НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

**КУРСОВА РОБОТА**

з дисципліни «Аналіз даних в інформаційних системах»

на тему: «Класифікація і кластеризація книг»

Студентки 3 курсу ІП-з11 групи

Спеціальності: 121

«Інженерія програмного забезпечення»

Трущак Ольги Ігорівни

«ПРИЙНЯВ» з оцінкою

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доц. Ліхоузова Т.А. / доц. Олійник Ю.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Підпис                    Дата

Київ - 2023 рік

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Кафедра інформатики та програмної інженерії

Дисципліна Аналіз даних в інформаційно-управляючих системах

Спеціальність 121 "Інженерія програмного забезпечення"

Курс 2 Група ІП-з11 Семестр 4

**ЗАВДАННЯ**

**на курсову роботу студентки**

|  |
| --- |
| Трущак Ольги Ігорівни |

|  |  |
| --- | --- |
| 1.Тема роботи | Класифікація та кластеризація книг |
|  | |
|  | |

|  |  |
| --- | --- |
| 2.Строк здачі студентом закінченої роботи | 30.11.2023 |

|  |  |
| --- | --- |
| 3. Вхідні дані до роботи | методичні вказівки до курсової робота, обрані дані з сайтів |
| <https://data.seattle.gov/Community/Library-Collection-Inventory/6vkj-f5xf> , | |
| <https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m> | |
|  | |
|  | |
|  | |

4.Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
| 1. Постановка задачі |
| 2. Аналіз предметної області |
| 3. Очищення даних |
| 4. Інтелектуальний аналіз даних |

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

|  |  |
| --- | --- |
|  | |
|  | |
|  | |
|  | |
| 6.Дата видачі завдання | 17.05.2023 |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 16.04.2022 |  |
| 2. | Визначення зовнішніх джерел даних | … |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи |  |  |
| 4. | Розробка моделі сховища даних |  |  |
| 4. | Розробка ETL процесів |  |  |
| 5. | Обґрунтування методів інтелектуального аналізу даних |  |  |
| 6. | Застосування та порівняння ефективності методів інтелектуального аналізу даних |  |  |
| 7. | Підготовка пояснювальної записки |  |  |
| 8. | Здача курсової роботи на перевірку |  |  |
| 9. | Захист курсової роботи | 19.06.2022 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Трущак Ольга Ігорівна |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | доц. Ліхоузова Т.А |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |
| Керівник |  |  | доц. Олійник Ю.О. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"26" червня 2022 р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової роботи: 50 сторінок, 22 рисунки, 1 таблиця, 5 посилань.

Об’єкт дослідження: інтелектуальний аналіз даних.

Предмет дослідження: створення програмного забезпечення, що проводить аналіз даних з подальшою кластеризацією та класифікацією і графічним відображенням результатів.

Мета роботи: проектування та реалізація програмного забезпечення для отримання даних зі сховища та їх подальшого аналізу та кластеризацією і класифікацією.

Дана курсова робота включає в себе: опис створення програмного забезпечення для очищення та інтелектуального аналізу даних, їх графічного відображення та кластеризації і класифікації за допомогою різних моделей.

**ЗМІСТ**

|  |  |
| --- | --- |
| ВСТУП | 7 |
| 1. ПОСТАНОВКА ЗАДАЧІ | 8 |
| 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ | 9 |
| 3. ОЧИЩЕННЯ ДАНИХ | 12 |
| 3.1 ОПИС ДЖЕРЕЛА ДАНИХ ТА ОТРИМАННЯ ВИБІРКИ | 12 |
| 3.2 ПІДГОТОВЧИЙ АНАЛІЗ ВХІДНИХ ДАНИХ | 14 |
| 3.3 ISBN | 15 |
| 3.4 РІК ПУБЛІКАЦІЇ | 17 |
| 3.5 ВИДАВЕЦЬ | 17 |
| 3.6 АВТОР | 24 |
| 3.7 ЗАГОЛОВОК І ПІДЗАГОЛОВОК | 28 |
| 3.8 ОБРОБКА ТЕМ КНИГИ | 29 |
| 3.9 ІНШІ ВХІДНІ ДАНІ | 30 |
| 4. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ | 31 |
| 4.1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ | 31 |
| 4.2 K-MEANS КЛАСТЕРИЗАЦІЯ | 34 |
| 4.3 AGGLOMERATIVE КЛАСТЕРИЗАЦІЯ | 37 |
| 4.4 КЛАСИФІКАЦІЯ RANDOM FOREST НА ДОПОВНЕНИХ ДАНИХ | 40 |
| 4.5 КЛАСИФІКАЦІЯ ЗА ДОПОМОГОЮ SVC НА ДОПОВНЕНИХ ДАНИХ | 42 |
| 4.6 КЛАСИФІКАЦІЯ З GRADIENT BOOSTING НА ДОПОВНЕНИХ ДАНИХ | 44 |
| 4.7 КЛАСИФІКАЦІЯ ВИБІРКИ З МАЛОЮ КІЛЬКІСТЮ ЧИСЛОВИХ ДАНИХ ІЗ ВИКОРИСТАННЯМ ЗАГОЛОВКА | 45 |
| 4.8 КЛАСИФІКАЦІЯ ВИБІРКИ З МАЛОЮ КІЛЬКІСТЮ ЧИСЛОВИХ ДАНИХ БЕЗ ВИКОРИСТАННЯ ЗАГОЛОВКА | 47 |
| 4.9 ЗАГАЛЬНЕ ПОРІВНЯННЯ ВИКОРИСТАНИХ АЛГОРИТМІВ КЛАСИФІКАЦІЇ ДЛЯ РІЗНИХ НАБОРІВ ВХІДНИХ ДАНИХ | 48 |
| ВИСНОВКИ | 49 |
| ПЕРЕЛІК ПОСИЛАНЬ | 51 |

**Вступ**

Книги є одним із найдавінших засобів збереження та поширення знань в світі. Якщо взяти до уваги очікувану тривалість життя, і дані про те, що людина, в середньому читає близько 12 книг на рік, ми отримуємо, що протягом життя одній особі знадобиться принаймні 700 книг [1]. Очевидно, що особисто зберігати таку кількість в домашніх умовах є неоптимальним, а, навіть з поширенням цифрової дистрибуції книг, значна кількість робіт залишається неоцифрованою. Тому, потреба в публічних бібліотеках була і залишається дуже високою.

Фонди публічних бібліотек налічують сотні мільйонів записів, більшість з яких складають саме книги. При надходженні книги до бібліотеки, її дані вносяться в реєстр і віденсення книги до певної категорії здійснюється персоналом вручну. Ми вважаємо, що, із залученням до даної категоризації методів інтелектуального аналізу даних, цей процес можна значно спростити і, можливо, в подальшому навіть автоматизувати.

В рамках даної курсової роботи здійснена класифікація та кластеризація книг з використанням даних бібліотечного фонду міста Сієтл. Виконане об’єднання з даними із зовнішніх джерел. Здійснене порівняння різних алгоритмів класифікації та кластеризації з використанням різних комбінацій вхідних даних з основного та додаткових джерел. Зроблені висновки стосовно найефективнішого з розглянутих алгоритмів класифікації для даної задачі та висунуті гіпотези стосовно необхідного набору вхідних даних про запис для досягнення максимальної точності.

Система реалізована на мові програмування Python3 із використанням зовнішніх статистичних бібліотек та бібліотек машинного навчання. Застосунок доступний у формі сторінки Jupyter notebook.

**1.Постановка задачі**

Під час виконання курсової роботи необхідно виконати наступні завдання:

Отримання достатньої для аналізу вибірки даних з основного сховища, яким виступає найновіша версія дампу даних бібліотечного фонду міста Сієтл. Вибірка повинна містити принаймні 10 000 записів про англомовні книги із наявними даними про ISBN та не включати жодних записів, які не задовільняють дані вимоги.

Здійснення очистки даних з використанням регулярних виразів, виділення категористичних сутностей видавців методами кластеризації на основі ортографічної відстані між попередньо очищеними назвами та заповнення відсутніх даних про автора виділенням текстових сутностей із наявних додаткових даних про книгу.

Додавання відсутніх в базовому наборі даних параметрів книги із зовнішніх джерел для утворення меншої за об’ємом, але більш повної за кількістю даних вибірки.

