JSON. Оскільки введені користувачем дані отримуються з автентифікованих даних, було зроблено вручну помилки випадковим чином у введених користувачами даних JSON. Зразок введених користувачем даних із неправильними значеннями наведено нижче на рисунку 4.5:

```
[{'address':'вул. Надплна','country':'Україна',
    'region':'Черкаська область','city':'Черкаси','postalZip':'18000'},
{'address':'пр. Миру','country':'Украна',
    'region':'Донецька область','city':'Донецьк','postalZip':'67000'},
{'address':'вул. Богдана Лепкого','country':'Україна',
    'region':'Івано-Франківська область','city':'Івано-Франківськ','postalZip':'76000'},
{'address':'вул. Шевченка','country':'Україна',
    'region':'Харківська область','city':'Харків','postalZip':'61000'},
{'address':'пров. Айвазовського','country':'Україна',
    'region':'Київка область','city':'Київ','postalZip':'01000'},
{'address':'вул. Радісна','country':'Україна',
    'region':'Львівська область','city':'Львів','postalZip':'79000'},
{'address':'вул. Салютна','country':'Україна',
    'region':'Одеська область','city':'Одеса','postalZip':'65000'},
{'address':'вул. Прибизька','country':'Україна',
    'region':'Хмельницька область','city':'Хмельницький','postalZip':'11100'},
{'address':'вул. Північна','country':'Україна',
    'region':'Хмельницька область','city':'Кам'янець-Подільський','postalZip':'32300'},
{'address':'вул. Соборна','country':'Україна',
    'region':'Хмельницька область','city':'Запоріжжя','postalZip':'69000'}]
```


Рисунок 4.5 – Зразок тестових даних JSON для оцінки

4.2 Оцінка

Тестові дані JSON містять комбінацію назви вулиці, міста, країни, області та поштового індексу. Після запуску реалізованої методики щодо файлів JSON тестових даних і даних аутентифікації ми отримуємо результати, показані на рисунку 4.6, рисунку 4.7:

```
PS E:\XHU\Магістратура\Диплом\Program>  e:; cd 'e:\XHU\Маг?стратура\Диплом\Prog
Data\Local\Programs\Python\Python310\python.exe' 'c:\Users\Andriy\.vscode\exte
11.1422169775\pythonFiles\lib\python\debugpy\launcher' '63434' '--' 'e:\XHU\Ma
test_1000.py'
Фінальні результати:
Рекомендації до:
['вул. Борва','Луганськ','Луганська область','Україна','91050']
Рекомендовані дані:
['вул. Борова','Луганськ','Луганська область','Україна','91000']
--------------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Героїв Дніпра','Терпіль','Тернопільська область','Україна','47050']
Рекомендовані дані:
['вул. Героїв Дніпра','Тернопіль','Тернопільська область','Україна','47050']
--------------------------------------------------------------------------
```

Рисунок 4.6 – Результати оцінювання

```
Фінальні результати:
Рекомендації до:
['вулиця Захіа','Рівно','Рівненська область','Україна','33000']
Рекомендовані дані:
['вул. Західна','Рівно','Рівненська область','Україна','33000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Кобзарська','Чернігів','Чернівецька область','Україна','58000']
Рекомендовані дані:
['вул. Кобзарська','Чернігів','Чернівецька область','Україна','58500']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['пров. Кобзарський','Вінниця','Запорізька область','Україна','21050']
Рекомендовані дані:
['пров. Кобзарський','Вінниця','Вінницька область','Україна','21050']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Моторна','Суми','Сумська область','Україна','55800']
Рекомендовані дані:
['вул. Моторна','Суми','Сумська область','Україна','40000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Ніжинськала','Хмельницький','Хмельницька область','Україна','33000']
Рекомендовані дані:
['вул. Ніжинська','Хмельницький','Хмельницька область','Україна','33000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Охотська','Київ','Хмельницька область','Україна','07000']
Рекомендовані дані:
['вул. Охотська','Київ','Київська область','Україна','07000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Панельна','Луцьк','Волонська область','Україна','43000']
Рекомендовані дані:
['вул. Панельна','Луцьк','Волинська область','Україна','43000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Сквирська','Чернігів','Чернігівська область','України','14050']
Рекомендовані дані:
['вул. Сквирська','Чернігів','Чернігівська область','Україна','14050']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Тиха','Біла Церква','Київська область','Україна','09500']
Рекомендовані дані:
['вул. Тиха','Біла Церква','Київська область','Україна','09500']
--------------------------------------------------------------------
```

Рисунок 4.7 – Результати оцінювання

Фактичний результат набагато довший, ніж результати, показані на наведених вище малюнках. Аналіз вищевказаного тесту наведено в таблиці 4.1:

Таблиця 4.1 – аналіз тестової оцінки

Розмір зовнішніх даних	1000
Розмір тестових даних	97
Кількість помилок у тестових даних	41
Тип помилок	Назва вулиці/місто/область/країна/поштовий індекс
Кількість рекомендацій	41
Кількість помилкових спрацьовувань	2
Кількість помилкових негативних результатів	0
Витрачений час	0,45 секунди

Оскільки використовується гібридний алгоритм пошуку, його потрібно порівняти з базовими методу пошуку залежностей . Запропонований алгоритм поєднує обхід решітки з висновком з пар записів. Оскільки обидві стратегії здатні виявляти всі MD самостійно, порівнюємо їхню продуктивність з продуктивністю їх гібридної комбінації. У наборі даних в 1000 кортежів було заміряно час створення рекомендацій щодо очищення даних: 1,1 секунди (обхід), 3 секунди (висновок) та 0,89 секунди (гібридний); У наборі з 15000 записів: 16,2 секунди (обхід), 42,2 секунди (висновок) та 13 секунди (гібридний). Результати зображені у таблиці 4.2. Оскільки обидва набори даних відносно короткі, стратегія висновку явно перевершує стратегію обходу. Проте гібридна стратегія перевершує обидва індивідуальні підходи.

Усі оцінки проводилися на персональному ноутбуці з такими конфігураційними параметрами:

– процесор Intel Core i7-8550U CPU @ 1,8 ГГц;
– встановлена фізична пам’ять (RAM) 8 ГБ;
– операційна система Microsoft Widows 10 LTSC 1809 і версія Python 3.10.

Таблиця 4.2 – порівняльний аналіз алгоритмів пошуку залежностей

Тип алгоритму пошуку	Розмір зовнішніх даних	Розмір тестових даних	Кількість помилок у тестових даних	Витрачений час (с.)
Гібридний алгоритм	1000	150	87	0,89
Обхід решітки	1000	150	87	1,1
Висновок з пар записів	1000	150	87	3
Гібридний алгоритм	15000	1500	389	13
Обхід решітки	15000	1500	389	16,2
Висновок з пар записів	15000	1500	389	42,2

4.3 Оцінка великих даних

Спочатку було створено набір даних із 1000 записів, щоб провести оцінку та перевірити реалізовану техніку. Іншою віхою було перевірити його з набором даних, більшим за початкову оцінку. Цього разу таким же чином створено 15000 адресних записів для України. Згодом програмно виведено 1500 записів з JSON великих даних (кожен 10-й запис) і збережено їх окремо в іншому файлі JSON, який буде використовуватися як дані для введення користувача.