Проведення кластеризації змішаних числових та категористичних даних про книги методами K-means та Agglomerative clustering. Здійснення PCA для K-means кластеризації. Порівняння впливу комбінацій очищення назви книги та використання повної або скороченої назви на результати Agglomerative кластеризації.

Порівняння класифікації доповненої цифровими даними вибірки з використанням алгоритмів Random forest, Support Vector Machines і Gradient boosting. Порівняння результатів роботи вказаних алгоритмів за умови використання векторизованої очищеної назви книги як параметра та за її відсутності у вхідних даних.

Здійснення порівняння класифікації за вищезгаданими алгоритмами для вибірки, в якій не були додані числові дані із зовнішніх джерел. Порівняння на основній вибірці використання назви книги як параметра та її відсутності як вхідних даних для роботи алгоритму.

**2.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

В загальноприйнятому розумінні кожну книгу можна охарактеризувати її назвою, автором, видавцем, обкладинкою, роком випуску та жанрами, до яких книга відноситься. У випадку фізичної книги часто додається кількість сторінок і габарити. ISBN не є характеристикою книги, а її ідентифікатором, єдина інформація, яку можна з нього легко отримати – яка ISBN агенція, що діє на певній території, цю книгу зареєструвала і для якої видавничої групи.

Назва книги є її найважливішою характеристикою для людини, всі ми ідентифікуємо книги в першу чергу саме за назвою і навіть можемо здійснювати припущення про жанрову належність книги лише за назвою. Проте, коли мова йде про автоматизовану якимось чином систему, текстові дані часто вимагають значно більше ресурсів для обробки ніж числові чи категористичні. Тому межах роботи ми висуваємо гіпотезу, що врахування назви у класифікації книг покращує точність передбачення, але погіршує швидкодію програмного рішення. Питання, на якому дана робота буде значною мірою зосереджена – за умови підтвердження вищезгаданої гіпотези, чи буде виграш в точності для даного застосунку виправдовувати збільшення необхідних для даного рішення програмних ресурсів.

У класифікації книг найчастіше виникає проблема із жанровою належністю книги. Проблема полягає у відсутності уніфікованої системи визначення жанрів, жанрове визначення книги може відрізнятися в різних культурах, більшість жанрів не є взаємовиключними. Прикладом культурних відмінностей в жанровому визначенні може слугувати Японія, де, окрім звичного для нашої культури тропового жанрового опису, в жанрах є також опис цільової аудиторії творів, як, наприклад «шьонен», «шьоджо», «сейнен» і «джосей», які буквально перекладаються як «хлопчик», «дівчинка», «юнак» (хоча, зазвичай, мається на увазі доросла чоловіча аудиторія) і «жінка».

Коли мова йде про класифікацію книг з метою використання обраних класів для бібліотеки, важливим є саме поєднання тропового жанрового опису з жанрами, які позначають цільову аудиторію. Це часто необхідно для близького фізичного розташування пов’язаних цільовою аудиторією і жанром книг в читальному залі. Для більшості бібліотек такі класи книг є чітко визначені в межах бібліотечної системи, при чому цільова аудиторія є важливішим фактором в такій класифікації. Зокрема, таку класифікацію з тенденцією до пріоритезації цільової аудиторії можна спостерігати і на розглянутій в роботі базі даних, де такий псевдожанр називають колекцією.

Для полегшення пошуку в межах бібліотеки, крім колекції, часто в інформації про книгу зазначають теми, які в ній порушуються. Це відмінна від жанру характеристика, більше схожа на користувацькі теги які часто можна зустріти в книжкових рекомендаційних системах, наприклад в Goodreads. Власне, це один із способів, яким такі рекомендаційні системи вирішують проблему відсутності стандартизації жанрів – користувачі можуть голосувати за певний вже популярний тег для книги або додати власний. При внесенні книги в бібліотечний реєстр теми, найімовірніше, визначає та вказує співробітник бібліотеки. Тому, хоча дана характеристика книги є безумовно важливою, вона абсолютно не підходить як ціль класифікації. Більше того, якщо ми намагаємося рухатися до автоматизації системи, її врахування може бути шкідливим, оскільки додавання даної характеристики до запису про книгу сильно залежить від людської роботи і пов’язаних з цим часових затрат та факторів ризику.

Власне, дана робота полягає у перевірці гіпотез стосовно точності класифікації книг за їх колекціями за умов різних комбінацій наявних даних, які не включають в себе тематику книг. У програмному забезпеченні буде реалізований такий функціонал:

* Відбір вибірки записів в достатньому для аналізу розмірі паперових книг англійською мовою з дампу бази даних визначеного бібліотечного фонду
* Очищення отриманих даних з метою зведення їх до оптимального для аналізу вигляду
* Підключення додаткових джерел для отримання більш повних даних про книги
* Кластеризація даних для виявлення тенденцій у вибірці
* Експеримент з кластеризацією даних за умови використання текстових даних про назву книги в різних формах
* Класифікація книг з використанням доповнених зовнішніми джерелами даних за умов наявності та відсутності даних про назву книги
* Класифікація книг з даними лише з основного джерела за умов наявності та відсутності даних про назву книги
* Графічне відображення результатів та їх аналіз

**3.ОЧИЩЕННЯ ДАНИХ**

**3.1 Опис джерела даних та отримання вибірки**

Дані про книги отримані з американського державного бібліотечного реєстру міста Сієтл, який можна знайти за посиланням https://data.seattle.gov/Community/Library-Collection-Inventory/6vkj-f5xf . У даному реєстрі міститься інформація про: унікальний бібліотечний ідентифікатор, повну назву предмета, ім'я автора, ISBN, рік публікації, видавця, тематики, тип предмета, колекція до якої належить предмет, індикатор неналежності предмету до конкретної локації, місцезнаходження предмета, дата надходження до бібліотечного фонду та кількість таких предметів в бібліотечному фонді.

Вказане джерело даних містить не лише книги, а й інші предмети, які можуть знаходитися в бібліотечному фонді, зокрема дискети, відео та аудіо касети, CD і DVD диски, журнали, мікрофільми, аудіозаписи, слайди та інше. Інформація про те, якому саме типу предмета відповідає запис в базі даних міститься у стовпчику 'ItemType'. Типи в базі даних містяться під кодовими назвами, їхню розшифровку можна знайти у файлі 'CollectionInventory\_Codes\_EXCLUDED\_INCLUDED.xlsx'. Здійснивши дослідження вищенаведеної розшифровки, легко бачити, що для виділення із бази даних предметів лише паперових книг достатньо знайти всі предмети, в яких код типу предмету відповідає одному з наведених: 'bcbk', 'acbk', 'ahbk', 'jcbk', 'arbk', 'jrbk'.

Даний бібліотечний фонд містить предмети, назви та вміст яких не обов'язково є англомовним. Попереднє дослідження набору даних продемонструвало, що більшість неангломовних предметів належать до колекцій, які, згідно файлу 'CollectionInventory\_Codes\_EXCLUDED\_INCLUDED.xlsx', містять в своєму описі слово Language. Крім цього, якщо попередньо відфільтрувати дані на предмет того, що за своїм типом відповідають книгам, достатньо не включати у вибірку лише ті неангломовні колекції, які стосуються книг. Інформацію про групу типів предметів, яких стосується колекція, можна знайти у описовому файлі в стовпчиках 'FormatGroup' і 'FormatSubgroup'. Такими колекціями є: 'caln', 'calnr', 'ccln', 'naln', 'nalnpb', 'ncln', 'cs1lew' та 'cs1lewr'.

Кожна книга в межах обраної для аналізу бази даних ідентифікується унікальним бібліотечним номером, проте цей ідентифікатор діє лише в межах цієї бази даних і не є універсальним. У випадку книг найкращим ідентифікатором слугуватиме ISBN, що дозволить надалі додавати до отриманої вибірки інформацію з інших джерел для аналізу, наприклад, кількість сторінок і зображення обкладинки. Попереднє дослідження вибірки, яка включає лише книги, продемонструвало, що в більшості випадків, коли дані про ISBN книги відсутні, даний предмет є або неангломовною книгою або п'єсою. Таким чином, виключення із вибірки предметів, ISBN яких відсутній в базі даних, найімовірніше, не завдасть вибірці жодної шкоди а лише покращить її розширюваність та здатність до модифікації.

Повна база даних на момент здачі роботи містить приблизно 116 000 000 предметів і оновлюється щомісяця. Оскільки аналіз такого об'єму даних вимагає значних комп'ютерних потужностей, було прийняте рішення виділити з даного джерела вибірку із 10 000 предметів, які є англомовними книгами що містять дані про присвоєний їм ISBN.

Завантаження даних здійснюється за допомогою функції:

def get\_english\_books\_with\_isbn(min\_number\_of\_records = 1000, database\_location = r'C:\Users\trush\Downloads\Library\_Collection\_Inventory\_20231102.csv'):  
 i = 1  
 book\_codes = ['bcbk', 'acbk', 'ahbk', 'jcbk', 'arbk', 'jrbk']  
 column\_names = ['BibNum', 'Title', 'Author', 'ISBN', 'PublicationYear', 'Publisher', 'Subjects', 'ItemType', 'ItemCollection', 'FloatingItem', 'ItemLocation', 'ReportDate', 'ItemCount']  
 language\_collections = ['caln', 'calnr', 'ccln', 'naln', 'nalnpb', 'ncln', 'cs1lew', 'cs1lewr']  
 books = pd.DataFrame()  
 while books.shape[0] < min\_number\_of\_records:  
 temp = pd.read\_csv(r'C:\Users\trush\Downloads\Library\_Collection\_Inventory\_20231102.csv', on\_bad\_lines='skip', skiprows=i, nrows=1000, header=None, names=column\_names)  
 temp = temp.loc[temp['ItemType'].isin(book\_codes)]  
 temp = temp.loc[~temp['ItemCollection'].isin(language\_collections)]  
 temp = temp.loc[temp['ISBN'].notna()]  
 books = pd.concat([books, temp], ignore\_index=True)  
 i += 1000  
 return books

books\_sample = get\_english\_books\_with\_isbn(10000)

Отриману вибірку виносимо у зовнішній файл для подальшого перетворення та очищення даних. Створена функція дозволяє за потреби отримати вибірку інших розмірів та, за незначних модифікацій, здійснювати фільтрування предметів з іншої бази даних та з іншої стартової точки.

books\_sample.to\_csv('sample\_final.csv', index=False)

**3.2 Підготовчий аналіз вхідних даних**

Вивантажуємо дані з файлу вибірки і проводило аналіз.

books = pd.read\_csv('sample\_final.csv')  
books

Зображення, що містить знімок екрана, комп’ютер, клавіатура

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований описbooks.isna().sum()

Аналіз отриманої вибірки на предмет відсутніх даних демонструє, що індикатор неналежності предмету до конкретної локації є відсутнім значенням для практично всієї вибірки. Даний показник є за визначенням бінарним, відсутніх значень можна позбутися заповнивши їх нулями або рядковими значеннями, наприклад, 'NotFloating'. Проте, в такому випадку розподіл даних значень буде дуже нерівномірним, що при аналізі принесе більше шкоди аніж користі. Таким чином було прийняте рішення в даній вибірці цей індикатор не враховувати, тому стовпчика 'FloatingItem' можна позбутися.

books = books.drop('FloatingItem', axis=1)

**3.3 ISBN**

ISBN - міжнародний унікальний ідентифікатор книги, який присвоюється окремо кожному виданню. До 2007 року стандартом був формат ISBN-10, який складався з десяти цифер, де X може позначати число 10 в чексумі. З 2007-го книгам присвоюється тільки ISBN-13, який складається з тринадцяти цифер. Реєструється видавничою групою через локальне ISBN агентство, тому, ймовірно, між ISBN та видавництвом буде помітна кореляція. Видавнича група може звернутися до ISBN агентства із запитом додати для книги за наявного ISBN-10 відповідний ISBN-13. [2] Саме тому, в нашій вибірці ми спостерігаємо кілька ISBN на одну книгу, а іноді для однієї книги і кілька наборів ISBN-10 і ISBN-13.

В межах даної вибірки кожна книга має хоча б одне значення ISBN. Для подальшого аналізу виділимо всі дані про ISBN в окремий датафрейм, при чому, через наявність в ISBN-10 символу X для позначення 10, та, з метою подальшого розділення на ISBN-10 та ISBN-13 за допомогою регулярних виразів, будемо розглядати значення отриманого стовпчика як рядкового типу.

isbn\_df = books['ISBN'].copy().to\_frame(name='ISBN\_dirty')  
isbn\_df['ISBN\_dirty'] = isbn\_df['ISBN\_dirty'].astype('string')

Тепер можна легко перетворити рядкові значення у список всіх наявних в базі даних ISBN для кожної книги.

isbn\_df['ISBN\_list'] = isbn\_df['ISBN\_dirty'].str.split(', ').tolist()

За допомогою регулярних виразів здійснимо видобування зі списків по одному відповідному значенню ISBN. Це зручно виконати застосуванням функцій на відповідний стовпчик. Хоча, за наявності кількох наборів ISBN-10 і ISBN-13 для однієї позиції ми в результаті отримуємо лише один, який використовуватиметься як основний, зберігаючи всі наявні дані про ISBN в списку, ми все ще залишаємо за собою можливість звернутися до інших версій за потреби.

def check\_isbn\_10(isbn\_list):  
 isbn\_10 = r"^(?:ISBN(?:-10)?:?\ )?(?=[0-9X]{10}$|(?=(?:[0-9]+[-\ ]){3})[-\ 0-9X]{13}$)[0-9]{1,5}[-\ ]?[0-9]+[-\ ]?[0-9]+[-\ ]?[0-9X]$"  
 for i in range(len(isbn\_list)):  
 if re.search(isbn\_10, isbn\_list[i]) is not None:  
 return isbn\_list[i]  
 return None  
  
def check\_isbn\_13(isbn\_list):  
 isbn\_13 = r"^(?:ISBN(?:-13)?:?\ )?(?=[0-9]{13}$|(?=(?:[0-9]+[-\ ]){4})[-\ 0-9]{17}$)97[89][-\ ]?[0-9]{1,5}[-\ ]?[0-9]+[-\ ]?[0-9]+[-\ ]?[0-9]$"  
 for i in range(len(isbn\_list)):  
 if re.search(isbn\_13, isbn\_list[i]) is not None:  
 return isbn\_list[i]  
 return None

isbn\_df['ISBN10'] = isbn\_df['ISBN\_list'].apply(check\_isbn\_10)  
isbn\_df['ISBN13'] = isbn\_df['ISBN\_list'].apply(check\_isbn\_13)

Оцінимо для даної вибірки, яка версія ISBN більш придатна для того, щоб використовуватись для видобування додаткових даних.

isbn\_df.isna().sum()

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Робимо висновок, що оскільки відсутніх даних про ISBN-10 значно менше, ніж про ISBN-13, саме перший з цих форматів краще підходить для формування потенційних запитів на додаткові дані з інших баз даних.

Отримані дані можна додати як стовпці до базового датафрейму книг і позбутися стовпчика із неочищеними даними про ISBN.

books.insert(3, 'ISBN13', isbn\_df['ISBN13'])  
books.insert(3, 'ISBN10', isbn\_df['ISBN10'])  
books = books.drop('ISBN', axis=1)

**3.4 Рік публікації**

Наступний стовпчик, що потребує очищення - рік публікації. У кожної книги може бути тільки один рік публікації і, в межах цієї вибірки, рік буде чотиризначним цифровим значенням у стовпчику 'PublicationYear'. В даному випадку очищення можна здійснити "на місці".

books['PublicationYear'] = books['PublicationYear'].str.replace(r'\D', '', regex=True)

Також приймаємо, що кожна книга з присвоєним ISBN видана конкретним видавцем. Легко бачити, що дані в стовпчику, який містить інформацію про видавця, є помітно забрудненими, тому, для зручності очищення, найкраще винести його в окремий датафрейм. На відміну від ISBN, де ми припускали, що код для кожної книги є унікальним саме для неї, один видавець може мати множину відповідних книг у базі.

publisher\_df = pd.DataFrame()

#%%  
publisher\_df = books['Publisher'].copy().to\_frame(name='Publisher\_dirty')

**3.5 Видавець**

Створимо функцію для очистки текстових даних. Вважаємо, що всі назви видавництв, які в рядку йдуть після слешу ‘/’ є альтернативними, тож їх можна відкинути. Всі спецзнаки і посилання мають бути відкинуті. Назви видавництв повинні бути у ловеркейсі, між словами має йти одиночний пробіл.