Наразі введені користувачем дані мають чисті записи. Тому програмно вставлені неправильні значення у введені користувачем дані таким чином:

– Кожна 5-а назва вулиці має значення «погана_назва_вулиці»

– Кожна 10-та назва міста має значення «погана_назва_міста»

– Кожна 20-а назва області має значення «погана_назва_області»

– Кожна 40-та назва країни має значення «погана_назва_країни»

– Кожен 12-й поштовий індекс встановлюється на «неправильний_індекс».

Якщо всі значення адреси неправильні в даних, що вводяться користувачем, то програмний засіб видасть повідомлення про помилку через його конструкцію. Було використано функцію max() в Python, яка видасть ValueError, якщо для нього буде надано порожню послідовність. Таким чином,

створюючи введені користувачем дані, переконуємося, що жоден запис адреси не є повністю неправильним. Результати оцінки з великим набором даних показано на рисунку 4.8:

```
PS E:\ХНУ\Магістратура\Диплом\Program>  e:; cd 'e:\ХНУ\Маг?стратура\Диплом\Program'; & 'C:\Users\Andriy\A
ocal\Programs\Python\Python310\python.exe' 'c:\Users\Andriy\.vscode\extensions\ms-python.python-2021.11.1
5\pythonFiles\lib\python\debugpy\launcher' '63495' '--' 'e:\ХНУ\Маг?стратура\Диплом\Program\MD_test_15000
Фінальні результати:
Рекомендації до:
['погана_назва_вулиці','погана_назва_міста','Рівненська область','Україна','33800']
Рекомендовані дані:
['пров. Черешневий','Рівно','Рівненська область','Україна','33800']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['погана_назва_вулиці','погана_назва_міста','Одеська область','Україна','неправильний_індекс']
Рекомендовані дані:
['вул. Янтарна','Одеса','Одеська область','Україна','65000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['погана_назва_вулиці','Кропивницький','Кіровоградська область','Україна','25750']
Рекомендовані дані:
['проспект Броварський','Кропивницький','Кіровоградська область','Україна','25750']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['вул. Брюлова','Ужгород','погана_назва_області','Україна','73000']
Рекомендовані дані:
['вул. Замковецька','Херсов','Херсонська область','Україна','73000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['погана_назва_вулиці','погана_назва_міста','Харківська область','погана_назва_країни','61000']
Рекомендовані дані:
['вул. Брестська','Харків','Харківська область','Україна','61000']
--------------------------------------------------------------------
Фінальні результати:
Рекомендації до:
['погана_назва_вулиці','Ужгород','Полтавська область','Україна','неправильний_індекс']
Рекомендовані дані:
['вул. Гребінки','Полтава','Полтавська область','Україна','36000']
```

Рисунок 4.8 – Результати оцінювання великих даних

Наведені вище результати є початковою частиною фактичного результату, який має величезний розмір. Аналіз тесту з 15000 записів наведено в таблиці 4.3:

Таблиця 4.3 – аналіз тестової оцінки

Розмір зовнішніх даних	15000
Розмір тестових даних	1500
Кількість помилок у тестових даних	389
Тип помилок	Назва вулиці/місто/область/країна/поштовий індекс
Кількість рекомендацій	390
Кількість помилкових спрацьовувань	8
Кількість помилкових негативних результатів	0
Витрачений час	13 секунд

Також був проведений аналіз з багатьма іншими наборами даних різного розміру, від 2000 до 14000. Час, який зафіксований для всіх оцінок, продемонстровано на рисунку 4.9:

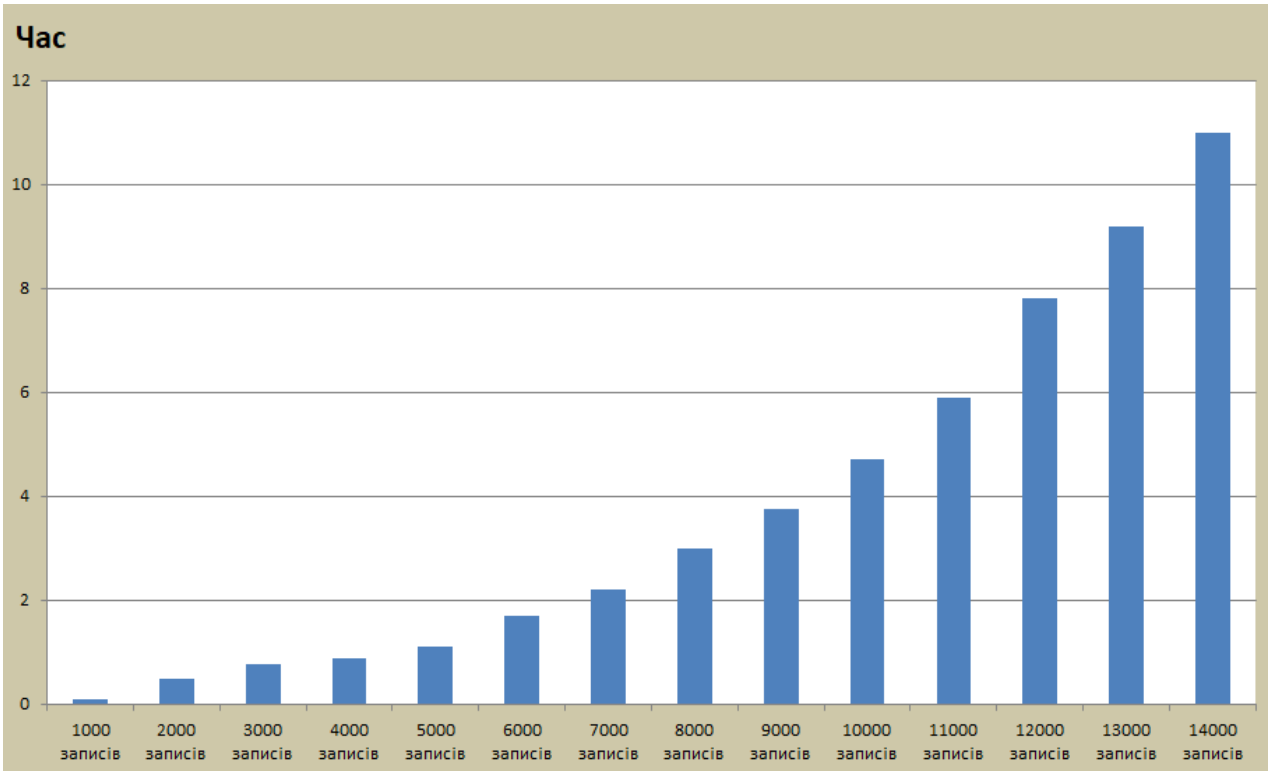


Рисунок 4.9 – Аналіз часу для різних розмірів наборів даних

Наведені нижче дані в таблиці 4.4 результатів дослідження різних розмірів даних при тих самих помилках в даних що і в менших наборах, а саме: назва вулиці, місто, область, країна, поштовий індекс.

Таблиця 4.4 – Дані аналізу оцінки для різних розмірів наборів даних

Розмір зовнішніх даних	Розмір тестових даних	Кількість помилок у тестових даних	Кількість рекомендацій	Кількість помилкових спрацьовувань	Кількість помилкових негативних результатів	Витрачений час (с.)
1000	100	27	27	0	0	0,1
2000	200	53	53	0	0	0,5
3000	300	81	81	1	0	0,76
4000	400	108	108	2	0	0,87
5000	500	134	134	2	0	1,1
6000	600	158	158	2	0	1,71
7000	700	185	185	3	0	2,2
8000	800	211	211	3	0	2,98
9000	900	239	239	2	0	3,76
10000	1000	255	255	4	0	4,7
11000	1100	301	301	6	0	5,9
12000	1200	325	325	5	0	7,8
13000	1300	344	344	4	0	9,2
14000	1400	394	394	6	0	11,3
15000	1500	399	399	8	0	14

4.4 Висновки

Для дослідження та оцінки методу у цьому розділі було створено тестові набори даних для перевірки реалізованого засобу, та ефективності методу.