def clean\_publisher(publisher\_name\_dirty):  
 publisher\_name\_clean = "None"  
 if not publisher\_name\_dirty is None:  
 publisher\_name\_clean = publisher\_name\_dirty.strip(',')  
 publisher\_name\_clean = publisher\_name\_clean.lower()  
 publisher\_name\_clean = publisher\_name\_clean.split("/")[0]  
 publisher\_name\_clean = re.sub(r"(@\[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+?", " ", publisher\_name\_clean)  
 publisher\_name\_clean = re.sub(r" +", " ", publisher\_name\_clean)  
 #print(publisher\_name\_clean)  
 return publisher\_name\_clean

Переводимо дані у рядковий тип і застосуємо отриману функцію.

publisher\_df['Publisher\_dirty'] = publisher\_df['Publisher\_dirty'].astype('string')  
publisher\_df['Publisher\_dirty'] = publisher\_df['Publisher\_dirty'].fillna('None')  
publisher\_df['Publisher\_clean'] = publisher\_df['Publisher\_dirty'].apply(clean\_publisher)  
publisher\_df

Після базової очистки тексту бачимо, що назви видавництв, які повинні бути однією сутністю, часто відрізняються доданими суфіксами, що не є оптимальним. Навіть дані суфікси можуть бути записані в межах вибірки по-різному, наприклад, одна назва містить слово 'pub', а інша використовує повне 'publishing'. Крім цього, деякі видавництва є частиною певної видавничої групи або підрозділом іншого видавництва. В даній вибірці не для всіх записів про назву видавництва вказано чи є воно підрозділом іншого видавництва. Таким чином, для того, щоб чітко виділити сутності, краще позбутися як вищезгаданих суфіксів, так і додаткової інформації про батьківське видавництво або видавничу групу.

publisher\_suffixes = ['publishing', 'press', 'books', 'co', 'book', 'publishers', 'pub', 'press', 'entertainment', 'media', 'publications', 'production', 'company', 'collection', 'illustrated', 'ink', 'ltd', 'publication', 'group', 'paperback', 'house', 'inc', 'company', 'productions', 'llc', 's', 'an', 'a']  
publisher\_imprint = ['imprint', 'an imprint', 'a division', 'published', 'books', 'in association', 'distributed', 'are imprints', 'a member', 'press of', 'large print']

До суфіксів publisher\_suffixes також входять деякі артиклі і вставні слова, що можуть залишатися після видалення додаткової інформації з назви. Інформація про батьківське видавництво повинна видалятися починаючи зі слів у списку publisher\_imprint до кінця рядка, включно із зазначеним словом. Суфікси повинні видалятися чітко після видалення даних про батьківське видавництво. Отримані списки були зібрані методом звичайного перегляду вибірки і не є вичерпними.

publisher\_df['Publisher\_clean\_imprints'] = publisher\_df['Publisher\_clean']  
for pub\_imp in publisher\_imprint:  
 pattern = re.compile(f'\\b{pub\_imp}\\b.\*$')  
 publisher\_df['Publisher\_clean\_imprints'] = publisher\_df['Publisher\_clean\_imprints'].apply(lambda x: re.sub(pattern, '', str(x)))

publisher\_df['Publisher\_clean\_imprints\_cut'] = publisher\_df['Publisher\_clean\_imprints']  
for pub\_suf in publisher\_suffixes:  
 pattern\_1 = re.compile(f'\\b{pub\_suf}\\b')  
 publisher\_df['Publisher\_clean\_imprints\_cut'] = publisher\_df['Publisher\_clean\_imprints\_cut'].apply(lambda x: re.sub(pattern\_1, '', str(x)))

publisher\_df['Publisher\_clean\_imprints\_cut'] = publisher\_df['Publisher\_clean\_imprints\_cut'].str.strip()  
publisher\_df['Publisher\_clean\_imprints\_cut'] = publisher\_df['Publisher\_clean\_imprints\_cut'].apply(lambda x: re.sub(r" +", ' ', str(x)))

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Виділяємо унікальне написання видавців, факторизуючи очищені і обрізані назви видавців. Найзручніше буде виділити для унікальних написаннь окремий датафрейм, на який таблиця publisher\_df має посилання і над яким ми можемо здійснювати подальші кроки з очищення і нормалізації.

publisher\_unique\_df = pd.DataFrame()  
publisher\_df['FK\_id'], publisher\_unique\_df['Publisher'] = pd.factorize(publisher\_df['Publisher\_clean\_imprints\_cut'])

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Розсортувавши дану вибірку в алфавітному порядку, легко бачити що, хоча, звичайне очищення тексту значно зменшило кількість унікальних видавців, проте ми все ще легко знаходимо назви, які явно належать одній сутності з відмінностями в написанні. Більше того, частина відмінностей між виділеними назвами видавництв, очевидно, пов'язана з одруками та використанням унікальних суфіксів, що позначають підрозділи конкретного видавництва. З цього можна зробити висновок, що отримана вибірка придатна до подальшого очищення. Крім цього, суть відмінностей у отриманих назвах видавництв наштовхує на думку, що для очищення слід скористатися певними метриками відстані між рядковими значеннями.

Для даного очищення такою метрикою була обрана відстань Левенштайна, яка визначає кількість змін, які необхідно здійснити над одним рядковим літералом для перетворення його в інший. Дана метрика добре працює для коротких рядків, якими і є наші назви видавництв. [3]

Якби у нас був наявний список всіх допустимих видавництв, задача би звелась до порівняння назв видавництв із вибірки з правильними назвами і заміною назви з вибірки на найкращий відповідник. Однак, за відсутності такого списку, перед нами стоїть задача кластеризації. На даній вибірці дуже непогано працює K-means кластеринг з кількістю кластерів в розмірі приблизно третини вибірки. Така кількість кластерів виявлена експериментально для цієї конкретної вибірки в ході роботи і може бути уточнена або змінена для інших даних. Один або більше кластерів виявляться такими, що містять змішані дані, в той час, як решта - дані з малою ортографічною відстанню між собою, достатньою для того, щоб їх можна було вважати однією сутністю.

from fuzzywuzzy import fuzz  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.cluster import KMeans

def compute\_similarity(publisher1, publisher2):  
 return 100 - fuzz.ratio(publisher1, publisher2)

unique\_publishers = publisher\_unique\_df['Publisher'].unique()  
  
similarity\_matrix = pd.DataFrame(index=unique\_publishers, columns=unique\_publishers)  
for publisher1 in unique\_publishers:  
 for publisher2 in unique\_publishers:  
 similarity\_matrix.loc[publisher1, publisher2] = compute\_similarity(publisher1, publisher2)

vectorizer = TfidfVectorizer()  
tfidf\_matrix = vectorizer.fit\_transform(unique\_publishers)  
  
num\_clusters = 700  
kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)  
cluster\_labels = kmeans.fit\_predict(tfidf\_matrix)  
  
publisher\_unique\_df['Cluster'] = publisher\_unique\_df['Publisher'].map(dict(zip(unique\_publishers, cluster\_labels)))

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований описОтримані результати виведемо на екран:

Знайдемо найбільші кластери.

publisher\_unique\_df['Cluster'].value\_counts()

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

Проглянувши елементи, що потрапили в найбільші кластери, можна помітити, що кластер з індексом 36 - видавці, які не мають схожих назв у переліку, 531 - видавництва університетів, 58 - ймовірно позначають одну сутність, 139 - починаються зі слова 'new' і не обов'язково позначають одну і ту ж сутність. Вибірковий огляд менших груп демонструє, що записи об'єднані достатньо точно для нашої задачі, хоча, ймовірно, застосування інших алгоритмів кластеризації може цю точність покращити.

Отже, для всіх кластерів, крім 36, 531 і 139, ми ставитимемо у відповідність найкоротшу назву видавництва з кластера, до якого вони належать. Для всіх інших використовуватиметься їхня назва очищена на попередньому етапі.

exclude\_clusters = [36, 531, 139, 2, 8, 16]  
shortest\_strings = []  
  
for cluster\_label in range(num\_clusters):  
 if cluster\_label not in exclude\_clusters:  
 cluster\_indices = (publisher\_unique\_df['Cluster'] == cluster\_label)  
 cluster\_strings = publisher\_unique\_df.loc[cluster\_indices, 'Publisher']  
 shortest\_string = min(cluster\_strings, key=len, default=None)  
 shortest\_strings.append(shortest\_string)  
 else:  
 shortest\_strings.append(None)  
  
publisher\_unique\_df['MappedPublisher'] = publisher\_unique\_df['Cluster'].map(dict(enumerate(shortest\_strings)))  
publisher\_unique\_df['MappedPublisher'] = publisher\_unique\_df['MappedPublisher'].fillna(publisher\_unique\_df['Publisher'])

Витяг з результату роботи кластеризації:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Через явну невідповідність одній сутності, після повторного перегляду до ігнорованих додані кластери 2, 8 і 16. Можемо додати знайдені нормалізовані назви видавництв до основного датафрейму та перевірити вибірку на збалансованість за наявними видавництвами.

for val in range(len(publisher\_unique\_df)):  
 publisher\_df.loc[publisher\_df['FK\_id'] == val, ['ClusteredPublisher']] = publisher\_unique\_df.at[val, 'MappedPublisher']

books.insert(6, 'PublisherCl', publisher\_df['ClusteredPublisher'])

books = books.drop('Publisher', axis=1)  
books = books.rename(columns={'PublisherCl' : 'Publisher'})

books['Publisher'].value\_counts()

Зображення, що містить знімок екрана, текст, схема, Графік

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

**3.6 Автор**

Наступна очистка, яку необхідно здійснити, це назви книг та імена авторів. В обраній базі даних ім'я автора книги зазначене в стовпчику 'Author' в порядку прізвище та ім'я через кому, при чому, в більшості записів для книги вказаний лише один автор. Однак, якщо розглянути стовпчик 'Title' в якому зазначена назва книги, легко побачити, що крім назви після слешу '/' у стовпчику також вказана додаткова інформація про книгу. Додаткова інформація містить список авторів і/або під чиєю редакцією випущена книга і/або автора ілюстрацій etc. Крім цього, сам стовпчик з іменами авторів дещо забруднений. Зокрема для частини книг автори не вказані, або, крім імен, додані числа, найімовірніше, роки життя. Питання того, чи допомагає додавання років життя авторів для даної бази з, наприклад, розрізненням кількох авторів з однаковим іменем, може бути важливим і є дискусійним. Проте, для даної вибірки, за попередніми спостереженнями, така проблема не стоїть, тому всі роки з імен авторів при очищенні слід видаляти.

Для того, щоб якнайточніше відобразити авторів книги не звертаючись до зовнішніх джерел, необхідно паралельно з очищенням даних про авторів, здійснювати очистку даних про назву книги, виділяючи додаткову інформацію зі стовпчика назви та порівнюючи із наявними даними про автора зі стовпчика авторів.

titles\_and\_authors = pd.DataFrame()  
titles\_and\_authors[['Title', 'Author']] = books[['Title', 'Author']]

Розділимо стовпчик 'Title' на назву та підназву для порівняння списку авторів з підназви із вказаними в базі даних авторами.

titles\_and\_authors[['TitleMain', 'Subtitle']] = titles\_and\_authors['Title'].str.split('/', n=1, expand=True)

Легко бачити, що для книг, в яких відсутнє значення в стовпчику ‘Author’ може бути зазначений автор у підназві. Дослідимо це явище, виділивши підмножину книг, в яких не зазначений автор.

rows\_without\_author = titles\_and\_authors[titles\_and\_authors['Author'].isna()]

rows\_without\_author['Subtitle'].isna().sum()

Відносно всієї вибірки, приблизно в 5% (117) записів про книг не містять інформації про автора. В підмножині книг, для яких відсутній автор, приблизно 20% записів про книги, які не мають зазначеного автора, також не мають жодної додаткової інформації. Розглянувши дану підмножину книг, можна виділити такі категорії:

1. Книги з великою кількістю співавторів, наприклад, збірники есе на певну тему.

2. Книги видані організацією, де авторство належить не одній особі, а всій організації.

3. Довідкова література, підручники та каталоги, де за відсутності автора в додатковій інформації вказується редактор.

4. Релігійні тексти, де вказується лише перекладач.

5. Фольклор, де вказується укладач, перекладач або редактор.

6. Дитячі книги, де в додатковій інформації вказаний лише ілюстратор.

7. Книги, де автор чітко зазначений в додатковій інформації, але не внесений в стовпчик авторів через людську помилку.

Якщо ми прагнемо до максимальної чистоти та густини даних без звертання до зовнішніх джерел, найпростішим виходом буде викинути записи з відсутніми даними, оскільки ця проблема вражає відносно малу частину вибірки. Проте, якщо ми в результаті захочемо підтягувати більше записів, особливо якщо нас цікавитиме в межах аналізу якась із виділених категорій, дана проблема може постати більш гостро. Обробкою підзаголовка частину з цих даних можна врятувати.

Визначимо набір логічних правил, за якими будемо виділяти автора з довідкової інформації:

1. Автором може бути тільки людина. Тобто, в книг за авторством організацій без додаткової інформації автор буде відсутній.

2. У вибірку буде внесений тільки один автор на книгу. Якщо в додатковій інформації перелічено кілька авторів, буде обраний перший в списку.

3. За відсутності автора, автором можна вносити редактора, ілюстратора або перекладача.

4. Припускаємо, що особи, що долучилися до створення книги вказані в порядку спадання їх внеску, тому перша людина в списку імен в додатковій інформації (підзаголовоку) вважається головним автором.

Враховуючи ці правила, можна оглянути підмножину записів з відсутніми авторами і, за допомогою бібліотеки spacy, знайти авторів в підзаголовках.

import spacy

nlp = spacy.load('en\_core\_web\_sm')

def extract\_names(text):  
 if pd.notna(text):  
 doc = nlp(text)  
 names = [ent.text for ent in doc.ents if ent.label\_ == 'PERSON']  
 return names  
 else:  
 return np.nan

titles\_and\_authors['Subtitle'] = titles\_and\_authors['Subtitle'].astype('str')  
titles\_and\_authors['SubtitleAuth'] = titles\_and\_authors.loc[titles\_and\_authors['Author'].isna(), 'Subtitle'].apply(extract\_names)

Зображення, що містить текст, знімок екрана, програмне забезпечення, Шрифт

Автоматично згенерований опис

При перегляді результатів витягування повних імен із підзаголовку, можна зробити висновок, що для більшості з них спочатку йде ім'я, опціонально друге (так зване середнє) ім'я, прізвище вкінці. Натомість, для стовпчика 'Author' спочатку зазначене прізвище, а потім ім'я. Необхідно звести все до єдиного формату. В межах даної роботи, оскільки не розглядається потенційний аналіз книг за певними характеристиками імені автора, повне ім'я можна вказати не розділяючи на частини і нормалізувати відображення імені буде зручніше у форматі "ім'я прізвище" (на противагу формату "прізвище ім'я").

def strip\_everything(dataframe, column\_name):  
 pattern = r'^[^A-Za-z\.]+|[^A-Za-z\.]+$|\(.\*$'  
 dataframe[column\_name] = dataframe[column\_name].str.replace(pattern, '', regex=True)  
 return dataframe

titles\_and\_authors['AuthorClean'] = titles\_and\_authors['Author'].astype('string')  
titles\_and\_authors = strip\_everything(titles\_and\_authors, 'AuthorClean')

titles\_and\_authors[['AuthSurname', 'AuthName']] = titles\_and\_authors['AuthorClean'].str.split(', ', n=1, expand=True)

titles\_and\_authors['AuthName'] = titles\_and\_authors['AuthName'].str.replace(r',.\*$', '', regex=True)

def get\_first\_author(lst):  
 return lst[0] if (lst and isinstance(lst, list)) else np.nan

titles\_and\_authors['AuthorFinal'] = None  
titles\_and\_authors['AuthorFinal'] = titles\_and\_authors['AuthorFinal'].fillna(titles\_and\_authors['SubtitleAuth'].apply(get\_first\_author))

titles\_and\_authors['AuthorFinal'] = titles\_and\_authors['AuthorFinal'].fillna(titles\_and\_authors['AuthName'] + ' ' + titles\_and\_authors['AuthSurname'])

titles\_and\_authors['AuthorFinal'] = titles\_and\_authors['AuthorFinal'].str.replace(r' +', ' ', regex=True)  
titles\_and\_authors['AuthorFinal'] = titles\_and\_authors['AuthorFinal'].str.strip()