Також розділі було проведене дослідження та оцінка втіленого методу очищення даних на основі відповідних залежностей. Результати тестування показують, що вдосконалений метод на основі відповідних залежностей має високу швидкість і ефективно здійснює пошуй та рекомендації адрес.

В результаті експериментальних дослідження запропонованого методу, було з’ясовано, що він має високу швидкість, без втрачання ефективності, за рахунок використання гібридного алгоритму пошуку, який перевершує обхід решітки та висновок із пар записів.

ВИСНОВОК

Дослідження сучасних методів виявлення та відновлення даних та особливості реалізації засобів надали можливість виокремити ключові недоліки сучасних способів очищення даних.

В результаті виконання дипломної роботи було реалізовано та вдосконалено метод, розроблено та протестовано засіб очищення даних на основі Matching Dependency Technique.

У першому розділі було проведено аналіз з роботами про машинне навчання та методи дослідження очищення даних. Виконано аналіз літератури та існуючих рішень та обрано метод очищення даних, який якнайкраще підходить для виявлення та відновлення даних для нашого завдання. Який проаналізовано на предмет наявних у ньому недоліків та описано способи їх усунення. Техніка відповідності залежностей здавалася багатообіцяючою для процесу очищення даних.

У другому розділі розглядається метод очищення даних на основі техніки відповідності залежностей. Було проаналізовано ряд алгоритмів пошуку залежностей. На їх основі було запропоновано гібридний метод пошуку відповідних залежностей. Втілена техніка машинного навчання, яка дозволила використовувати «брудні дані», як дійсні, для перевірки інших записів для того самого адресу.

Третій розділ присвячений програмній реалізації засобу надання рекомендацій щодо очищення даних. Техніка була реалізована мовою Python, самостійно створені набори даних, дані введені користувачем, та дані, що перевіряються, а також модель для виправлення помилок.

У четвертому розділі було проведено оцінка та аналіз реалізованої техніки. Для процесу оцінки було створено великі набори даних різного розміру, від 1000 до 15000. Для глибокого аналізу оцінено час, розмір введених користувачем/автентифікованих наборів даних, кількість помилок у введених користувачем даних, помилкових спрацьовувань, помилкових негативів і тип

помилок для різних розмірів даних. Також була проведена порівняльна оцінка видачі точності рекомендацій, сформованих базовими алгоритмами, та відповідна точність для гібридного алгоритму, яка показала, що модифікований алгоритм рекомендацій адрес має більшу швидкість у порівнянні з базовими алгоритмами, без втрачання точності рекомендацій.

Зроблено наступні висновки з реалізації та вдосконаленню методу на основі техніки відповідності залежностей:

– це може бути одним із корисних методів очищення будь-якого набору даних, який має погані вхідні дані;

– він здатний обробляти великі набори даних, оскільки використовується гібридний алгоритм пошуку, що покращує ефективність процесу пошуку;

– процес очищення набору даних став легшим, оскільки впроваджена та вдосконалена методика виявляє нечисті набори даних, а потім дає відповідні можливі рекомендації.

Таким чином, в результаті виконання дипломної роботи магістра було вдосконалено ряд алгоритмів та засобів, в результаті чого збільшилась ефективність методу очищення даних на основі техніки відповідності залежностей.

За темою і результатами дипломної роботи опубліковані тези доповіді на всеукраїнській науково-практичній конференції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Біловол А.І. Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique/ Біловол А.І. // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». Хмельницький – 2021. – 287-290с.

2. T. Johnson and T. Dasu. Data quality and data cleaning: An overview. In SIGMOD, page 681, 2003

3. E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. IEEE Data Eng. Bull., 23(4), 2000.

4. P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In SIGMOD, pages 143–154. ACM, 2005.

5. X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In ICDE, pages 458–469, 2013.

6. X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. PVLDB, 6(13):1498–1509, 2013.

7. A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In SIGMOD, pages 445–456, 2014.

8. G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In ICDE, pages 541–552, 2013.

9. X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In 29th IEEE International Conference on Data Engineering, pages 458–469, 2013.

10. F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The llunatic datacleaning framework. Proceedings of the VLDB Endowment, 6(9):625– 636, 2013.

11. M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. PVLDB, 4(5):279–289, 2011.

12. W. Fan. Dependencies revisited for improving data quality. In PODS, pages 159–170, 2008.

13. Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. VLDB J. 10, 4 (2001), 334–350.

14. Shaoxu Song and Lei Chen. 2013. Efficient discovery of similarity constraints for matching dependencies. Data Knowl. Eng. 87 (2013), 146–166.

15. Yihan Wang, Shaoxu Song, Lei Chen, Jeffrey Xu Yu, and Hong Cheng. 2017. Discovering conditional matching rules. ACM Trans. Knowl. Disc. Data 11, 4 (2017), 46.

16. Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. Comput. J. 42, 2 (1999), 100–111.

17. Peter A. Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. AI Commun. 12, 3 (1999), 139–160

18. Thorsten Papenbrock and Felix Naumann. 2016. A hybrid approach to functional dependency discovery. In Proceedings of the International Conference on Management of Data (SIGMOD'16). 821–833.

19. Thorsten Papenbrock and Felix Naumann. 2016. A hybrid approach to functional dependency discovery. In Proceedings of the International Conference on Management of Data (SIGMOD'16). 821–833.

20. Creating a Spell Checker with TensorFlow — Режим доступу до ресурсу: https://towardsdatascience.com/creating-a-spell-checker-with-tensorflow-d35b23939f60

21. Generate test data / Генерація адрес для тестових даних — Режим доступу до ресурсу:https://generatedata.com/

22. Geographic / Генерація вулиць України — Режим доступу до ресурсу: https://geographic.org/streetview/ukraine/kiev_city/kiev_city.html