**3.7 Заголовок і підзаголовок**

Таким чином дані про авторів можна вважати очищеними і їх пожна перенести в головний датафрейм. Проте, це не можна сказати про заголовки книг. Додаткова інформація, яку ми до цього взаємозамінно назвили то так, то підзаголовком, є саме лише додатковою інформацїєю. Якщо проаналізувати назви книг, які залишилися після її видалення, стане чітко зрозуміло, що у книг після двокрапки ":" вказаний саме підзаголовок у класичному розумінні. Задля очищення даних, заголовок і підзаголовок доречно розділити, та зберігати в окремих стовпчиках головного датафрейму.

titles\_and\_authors[['CleanTitle', 'CleanSubtitle']] = titles\_and\_authors['TitleMain'].str.split(' : ', n=1, expand=True)

books.insert(2, 'AuthorCl', titles\_and\_authors['AuthorFinal'])  
books.insert(2, 'Subtitle', titles\_and\_authors['CleanSubtitle'])  
books.insert(2, 'TitleCl', titles\_and\_authors['CleanTitle'])

books = books.drop('Title', axis=1)  
books = books.drop('Author', axis=1)  
books = books.rename(columns={'TitleCl' : 'Title', 'AuthorCl' : 'Author'})

Створимо функції для очистки тексту заголовків та підзаголовків.

import nltk  
nltk.download('stopwords')  
nltk.download('wordnet')  
from nltk.corpus import stopwords

def preprocess\_text(text):  
 text = text.lower()  
 text = re.sub(r'\d+', '', text)  
 text = re.sub(r'[^\w\s]', '', text)  
 tokens = nltk.word\_tokenize(text)  
 return tokens

def remove\_stopwords(tokens):  
 stop\_words = set(stopwords.words('english'))  
 filtered\_tokens = [word for word in tokens if word not in stop\_words]  
 return filtered\_tokens

def perform\_lemmatization(tokens):  
 lemmatizer = nltk.WordNetLemmatizer()  
 lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in tokens]  
 return lemmatized\_tokens

def full\_clean\_text(text):  
 if pd.notna(text):  
 tokens = preprocess\_text(text)  
 filtered\_tokens = remove\_stopwords(tokens)  
 lemmatized\_tokens = perform\_lemmatization(filtered\_tokens)  
 clean\_text = ' '.join(lemmatized\_tokens)  
 return clean\_text

Вказані функції виконують видалення будь-яких немовних знаків з рядка, очищення від стоп слів та лематизацію. Виконання їх над рядковим літералом підряд можна здійснити викликом на рядковому літералі об'єднуючої функції full\_clean\_text.

**3.8 Обробка тем книги**

Для обраного набору даних найцікавішим елементом був стовпчик 'Subjects', що містить теми, які зачіпаються в кожній книзі. Очікувалося, що, в межах бази даних бібліотеки, такі описи тем будуть значно більш стандартизовані, ніж при використанні даних із сайтів продажу та рекомендаційних сервісів книг. Проте, можна побачити, що, хоча між даними темами часто трапляються спільні слова і вони значною мірою перетинаються, їхня ортографічна близькість не означає належність одній сутності, тому, виділення сутностей методом, який ми використали для видавництв, принесе більше шкоди, ніж користі.

books['Subjects'] = books['Subjects'].astype('string')

all\_subjects = []  
def find\_all\_subjects(text):  
 if pd.notna(text):  
 for t in text.split(', '):  
 all\_subjects.append(t.strip())

books['Subjects'].apply(find\_all\_subjects)

subjects\_df = pd.DataFrame(all\_subjects, columns=['Subject'])

subjects\_df.value\_counts().value\_counts()

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Як вже було згадано, через їхню кількість і різноманіття, теми книг погано підходять як категорії-цілі для класифікації.

**3.9 Інші вхідні дані**

Решта стовпчиків не вимагають застосування процесів очистки даних для їх стандартизації. Єдине, що необхідно здійснити для підготовки до аналізу даних цих вимірів – закастити їх у відповідні типи даних Python, що, для категористичних показників буде string, для цілочисельних int, а для дат – datetime.

**4.ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ**

**4.1 Обґрунтування вибору методів інтелектуального аналізу даних**

З-поміж категористичних даних для класифікації для досягення мети роботи слід виділити дані про належність книги певній колекції, які представлені у обраній базі даних під хедером 'ItemCollection'. Даний вимір обраний як ціль класифікації з кількох причин:

1. Він чітко визначений для кожної книги за її цільовою аудиторією та тематикою.

2. Кожна книга може мати тільки одну колекцію, якій вона належить.

3. Автоматизація процесу віднесення книги до певної колекції покращує процеси прийому книг в бібліотеці на облік і, таким чином, приносить суспільну користь та може бути предметом для подальших досліджень.

Здійснимо аналіз вибірки за стовпчиком колекції книги.

books['ItemCollection'] = books['ItemCollection'].astype('string')  
books['ItemCollection'] = books['ItemCollection'].str.strip()  
books['ItemCollection'].value\_counts()

Зображення, що містить текст, знімок екрана, Графік, схема

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Бачимо, що вибірка не дуже добре збалансована стосовно колекцій, які в неї входять і це буде потрібно врахувати при здійсненні класифікації. Якщо виділити класифікацію за колекціями та кластеризацію як основні цілі аналізу даних в цій вибірці, ймовірно, з більшою кількістю числових показників ми могли б досягнути більшої точності. Такими числовими показниками можуть стати, наприклад, кількість сторінок у книзі або навіть оцінка книги на певному сервісі. Ми можемо видобути такі показники з інших баз даних або відкритих джерел здійснюючи пошук за ISBN.

В ролі зовнішнього джерела даних використаний набір даних з сайту goodreads, який можна знайти за посиланням https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m . З нього ми збиратимемо лише найцікавіші для нас дані - ISBN, кількість сторінок і рейтинг. Дані були розбиті на 23 файли csv, востаннє оновлювались 3 роки тому. Свіжі датасети з goodreads зараз загалом досить складно знайти, оскільки дана організація припинила надавати API ключі з 2020 року і вже тривалий час збільшує кількість перешкод на шляху до доступу до своїх даних, що, в тому числі, проявляється у перешкодах парсингу вказаного сайту. Враховуючи те, що більшість даних з бібліотечної вибірки, мають дату надходження примірників 2020 року, даний датасет з goodreads містить найактуальніші дані на момент отримання даних книг бібліотечним фондом. [4]

def get\_additional\_info\_goodreads(csv\_name):  
 gr\_info = pd.DataFrame()  
 columns\_to\_extract = {  
 'ISBN' : str,  
 'pagesNumber' : int,  
 'Rating' : float  
 }  
 gr\_info = pd.read\_csv(csv\_name, usecols=columns\_to\_extract.keys(), dtype=columns\_to\_extract)  
 return gr\_info

def get\_additional\_info\_goodreads1(csv\_name):  
 gr\_info = pd.DataFrame()  
 columns\_to\_extract = {  
 'ISBN' : str,  
 'PagesNumber' : int,  
 'Rating' : float  
 }  
 gr\_info = pd.read\_csv(csv\_name, usecols=columns\_to\_extract.keys(), dtype=columns\_to\_extract)  
 return gr\_info

goodreads\_books\_add = pd.DataFrame()

goodreads\_datasets = ['book1-100k.csv', 'book100k-200k.csv', 'book200k-300k.csv', 'book300k-400k.csv', 'book400k-500k.csv', 'book500k-600k.csv', 'book600k-700k.csv', 'book700k-800k.csv', 'book800k-900k.csv', 'book900k-1000k.csv', 'book1000k-1100k.csv', 'book1100k-1200k.csv', 'book1200k-1300k.csv', 'book1300k-1400k.csv', 'book1400k-1500k.csv', 'book1500k-1600k.csv', 'book1600k-1700k.csv', 'book1700k-1800k.csv']  
goodreads\_datasets\_other =['book1800k-1900k.csv', 'book1900k-2000k.csv', 'book2000k-3000k.csv', 'book3000k-4000k.csv', 'book4000k-5000k.csv']

for b in goodreads\_datasets:  
 goodreads\_books\_add = pd.concat([goodreads\_books\_add, get\_additional\_info\_goodreads(b)], axis=0)

goodreads\_books\_add = goodreads\_books\_add.rename(columns={'pagesNumber' : 'PagesNumber'})

for b in goodreads\_datasets\_other:  
 goodreads\_books\_add = pd.concat([goodreads\_books\_add, get\_additional\_info\_goodreads1(b)], axis=0)

goodreads\_books\_add\_clean.to\_csv('goodreads\_data.csv', index=False)