ДОДАТОК А

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А. 1 Програмний код файлу SpellChecker

```
def load_data(path):
        """Load a data from its file"""
        input_file = os.path.join(path)
        with open(input_file) as f:
            data = f.read()
        return data
clean_datas = []
    for data in datas:
        clean_datas.append(clean_text(data))
vocab_to_int = {}
    count = 0
    for data in clean_datas:
        for character in data:
            if character not in vocab_to_int:
                vocab_to_int[character] = count
                count += 1
int_to_vocab = {}
    for character, value in vocab_to_int.items():
        int_to_vocab[value] = character
training_sorted = []
    testing_sorted = []
    for i in range(min_length, max_length+1):
        for adress in training:
            if len(adress) == i:
                training_sorted.append(adress)
        for adress in testing:
            if len(adress) == i:
                testing_sorted.append(adress)
for i in range(5):
        print(training_sorted[i], len(training_sorted[i]))

letters = ['а','б','в','г','ґ ','д','е','є','ж','з','и','і','ї',
'й','к','л','м','н','о','п','р','с','т','у','ф','х','ц','ш','щ',
'ь','ю','я']
def noise_maker(adress, threshold):
        noisy_adress = []
        i = 0
        while i < len(adress):
            random = np.random.uniform(0,1,1)
            # Most characters will be correct since the
threshold value is high
            if random < threshold:
```

```
                noisy_adress.append(adress[i])
            else:
                 __new___random = np.random.uniform(0,1,1)
                # ~33% chance characters will swap locations
                if new_random > 0.67:
                    if i == (len(adress) - 1):
                        # If last character in adress, it will
not be typed
                        continue
                    else:
                        # if any other character, swap order
with following character
                        noisy_adress.append(adress[i+1])
                        noisy_adress.append(adress[i])
                        i += 1
                # ~33% chance an extra lower case letter will be
added to the adress
                elif new_random < 0.33:
                    random_letter = np.random.choice(letters,
1)[0]
                    noisy_adress.append(vocab_to_int[random_lett
er])
                    noisy_adress.append(adress[i])
                # ~33% chance a character will not be typed
                else:
                    pass
            i += 1
        return noisy_adress

def model_inputs():
        '''Create palceholders for inputs to the model'''

        with tf.name_scope('inputs'):
            inputs = tf.placeholder(tf.int32, [None, None],
name='inputs')
        with tf.name_scope('targets'):
            targets = tf.placeholder(tf.int32, [None, None],
name='targets')
        keep_prob = tf.placeholder(tf.float32, name='keep_prob')
        inputs_length = tf.placeholder(tf.int32, (None,),
name='inputs_length')
        targets_length = tf.placeholder(tf.int32, (None,),
name='targets_length')
        max_target_length = tf.reduce_max(targets_length,
name='max_target_len')


        return inputs, targets, keep_prob, inputs_length,
targets_length, max_target_length
def process_encoding_input(targets, vocab_to_int, batch_size):
        '''Remove the last word id from each batch and concat
the <GO> to the begining of each batch'''
```

```
        with tf.name_scope("process_encoding"):
            ending = tf.strided_slice(targets, [0, 0],
[batch_size, -1], [1, 1])
            dec_input = tf.concat([tf.fill([batch_size, 1],
vocab_to_int['<GO>']), ending], 1)

def encoding_layer(rnn_size, sequence_length, num_layers,
rnn_inputs, keep_prob, direction):
    '''Create the encoding layer'''

    if direction == 1:
        with tf.name_scope("RNN_Encoder_Cell_1D"):
            for layer in range(num_layers):
                with
tf.variable_scope('encoder_{}'.format(layer)):
                    lstm = tf.contrib.rnn.LSTMCell(rnn_size)
                    drop =
tf.contrib.rnn.DropoutWrapper(lstm,
                            input_keep_prob=keep_prob)
                    enc_output, enc_state =
tf.nn.dynamic_rnn(drop,
rnn_inputs,
sequence_length,
                                                        dt
ype=tf.float32)
            return enc_output, enc_state

    if direction == 2:
        with tf.name_scope("RNN_Encoder_Cell_2D"):
            for layer in range(num_layers):
                with
tf.variable_scope('encoder_{}'.format(layer)):
                    cell_fw = tf.contrib.rnn.LSTMCell(rnn_size)
                    cell_fw =
tf.contrib.rnn.DropoutWrapper(cell_fw,
t_keep_prob=keep_prob)
                    cell_bw = tf.contrib.rnn.LSTMCell(rnn_size)
                    cell_bw =
tf.contrib.rnn.DropoutWrapper(cell_bw,
                                                      inpu
t_keep_prob=keep_prob)

                    enc_output, enc_state =
tf.nn.bidirectional_dynamic_rnn(cell_fw,
            cell_bw,
            rnn_inputs,
          sequence_length,
            dtype=tf.float32)
            # Join outputs since we are using a bidirectional
RNN
            enc_output = tf.concat(enc_output, 2)
            # Use only the forward state because the model can't
use both states at once
```

```
            return enc_output, enc_state[0]

def training_decoding_layer(dec_embed_input, targets_length,
dec_cell, initial_state, output_layer,
                            vocab_size, max_target_length):
    '''Create the training logits'''

    with tf.name_scope("Training_Decoder"):
        training_helper =
tf.contrib.seq2seq.TrainingHelper(inputs=dec_embed_input,
            sequence_length=targets_length,
            time_major=False)

        training_decoder =
tf.contrib.seq2seq.BasicDecoder(dec_cell,
         training_helper,
         initial_state,
         output_layer)

        training_logits, _ =
tf.contrib.seq2seq.dynamic_decode(training_decoder,
output_time_major=False,
impute_finished=True,
maximum_iterations=max_target_length)
        return training_logits

def seq2seq_model(inputs, targets, keep_prob, inputs_length,
targets_length, max_target_length, vocab_size, rnn_size,
num_layers, vocab_to_int, batch_size, embedding_size,
direction):
    '''Use the previous functions to create the training and
inference logits'''

    enc_embeddings = tf.Variable(tf.random_uniform(
        [vocab_size, embedding_size], -1, 1))
    enc_embed_input = tf.nn.embedding_lookup(enc_embeddings,
inputs)
    enc_output, enc_state = encoding_layer(rnn_size,
inputs_length, num_layers,
                                           enc_embed_input,
keep_prob, direction)

    dec_embeddings = tf.Variable(tf.random_uniform(
        [vocab_size, embedding_size], -1, 1))
    dec_input = process_encoding_input(targets, vocab_to_int,
batch_size)
    dec_embed_input = tf.nn.embedding_lookup(dec_embeddings,
dec_input)

    training_logits, inference_logits =
decoding_layer(dec_embed_input, dec_embeddings, enc_output,
enc_state, vocab_size,inputs_length,  max_target_length,
```

```
rnn_size, vocab_to_int, keep_prob, batch_size, num_layers,
direction)
    return training_logits, inference_logits

def get_batches(adresss, batch_size, threshold):
    """Batch adresss, noisy adresss, and the lengths of their
adresss together.With each epoch, adresss will receive new
mistakes"""

    for batch_i in range(0, len(adresss)//batch_size):
        start_i = batch_i * batch_size
        adresss_batch = adresss[start_i:start_i + batch_size]

        adresss_batch_noisy = []
        for adress in adresss_batch:
            adresss_batch_noisy.append(noise_maker(adress,
threshold))

        adresss_batch_eos = []
        for adress in adresss_batch:
            adress.append(vocab_to_int['<EOS>'])
            adresss_batch_eos.append(adress)

        pad_adresss_batch =
np.array(pad_adress_batch(adresss_batch_eos))
        pad_adresss_noisy_batch = np.array(
            pad_adress_batch(adresss_batch_noisy))

        # Need the lengths for the _lengths parameters
        pad_adresss_lengths = []
        for adress in pad_adresss_batch:
            pad_adresss_lengths.append(len(adress))

        pad_adresss_noisy_lengths = []
        for adress in pad_adresss_noisy_batch:
            pad_adresss_noisy_lengths.append(len(adress))

        yield pad_adresss_noisy_batch, pad_adresss_batch,
pad_adresss_noisy_lengths, pad_adresss_lengths

def train(model, epochs, log_string):
    '''Train the RNN'''

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        # Used to determine when to stop the training early
        testing_loss_summary = []

        # Keep track of which batch iteration is being trained
        iteration = 0
```

```
        display_step = 30  # The progress of the training will
be displayed after every 30 batches
        stop_early = 0
        stop = 3  # If the batch_loss_testing does not decrease
in 3 consecutive checks, stop training
        per_epoch = 3  # Test the model 3 times per epoch
        testing_check =
(len(training_sorted)//batch_size//per_epoch)-1

        print()
        print("Training Model: {}".format(log_string))

        train_writer = tf.summary.FileWriter(
            './logs/1/train/{}'.format(log_string), sess.graph)
        test_writer = tf.summary.FileWriter(
            './logs/1/test/{}'.format(log_string))

        for epoch_i in range(1, epochs+1):
            batch_loss = 0
            batch_time = 0

            for batch_i, (input_batch, target_batch,
input_length, target_length) in enumerate(
                    get_batches(training_sorted, batch_size,
threshold)):
start_time = time.time()
summary, loss, _ = sess.run([model.merged, model.cost,
model.train_op], {model.inputs: input_batch, model.targets:
target_batch, model.inputs_length: input_length,
model.targets_length: target_length, model.keep_prob:
keep_probability})
batch_loss += loss
end_time = time.time()
                batch_time += end_time - start_time

                # Record the progress of training
                train_writer.add_summary(summary, iteration)

                iteration += 1

                if batch_i % display_step == 0 and batch_i > 0:
                    print('Epoch {:>3}/{} Batch {:>4}/{} - Loss:
{:>6.3f}, Seconds: {:>4.2f}'
                          .format(epoch_i,
                                  epochs,
                                  batch_i,
                                  len(training_sorted) //
batch_size,
                                  batch_loss / display_step,
                                  batch_time))
                    batch_loss = 0
                    batch_time = 0
```

```
if batch_i % testing_check == 0 and batch_i > 0:
batch_loss_testing = 0
batch_time_testing = 0
for batch_i, (input_batch, target_batch, input_length,
target_length) in enumerate(
        get_batches(testing_sorted, batch_size, threshold)):
    start_time_testing = time.time()
    summary, loss = sess.run([model.merged,
                              model.cost],
                              {model.inputs: input_batch,
                              model.targets: target_batch,
                              model.inputs_length:
input_length,
                              model.targets_length:
target_length,
                              model.keep_prob: 1})

    batch_loss_testing += loss
    end_time_testing = time.time()
    batch_time_testing += end_time_testing - start_time_testing

    # Record the progress of testing
    test_writer.add_summary(summary, iteration)

n_batches_testing = batch_i + 1
print('Testing Loss: {:>6.3f}, Seconds: {:>4.2f}'
        .format(batch_loss_testing / n_batches_testing,
                batch_time_testing))

batch_time_testing = 0

# If the batch_loss_testing is at a new minimum, save the model
testing_loss_summary.append(batch_loss_testing)
if batch_loss_testing <= min(testing_loss_summary):
    print('New Record!')
    stop_early = 0
    checkpoint = "./{}.ckpt".format(log_string)
    saver = tf.train.Saver()
    saver.save(sess, checkpoint)

else:
    print("No Improvement.")
    stop_early += 1
    if stop_early == stop:
        break

if stop_early == stop:
print("Stopping Training.")
break
```

A. 2 – Програмний код файлу HybridMD

```
 def class HyMD :
MatchingDependencyAlgorithm,RelationalInputParameterAlgorithm,
StringParameterAlgorithm, IntegerParameterAlgorithm {

  static {
    CPSTypeIdResolver.addClassLoader(HyMD.class.getClassLoader()
)
  }

  def MatchingDependencyResultReceiver resultReceiver
  def Relation leftRelation
  def Relation rightRelation
  def double minThreshold = 0.7
  def SupportCalculator supportCalculator = new
SizeBasedSupportCalculator()
  def MappingConfiguration mappingConfiguration = new
MappingConfiguration()

  def static ConfigurationRequirement<?>
createMinThresholdConfigurationRequirement() {
    ConfigurationRequirementString config = new
ConfigurationRequirementString(
        Identifier.THRESHOLD.name())
    config.setRequired(false)
    config.setDefaultValues(new String[]{Double.toString(0.7)})
    return config
  }

  def static ConfigurationRequirement<?>
createRelationConfigurationRequirement() {
    ConfigurationRequirementRelationalInput config = new
ConfigurationRequirementRelationalInput(
        Identifier.RELATION.name())
    config.setRequired(true)
    return config
  }

  def static ConfigurationRequirement<?>
createMappingConfigurationRequirement() {
    ConfigurationRequirementString config = new
ConfigurationRequirementString(
        Identifier.CONFIG.name())
    config.setRequired(false)
    config.setDefaultValues(new String[]{"{}"})
    return config
  }

  def static ConfigurationRequirement<?>
createSupportConfigurationRequirement() {
    ConfigurationRequirementInteger config = new
ConfigurationRequirementInteger(
```

```
        Identifier.SUPPORT.name())
    config.setRequired(false)
    config.setDefaultValues(new Integer[]{Integer.valueOf(0)})
    return config
  }


   def ArrayList<ConfigurationRequirement<?>>
getConfigurationRequirements() {
    ArrayList<ConfigurationRequirement<?>> configs = new
ArrayList<>()

    configs.add(createMinThresholdConfigurationRequirement())

    configs.add(createRelationConfigurationRequirement())

    configs.add(createMappingConfigurationRequirement())

    configs.add(createSupportConfigurationRequirement())

    return configs
  }


   def execute() throws AlgorithmExecutionException {
    if (leftRelation == null) {
      throw new AlgorithmConfigurationException("At least one
relation needed")
    }
    if (rightRelation == null) {
      MDMapping mappings =
mappingConfiguration.createMapping(leftRelation)
      Discoverer discoverer = createDiscoverer(mappings)
      discoverer.discover(leftRelation)
    } else {
      MDMapping mappings =
mappingConfiguration.createMapping(leftRelation, rightRelation)
      Discoverer discoverer = createDiscoverer(mappings)
      discoverer.discover(leftRelation, rightRelation)
    }
  }


   def String getAuthors() {
    return "Philipp Schirmer"
  }


   def String getDescription() {
    return "Discover Matching Dependencies"
  }
```

```
   def void setRelationalInputConfigurationValue(String
identifier,
      RelationalInputGenerator... values) throws
AlgorithmConfigurationException {
    if (Identifier.RELATION.name().equals(identifier)) {
      if (values.length == 0 || values.length > 2) {
        throw new AlgorithmConfigurationException(
            "MD Discovery expects one or two relational inputs")
      }
      this.leftRelation = wrap(values[0])
      if (values.length == 2) {
        this.rightRelation = wrap(values[1])
      } else {
        this.rightRelation = null
      }
    }
  }


   def void setResultReceiver(MatchingDependencyResultReceiver
resultReceiver) {
    this.resultReceiver = resultReceiver
  }


   def void setStringConfigurationValue(String identifier,
String... values)
      throws AlgorithmConfigurationException {
    if (Identifier.THRESHOLD.name().equals(identifier)) {
      if (values.length > 0) {
        String value = values[0]
        try {
          this.minThreshold = Double.parseDouble(value)
        } catch (NumberFormatException e) {
          throw new AlgorithmConfigurationException("Min
threshold is not valid", e)
        }
      } else {
        throw new AlgorithmConfigurationException("Min threshold
not provided")
      }
    }
    if (Identifier.CONFIG.name().equals(identifier)) {
      if (values.length > 0) {
        String json = values[0]
        this.mappingConfiguration = readConfig(json)
      } else {
        throw new AlgorithmConfigurationException("Config file
not provided")
      }
    }
  }
```

```
  def Discoverer buildDiscoverer(MDMapping mappings) {
    DiscoveryConfiguration configuration = createConfiguration()
    return newBuilder()
        .mappings(mappings)
        .configuration(configuration)
        .build()
  }

  def DiscoveryConfiguration createConfiguration() {
    DiscoveryConfiguration configuration = new
DiscoveryConfiguration()
    configuration.setMinThreshold(minThreshold)
    configuration.setSupportCalculator(supportCalculator)
    return configuration
  }

  def Discoverer createDiscoverer(MDMapping mappings) {
    Discoverer discoverer = buildDiscoverer(mappings)
    Optional.ofNullable(resultReceiver)
        .map(MetanomeResultListener:: __new__)
        .ifPresent(discoverer::register)
    return discoverer
  }

  def HybridDiscovererBuilder  __new__Builder() {
    return HybridDiscoverer.builder()
        .store(false)
        .parallel(true)
  }

  def Relation wrap(RelationalInputGenerator input) {
    return  __new__ MetanomeRelation(input)
  }

  def MappingConfiguration readConfig(String json) throws
AlgorithmConfigurationException {
    try {
      ObjectReader reader =
Jackson.createReader(MappingConfiguration.class)
      return reader.readValue(json)
    } catch (IOException e) {
      throw  __new__ AlgorithmConfigurationException("Error
parsing json config", e)
    }
  }


   def void setIntegerConfigurationValue(String identifier,
Integer... values)
      throws AlgorithmConfigurationException {
    if (Identifier.SUPPORT.name().equals(identifier)) {
      if (values.length > 0) {
        int support = values[0].intValue()
```

```
        this.supportCalculator =
createSupportCalculator(support)
      } else {
        throw  __new__ AlgorithmConfigurationException("Support
not provided")
      }
    }
  }

  def SupportCalculator createSupportCalculator(int support) {
    SizeBasedSupportCalculator supportCalculator =  __new__
SizeBasedSupportCalculator()
    supportCalculator.setNonReflexiveMatches(support)
    return supportCalculator
  }
```