Вищевказаний код послідовно зчитує всі дані про ISBN, кількість сторінок і усереднену користувацьку оцінку з вказаного датасету. Отримані дані виводяться у зовнішній файл для полегшення подальшого доступу. Поєднуємо дані із основним джерелом.

merged\_books = pd.DataFrame()

merged\_books = pd.merge(books, goodreads\_from\_csv, left\_on='ISBN10', right\_on='ISBN', how='left')

full\_data\_books = pd.DataFrame()  
full\_data\_books = merged\_books.loc[merged\_books['ISBN'].notna()]

Бачимо, що із додаткового джерела нам вдалося отримати додаткові дані лише для 15% всієї вибірки. Можна здійснити кластеризацію та класифікацію на всіх даних вибірки і порівняти точність з класифікацією та кластеризацією на даних, для яких є додаткова інформація про кількість сторінок у книзі та про її користувацький рейтинг на goodreads. Крім цього цікавим експериментом може бути порівняння роботи різних алгоритмів за умови використання тільки заголовка та заголовка в поєднанні з підзаголовком.

**4.2 K-means кластеризація**

Спробуємо здійснити кластеризацію за допомогою K-means алгоритму на даній вибірці без врахування підзаголовку і тем.

full\_data\_books['Author'] = full\_data\_books['Author'].fillna('Unknown')

features = full\_data\_books[['Author\_Label', 'Publisher\_Label', 'ItemType\_Label', 'ItemCollection\_Label', 'ItemLocation\_Label', 'Report\_Year', 'Report\_Month', 'Report\_Day', 'PublicationYear', 'PagesNumber', 'Rating']]

scaler = StandardScaler()  
features\_scaled = scaler.fit\_transform(features)

kmeans = KMeans(n\_clusters=5, random\_state=42)  
full\_data\_books['KMeans\_Cluster'] = kmeans.fit\_predict(features\_scaled)

Графічно отримані кластери можна зобразити як набір звичайних графіків на різних вимірах або скористатися PCA для зменшення вимірності. Наприклад, розглянемо як виглядає кластеризація для вимірів видавництва і бібліотечної колекції, відносно вимірів типу книги та локації та відносно вимірів оцінки та кількості сторінок.

Зображення, що містить знімок екрана, фіолетовий, фіалка

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Мультимедійне програмне забезпечення, Графічний редактор

Автоматично згенерований опис

Спробуємо зменшити кількість вимірів із застосуванням PCA. Можна навіть порівняти із якимось із попередньо отриманих вимірів, наприклад, для наявних числових даних.

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)  
features\_pca = pca.fit\_transform(features\_scaled)

full\_data\_books['PCA1\_KMeans'] = features\_pca[:, 0]  
full\_data\_books['PCA2\_KMeans'] = features\_pca[:, 1]

plt.subplot(1, 2, 1)  
plt.scatter(full\_data\_books['PCA1\_KMeans'], full\_data\_books['PCA2\_KMeans'], c=full\_data\_books['KMeans\_Cluster'], cmap='viridis', edgecolor='k', s=50)  
plt.title('Scatter Plot of PCA Components')  
  
plt.subplot(1, 2, 2)  
plt.scatter(full\_data\_books['PublicationYear'], full\_data\_books['PagesNumber'], c=full\_data\_books['KMeans\_Cluster'], cmap='viridis', edgecolor='k', s=50)  
plt.title('Scatter Plot of Numerical Features')  
  
plt.tight\_layout()  
plt.show()

Зображення, що містить текст, знімок екрана, Графічний редактор

Автоматично згенерований опис

**4.3 Agglomerative кластеризація**

Кластеризація для цього набору даних може бути здійснена і за іншими алгоритмами, наприклад, цікавим може виявитись якогось роду ієрархічна кластеризація з врахуванням текстових стовпчиків як, власне, назва або певним чином опрацьований список тем.

Оберемо алгоритм агломеративного кластерінгу. Може бути цікавим розглянути кілька варіантів стосовно вхідних даних:

1. Брудний заголовок книги без підзаголовку.

2. Чистий заголовок книги без підзаголовку.

3. Повний брудний заголовок книги.

4. Повний чистий заголовок книги.

Під "брудним" заголовком розуміємо такий, що не пройшов обробку з видалення стоп-слів та лематизації. Враховуючи малу кількість слів у заголовку книг такі забруднені дані можуть також дати цікавий результат, який достатньо точно описує дійсність.

Слід додати нові стовпчики у вибірку, які позначатимуть чистий загловок без підзаголовку, чистий заголовок із та брудний заголовок з підзаголовком (без помітної точки розділення).

from sklearn.cluster import AgglomerativeClustering  
from scipy.cluster.hierarchy import dendrogram, linkage

full\_data\_books['Title\_Clean'] = full\_data\_books['Title'].apply(full\_clean\_text)  
full\_data\_books['Title\_Subtitle'] = full\_data\_books['Title'] + ' ' + full\_data\_books['Subtitle'].fillna('')  
full\_data\_books['Title\_Subtitle\_Clean'] = full\_data\_books['Title\_Subtitle'].apply(full\_clean\_text)

from sklearn.compose import ColumnTransformer  
from sklearn.pipeline import Pipeline

numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating', 'Report\_Year']  
categorical\_cols = ['Author', 'Publisher', 'ItemType', 'ItemCollection', 'ItemLocation']  
text\_cols = ['Title']

numerical\_data = full\_data\_books[numerical\_cols].copy()  
categorical\_data = full\_data\_books[categorical\_cols].copy()  
text\_data = full\_data\_books[text\_cols].copy()

from sklearn.base import TransformerMixin  
class DendrogramTransformer(TransformerMixin):  
 def fit(self, X, y=None):  
 return self  
  
 def transform(self, X):  
 return X.toarray()

preprocessor = ColumnTransformer( transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(), categorical\_cols),  
 ('text', TfidfVectorizer(), 'Title')  
])

pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('to\_dense', DendrogramTransformer()),  
 ('agglo', AgglomerativeClustering(n\_clusters=5))  
])

full\_data\_books['Agglo\_Cluster\_TD'] = pipeline.fit\_predict(full\_data\_books[numerical\_cols + categorical\_cols + text\_cols])

Повторимо даний процес для забруднених заголовків із забрудненими підзаголовками та для очищених даних із заголовком та без нього.

Зображення, що містить знімок екрана, дизайн

Автоматично згенерований опис

Зображення, що містить знімок екрана, дизайн, мистецтво

Автоматично згенерований описЗображення, що містить знімок екрана, текст, дизайн, мистецтво

Автоматично згенерований описЗображення, що містить знімок екрана, дизайн, мистецтво

Автоматично згенерований опис

Чисто візуально можна помітити, що зміни в очищенні даних для назви книги вносять на даній вибірці для Агломеративного алгоритму кластеризації відносно незначні зміни у зовнішній вигляд ієрархії кластерів. Проте, слід зазначити, що весь процес кластеризації для очищених даних заголовків і підзаголовків, згідно власних обрахунків PyCharm, на комп'ютері, на якому була виконана дана робота, займає в середньому на 1 секунду менше (приблизно на 6% швидше). Гіпотеза щодо того, що на очищених даних даний алгоритм кластеризації працюватиме швидше не суперечить здоровому глузду і може бути розглянута в подальших дослідженнях для різних вибірок і більшого об'єму даних.