ДОДОАТОК Б

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

УДК 004.021

Біловол А. І.

Хмельницький національний університет

УДОСКОНАЛЕННЯ МЕТОДУ ТА ЗАСОБІВ ОЧИЩЕННЯ ДАНИХ НА ОСНОВІ MATCHING DEPENDENCY TECHNIQUE

У роботі наведено результати дослідження проблеми очищення даних. Пропонується використовувати техніку машинного навчання, яка допомагає давати рекомендації щодо виправлення неправильного значення. Запропонована методика та її покращення ілюструє, як ми можемо уникнути помилок в поштових адресах, введених будь-яким користувачем системи.

The paper presents the results of a study of the problem of data cleaning. It is suggested to use machine learning techniques, which help to give recommendations for correcting incorrect values. The proposed method and its improvement illustrates how we can avoid errors in email addresses entered by any user of the system.

Люди стали свідками значного зменшення повсякденних фізичних зусиль за допомогою комп'ютерів. Іншими словами, ми з кожним днем стаємо все більш залежними від машин. У попередню епоху в школі, державному секторі, приватному секторі чи лікарні всі дані/записи раніше були фізичними або на паперах/файлах. Було б цілком природно помилятися, зберігаючи ці дані у файлі або папері. У сучасному світі всі рукописні та фізичні файли або перетворилися, або щодня перетворюються на м'які копії. Не було б несподіванкою очікувати помилки у результатах після того, як люди ввели дані у систему.

Найпоширеніші введені значення будь-якою людиною в наш час - це їхні імена, номери телефонів та адреси під час реєстрації на будь-якому веб-сайті, де ім'я людини не можна стандартизувати.

Можна розглянути приклад з найбільш розповсюджених проблем, одна і та ж назва може бути написана кількома способами відповідно до звуку. Наприклад, "вулиця Хрещатик" - це назва вулиці у Києві, яка може бути написана як "вулиця Хрищатик" або "вулиця Крищатик", оскільки вона буде звучати подібно до оригінальної вулиці.

Також прикладом є людина яка замовляє щось в Інтернеті, вказавши неправильний поштовий індекс у введеній адресі, ця помилка може призвести до затримки доставки товару або ще гірше, що товар може залишитися недоставленим.

Іншою проблемою, є те, що на сьогоднішній день більшість систем очищення даних створені лише для англійської мови. Розробка системи очищення

даних українською мовою потребує великих масивів українських даних, доцільно використати для цього доступні відкриті дані.

Методика відповідності залежності є дуже перспективною для процесу очищення даних, на основі робіт з машинного навчання та методів вивчення очищення даних, оскільки результати були більш точними та ефективними [1] [2].

Метою завдання є виявлення та виправлення помилкових записів у структурованому наборі даних та розробка технології очищення даних на основі відповідності залежностей.

Областю застосування було обрано адреси, які будь-який користувач, що проживає в будь-якій країні, буде вводити в Інтернеті. Тут дані зовнішніх джерел - це авторизовані та надійні дані, які використовувалися для перевірки та перевірки вхідних даних користувача. Дані введені користувачем, відповідно до назви - це дані, введені будь-яким користувачем. Було взято до уваги назву вулиці, номер вулиці, місто, область, країну та поштовий індекс для процесу очищення даних.

Відповідно до назви, метод відповідності залежності вимагає джерела даних, яке може бути використано як джерело з автентифікацією. Ці автентифіковані дані допомагають перевірити правильність введення, зіставляючи їх разом для процесу очищення. Метод бере в якості вхідного даних брудний набір даних. Щоб краще зрозуміти це, ми маємо зовнішнє автентифіковане джерело даних адрес для Кам'янець-Подільського, Хмельницької області, як показано у таблиці 1 нижче.

Таблиця 1 – Зовнішня автентифікована інформація, адрес в Кам'янець-Подільському

Автентифіковане джерело			
Адреса	Місто	Область	Індекс
проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
провулок Амбулаторний, 12	Кам'янець-Подільський	Хмельницька	32341
вулиця Заводська 3	Кам'янець-Подільський	Хмельницька	32307
вулиця Вишнева 8	Кам'янець-Подільський	Хмельницька	32302

Набори даних, введені користувачем, наведені у таблиці 2 й містять погані вхідні дані, які можна чітко побачити, порівнявши ці набори даних із зовнішнім джерелом даних у таблиці 1. Поштовий індекс у рядках 2 та 3 здається неправильним, назва підприємства та місто у рядку 4 також здаються неправильними. Першим кроком у робочому процесі методу є виявлення клітин у наборі даних користувача з потенційно неточними значеннями. Цей процес поділяє кортежі даних на шумні(забруднені) та чисті клітини. У нас є певні залежності, які

можуть бути корисними в процесі очищення даних користувача за допомогою зовнішніх даних. Ці залежності показані на рисунку 1.

Таблиця 2 – Набір забруднених даних введений користувачем

Найменування фірми	Адреса	Місто	Область	Індекс
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Avto	проспект Грушевського, 9	**Кам'янець-Подольський**	Хмельницька	32301

залежність1: Індекс = Авт.Індекс → Місто = Авт.Місту
залежність2: Індекс = Авт.Індекс → Область = Авт.Область
залежність3: Місто= Авт.Місто ∧ Область = Авт.Область ∧ Адреса = = Авт.Адреса → Індекс = Авт.Індекс

Рисунок 1 – Відповідність залежностей

Тут відповідність залежностей - це можливі сигнали, які допомагають виявляти та виправляти погані вхідні дані, порівнюючи їх із зовнішнім джерелом даних.

Відновленні та очищенні набори вихідних даних, які ми отримуємо за допомогою зовнішніх даних та їх можливих залежностей, мають правильне місто у рядку 4 та правильний поштовий індекс у рядках 2 та 3.

Проведенні дослідження показали[3], що на сьогоднішній день є вкрай важливим процес використання методів очищення даних, та існує простір для покращення ефективності, усунення недоліків систем очищення. Одним з можливих рішень для такої оптимізації є використання та вдосконалення методики відповідності залежностей:

– збільшення набору даних, які в свою чергу дають більш ефективні результати;

– інший алгоритм пошуку, такий як хеш-таблиці, замість двійкового пошуку, щоб зробити цю техніку ефективнішою;

– застосування більш розвиненої техніки машинного навчання [4]. Наприклад, якщо назва вулиці – "проспект Миру", і користувач ввів назву вулиці як "пр. Миру", то вдосконалена техніка повинна вважати її хорошими вхідними

даними та використовувати її як дійсні дані для перевірки інших записів для тієї ж адресу.

Отже в роботі досліджено існуючі проблеми очищення даних. В результаті виявлено, що техніка очищення даних на основі залежностей є корисною у складанні рекомендацій, щодо виправлення неправильного значення при введені даних.

Перелік посилань

1. Xu Chu, Ihab F. Ilyas, Sanjay Krishnan and Jiannan Wang, "Data Cleaning: Overview and Emerging Challenges", In Proceedings of the 2016 ACM SIGMOD Conference on Management of Data, San Francisco, USA, pp. 1-3
2. Theodoros Rekatsinas, Xu Chu and Christopher Ré, "HoloClean: Holistic Data repairs with Probabilistic Inference", Proceedings of the VLDB Endowment, Volume 10, No. 11, pp. 1190-1191, August 2017
3. Shaoxu Song, Lei Che, «Discovering Matching Dependencies», Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009
4. Spell Checker with TensorFlow. [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/creating-a-spell-checker-with-tensorflow-d35b23939f60

ДОДОАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique

Науковий керівник:
Бармак Олександр Володимирович,, д.т.н., проф
Виконавець:
Біловол Андрій Ігорович
Кафедра:
Інженерія програмного забезпечення

Актуальність роботи

- Очищення даних є однією з найважливіших проблем в управлінні даними, оскільки брудні дані часто призводять до неточних результатів аналізу даних і неправильних бізнес-рішень.
- За останні кілька років спостерігається сплеск інтересу як промисловості, так і наукових кіл до проблем очищення даних, включаючи нові абстракції, інтерфейси, підходи до масштабованості та статистичні методи.

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА ДОСЛІДЖЕННЯ

Об'єкт дослідження – виявлення та відновлення брудних даних в процесі очищення даних.

Предмет дослідження – метод очищення даних на основі техніки відповідності залежності.

Мета дослідження – виявлення та виправлення помилкових записів у структурованому наборі даних та вдосконалення методу очищення даних на основі відповідності залежностей та створення засобу, необхідного для застосування та реалізації.

Аналіз існуючих рішень

В якості обраної техніки було вирішено вибрати нещодавно запропоновані відповідні залежності (MD) для різних додатків якості даних, таких як виявлення порушення обмежень цілісності та ідентифікація повторюваних об'єктів.

Методика відповідності залежності є дуже перспективною для процесу очищення даних, на основі дослідження робіт з машинного навчання та вивчення методів очищення даних, оскільки результати були більш точними та ефективними.

Matching Dependency Technique

Зовнішня автентифікована інформація,адрес в Кам'янець-Подільському

Автентифіковане джерело			
Адреса	Місто	Область	Індекс
проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
провулок Амбулаторний, 12	Кам'янець-Подільський	Хмельницька	32341
вулиця Заводська 3	Кам'янець-Подільський	Хмельницька	32307
вулиця Вишнева 8	Кам'янець-Подільський	Хмельницька	32302

Набір забруднених даних введений користувачем

Найменування фірми	**Адреса**	**Місто**	Область	Індекс
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	32301
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Automobile	проспект Грушевського, 9	Кам'янець-Подільський	Хмельницька	**32302**
Fortetsya Avto	проспект Грушевського, 9	**Кам'янець-Подольський**	Хмельницька	32301

Matching Dependency Technique

Відповідні залежності

залежність1: Індекс = Авт.Індекс → Місто = Авт.Місту
залежність2: Індекс = Авт.Індекс → Область = Авт.Область
залежність3: Місто= Авт.Місто ∧ Область = Авт.Область ∧ Адреса = = Авт.Адреса → Індекс = Авт.Індекс

Тут відповідні залежності - це можливі сигнали, які допомагають виявляти та виправляти погані вхідні дані, порівнюючи їх із зовнішнім джерелом даних.

Удосконалення методу очищення даних на основі техніки відповідності залежностей

Вдосконалення методу відповідності залежностей за рахунок:

Застосування техніки машинного навчання для попередньої перевірки вхідних даних. Наприклад, якщо назва вулиці - «вул Хрещатик", і користувач ввів назву вулиці як «вул Крещатик", то техніка повинна вважати її хорошими вхідними даними та використовувати її як дійсні дані для перевірки інших записів для того ж адресу.

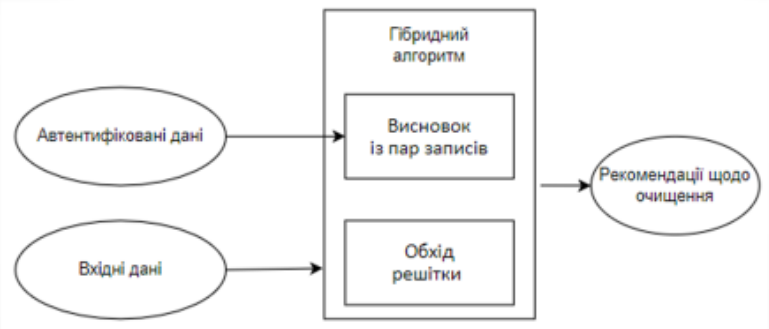
```
1 - ['вул. Соборн','Запоріжя','Запорізька область','Україна','69000']
2 - ['вул. Соборна','Запоріжя','Запорізька область','Україна','69000']

1 - ['вул. Радісна','Львів','Львівка область','Україна','79000']
2 - ['вул. Радісна','Львів','Львівська область','Україна','79000']
```

1 – це вхідні дані
2 – дані з перевіреною орфографією

Удосконалення методу очищення даних на основі техніки відповідності залежностей

Вдосконалення методу відповідності залежностей за рахунок:

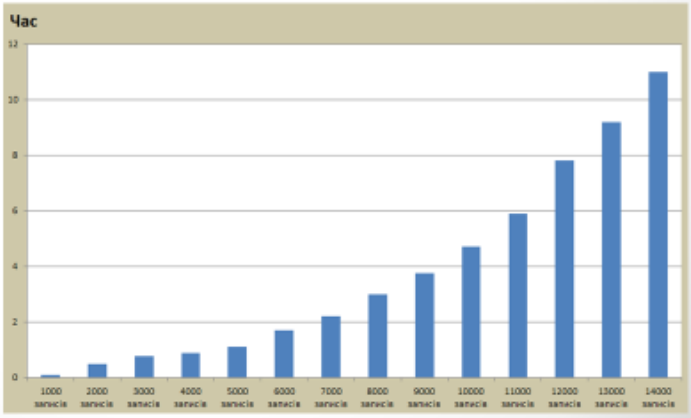
Розглянувши дві запропоновані стратегії пошуку, їх переваги та недоліки, а також їх структури даних, алгоритм пошуку залежностей можна вдосконалити поєднавши їх у гібридний алгоритм.



Оцінка та порівняльний аналіз

Розмір зовнішніх даних	Розмір тестових даних	Кількість помилок у тестових даних	Кількість рекомендацій	Кількість помилкових спрацьовувань	Кількість помилкових негативних результатів	Витрачений час (с.)
1000	100	27	27	0	0	0,2

Приклад оцінки тестових даних для перевірки реалізованої техніки



Оцінка та порівняльний аналіз

Тип алгоритму пошуку	Розмір зовнішніх даних	Розмір тестових даних	Кількість помилок у тестових даних	Витрачений час (с.)
Гібридний алгоритм	1000	150	87	0,89
Обхід решітки	1000	150	87	1,1
Висновок з пар записів	1000	150	87	3
Гібридний алгоритм	15000	1500	389	13
Обхід решітки	15000	1500	389	16,2
Висновок з пар записів	15000	1500	389	42,2

Порівняння запропонованого алгоритму пошку залежностей з базовими методами пошуку

Наукова новизна

- Удосконалено метод очищення даних на основі техніки відповідних залежностей за рахунок гібридного алгоритму пошуку залежностей, що дозволило зробити цю техніку ефективнішою.

- Удосконалено техніку відповідних залежностей за рахунок застосування техніки машинного навчання, що дозволило використовувати погані дані, як дійсні для перевірки інших записів для того ж набору.

Практичне значення

Практична цінність отриманих результатів полягає в успішній реалізації методу та програмного засобу для очищення даних. За рахунок втілених вдосконалень, програмний засіб є ефективнішим в порівнянні з класичними рішеннями на основі техніки відповідних залежностей.

Наукові публікації

Опубліковано тези доповіді у збірнику матеріалів Всеукраїнської науково-практичної конференції:

Біловол А.І. Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique // Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». Хмельницький – 2021. – 287-290с.

Висновки

- Було здійснено деталізований аналіз літературних джерел та класифікації методів очищення даних, щоб визначити стан проблеми дипломної роботи. Техніка відповідності залежностей здавалася багатообіцяючою для процесу очищення даних.

- Було розглянуто, реалізовано та покращено техніку відповідності залежностей, використовуючи мову Python, самостійно створені набори даних, дані, введені користувачем, та дані, що перевіряються.

Висновки

- Вдосконалена метод на основі відповідних залежностей може бути одним із корисних методів очищення будь-якого набору даних, у якому є погані вхідні дані.
- Вдосконалений метод здатний ефективніше обробляти великі набори даних, оскільки використовується гібридний пошук, що покращує ефективність процесу пошуку.
- Процес очищення набору даних став легшим, оскільки впроваджена методика швидше виявляє нечисті набори даних, а потім дає відповідні можливі рекомендації щодо їх очищення(виправлення).

Дякую за увагу!

Завідувачу кафедри інженерії програмного
забезпечення проф. Бедратюку Л. П.
здобувача вищої освіти
Біловола Андрія Ігоровича
факультет ІТ, 2 курс, група ІПЗм-20-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

23.11.2021
дата

підпис

Mon Dec 13 15:56:56 EET 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 8.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 6%**

ID: 99075 Назва: Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique Додано в БД: 2021-12-13 Автора: А.І. Біловол Керівники: О.В. Бармак Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	113171	946	14183 (13%)	126 (13%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1009666057

Дата перевірки:
13.12.2021 16:47:20 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.12.2021 16:49:18 EET

ID користувача:
100005589

Назва документа: **Диплом_Біловол_без_додатків**

Кількість сторінок: **97** Кількість слів: **17865** Кількість символів: **128948** Розмір файлу: **980.52 KB** ID файлу: **1009665480**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.5%
Схожість

Найбільша схожість: **2.88%** з джерелом з Бібліотеки (ID файлу: **1009513555**)

4.35% Джерела з Інтернету	193	Сторінка 99
4.79% Джерела з Бібліотеки	107	Сторінка 100

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	63

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique»
Автор: Біловол Андрій Ігорович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Бармак Олександр Володимирович, доктор технічних наук, професор
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості так як є елементами формул.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 8.5% і адресується до 193 джерел з Інтернет та 107 джерел з бібліотеки, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь дипломної роботи.

Керівник _____ О. В. Бармак

Гарант ОП _____ О. М. Яшина

Завідувач кафедри _____ Л. П. Бедратюк

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник: Біловол Андрій Ігорович

Тема: Удосконалення методу та засобів очищення даних на основі Matching Dependency Technique

Спеціальність: 121 «Інженерія програмного забезпечення»

Обсяг дипломної роботи:

Кількість листів креслень ___; кількість сторінок записки 81

1. Короткий зміст роботи та прийнятих рішень В дипломній роботі удосконалено метод очищення даних на основі методу відповідності залежностей з використанням машинного навчання, який надає можливість виявлення та виправлення помилкових записів у структурованому наборі даних. Удосконалений метод за рахунок запропонованого гібридного алгоритму надає можливість підвищити ефективність виявлення та виправлення помилкових записів в порівнянні з відомими алгоритмами. На основі запропонованого методу реалізовано відповідний програмний засіб.

2. Висновок про відповідність роботи дипломному завданню Дипломна робота в цілому відповідає виданому завданню в теоретичній та практичній частинах.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Розділ 1 – Проведено аналіз методів виявлення аномалій та помилок, методів та алгоритмів відновлення даних та методів очищення даних. Розділ 2 – Представлено основи удосконаленого методу очищення даних, а саме алгоритми виявлення залежностей та відновлення даних. Запропоновано гібридний алгоритм пошуку залежностей, який враховує недоліки відомих підходів. Залучено методи машинного навчання для підвищення ефективності удосконаленого методу за рахунок покращення алгоритму перевірки та виправлення вхідних даних з метою їх очищення. Розділ 3 – Представлено програмну реалізацію, яка використовує удосконелений метод. Розділ 4 – Представлено результати, що демонструють функціональність розробленого програмного засобу. Застосування розробленого програмного засобу

надає можливість підвищити ефективність виявлення та виправлення помилкових записів в структурованих наборах даних в порівнянні з відомими алгоритмами. В загальному розділи відповідають завданню.

4. Позитивні сторони роботи: Застосування удосконаленого методу очищення даних на основі підходу відповідності залежностей надає можливість підвищити ефективність виявлення та виправлення помилкових записів в структурованих даних в порівнянні з відомими алгоритмами.

5. Негативні сторони роботи: Наукова новизна роботи виписана недостатньо чітко. Стиль викладення матеріалу та наявність граматичних помилок ускладнюють його розуміння. Фрагменти коду програмної реалізації методу повинні бути перенесені з основного тексту пояснювальної записки в додатки. В переліку джерел посилань використано надмірну кількість застарілих джерел.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Оформлення пояснювальної записки виконано на задовільному рівні.

7. Відгук про роботу в цілому В загальному дипломна робота заслуговує задовільної оцінки. Дипломна робота присвячена вирішенню задачі підвищення ефективності виявлення та виправлення помилкових записів в структурованих наборах даних. _____

8. Інші зауваження: _____

_____

_____

_____

9. Оцінка дипломної роботи: Розглянувши позитивні та негативні сторони представленої дипломної роботи, можна зробити висновок, що вона заслуговує на оцінку «задовільно». _____

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) к.т.н., доцент кафедри комп'ютерної інженерії та інформаційних систем Бобровнікова К.Ю.

" 13 " грудня 2021 р. _____ (підпис)