**4.4 Класифікація Random Forest на доповнених даних**

Здійснимо класифікацію книг за допомогою методу Random Forest, який, побудований на використанні множини дерев вибору в процесі навчання. Дослідимо точність з якою на даній вибірці можливо здійснити передбачення колекції, до якої належить книга за її іншими ознаками. Спочатку здійснимо експеримент ігноруючи назву книги.

from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

X = full\_data\_books[['Author', 'Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation']]  
y = full\_data\_books['ItemCollection']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=12)

numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Author', 'Publisher', 'ItemType', 'ItemLocation']

preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols)  
 ]  
)

pipeline = Pipeline(  
 steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', RandomForestClassifier(random\_state=3))  
 ]  
)

pipeline.fit(X\_train, y\_train)

y\_pred = pipeline.predict(X\_test)

report = classification\_report(y\_test, y\_pred)

В результаті отримана точність 0.62. Бачимо, що виникає проблема, пов'язана із надмірною кількістю категористичних даних відносно вибірки. У тренувальному наборі моделі банально відсутня значна кількість категорій, які потім з'являються в тестовому наборі. Таким чином, ми в середньому отримуємо досить погане передбачення. Висуваємо гіпотезу, що найбільше проблем виникає в зв'язку з надмірним різноманіттям авторів. Перевіримо дану гіпотезу, здійснивши класифікацію без врахування даних про автора.

X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=18)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']

…

Отримана точність 0.63. Хоча виграш незначний, ми експериментально знайшли підтвердження гіпотези, що за відсутності даних про авторів, Random Forest може дати більшу точність передбачення для даної вибірки.

Проведений ще один експеримент, який виявив, що додавання параметра class\_weight='balanced' для моделі Random forest абсолютно ніяк не вплинула на точність класифікації. Це не означає, що проблеми із незбалансованістю даних не існує, але чітко демонструє, що запропоноване рішення не підходить для даної задачі. Питання стосовно збалансування даної вибірки може бути предметом для подальших досліджень.

Можна висунути гіпотезу, що попри труднощі класифікації, що виникають в зв'язку із незбалансованістю вибірки, ми можемо покращити точність класифікації, додавши дані стосовно слів, що використовуються у повній назві книги. Власне, в таком разі нульовою гіпотезою буде "класифікація з додаванням інформації про назву книги не буде відрізнятися за точністю від класифікації без даних про назву книги".

X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation', 'Title\_Subtitle\_Clean']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=18)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
text\_cols = ['Title\_Subtitle\_Clean']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols),  
 ('text', TfidfVectorizer(), 'Title\_Subtitle\_Clean')  
 ]  
)  
  
pipeline = Pipeline(  
 steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', RandomForestClassifier(random\_state=3, class\_weight='balanced'))  
 ]  
)  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Отримана точність 0.66. Спостерігаємо невелике покращення показників класифікації. Вважаємо, що цього покращення достатньо, щоб відкинути нульову гіпотезу і стверджувати, що класифікація з використанням моделі Random forest дає кращі результати за умови "згодовування" їй даних про назву книги.

**4.5 Класифікація за допомогою SVC на доповнених даних**

Здійснимо класифікацю книг за допомогою Support Vector Machines алгоритмів. Даний метод був обраний для цієї вибірки через її відносно велику вимірність і здатність цих алгоритмів ефективно справлятися із проблемами, які в таких вибірках виникають. Спочатку перевіримо точність моделі за умови використання нею тільки категористичних та цифрових даних, а далі порівняємо отримані результати із результатами за умови додавання даних про назву книги. Для даного експерименту інформацію про автора враховувати в обох випадках не будемо.

from sklearn.svm import SVC

X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols)  
 ]  
)  
  
pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', SVC(kernel='linear', C=1, class\_weight='balanced', random\_state=42))  
])  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Отримана точність класифікації 0.41. Бачимо, що чисто на нетекстових даних в межах нашої вибірки, на відміну від наших очікувань, точність класифікації з SVC помітно нижча, ніж для Random forest. Порівняємо із точністю класифікації для даних з додаванням інформації про назву книги.

X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation', 'Title\_Subtitle\_Clean']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=7)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
text\_cols = ['Title\_Subtitle\_Clean']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols),  
 ('text', TfidfVectorizer(), 'Title\_Subtitle\_Clean')  
 ]  
)  
  
pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', SVC(kernel='linear', C=1, class\_weight='balanced', random\_state=42))  
])  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Отримана точність 0.50. З додаванням текстових даних SVC дає дещо кращий результат, проте все ще гірший, ніж відповідні результати для Random forest.

**4.6 Класифікація з Gradient Boosting на доповнених даних**

Gradient boosting використовує комбінацію слабких дерев вибору для здійснення задач класифікації. Вважається використання даного алгоритму дає в загальному кращі результати, ніж Random forest. Приймемо це як нашу гіпотезу для даного експерименту і, аналогічно із попередніми алгоритмами, здійснимо класифікацію для даних без використання назви книги та з і порівняємо отримані результати між собою.

from sklearn.ensemble import GradientBoostingClassifier

X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=13)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols)  
 ]  
)  
  
pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', GradientBoostingClassifier(random\_state=7))  
])  
  
pipeline.fit(X\_train, y\_train)  
  
y\_pred = pipeline.predict(X\_test)

Отримана точність 0.54. Цікаво, що на даному етапі точність нижча, ніж для Random forest. Додамо дані про назву книги.

#%%  
X = full\_data\_books[['Publisher', 'PublicationYear', 'PagesNumber', 'Rating', 'ItemType', 'ItemLocation', 'Title\_Subtitle\_Clean']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=13)  
  
numerical\_cols = ['PublicationYear', 'PagesNumber', 'Rating']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
text\_cols = ['Title\_Subtitle\_Clean']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols),  
 ('text', TfidfVectorizer(), 'Title\_Subtitle\_Clean')  
 ]  
)  
  
pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', GradientBoostingClassifier(random\_state=9))  
])  
  
pipeline.fit(X\_train, y\_train)  
  
y\_pred = pipeline.predict(X\_test)

Отримана точність 0.53. Висловлена гіпотеза не підтвердилась. На даному наборі даних Gradient Boosting здійснює класифікацію з нижчою точністю, ніж Random Forest.

**4.7 Класифікація вибірки з малою кількістю числових даних із використанням заголовка**

Як можна було побачити в огляді алгоритмів класифікації на вибірці, для якої з обраного, зовнішнього відносно базового набору даних, джерела були додані дані про кількість сторінок та користувацьку оцінку, значну роль в точності передбачення відіграють:

1. Велика відносно розміру вибірки кількість категорій, на яких здійснюється класифікація - результати роботи стають менш точними, оскільки певні класи можуть не потрапити до тренувального набору даних або взагалі бути присутніми тільки один раз на всю вибірку

2. Додавання інших типів даних для класифікації окрім числових та категористичних - для всіх алгоритмів спостерігалося підвищення точності при додаванні текстових даних заголовку книги як вхідних для класифікації.

Таким чином, можна припустити, що хоча числові дані безумовно важливі, але, можливо, ми можемо здійснювати класифікацію ефективно і на базовій вибірці даних. Висунемо гіпотезу про те, що точність класифікації на даній вибірці виграє від збільшення її розміру більше, ніж від збільшення її повноти за рахунок зовнішніх даних.

Random forest:

X = full\_data\_books[['Publisher', 'PublicationYear', 'ItemType', 'ItemLocation', 'Title\_Subtitle\_Clean']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=19)  
  
numerical\_cols = ['PublicationYear']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
text\_cols = ['Title\_Subtitle\_Clean']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols),  
 ('text', TfidfVectorizer(), 'Title\_Subtitle\_Clean')  
 ]  
)  
  
pipeline = Pipeline(  
 steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', RandomForestClassifier(random\_state=4))  
 ]  
)  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Точність – 0.58.

SVC:

pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', SVC(kernel='linear', C=1, class\_weight='balanced', random\_state=17))  
])  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Точність – 0.54.

Gradient boosting:

pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', GradientBoostingClassifier(random\_state=11))  
])  
  
pipeline.fit(X\_train, y\_train)  
  
y\_pred = pipeline.predict(X\_test)

Точність – 0.49.

Легко помітити, що для даних, які не були збагачені, тільки SVC алгоритм демонструє вищу якість класифікації, ніж це спостерігалося у збагачених даних.

**4.8 Класифікація вибірки з малою кількістю числових даних без використання заголовка**

Random forest:

X = full\_data\_books[['Publisher', 'PublicationYear', 'ItemType', 'ItemLocation']]  
y = full\_data\_books['ItemCollection']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=19)  
  
numerical\_cols = ['PublicationYear']  
categorical\_cols = ['Publisher', 'ItemType', 'ItemLocation']  
  
preprocessor = ColumnTransformer(  
 transformers=[  
 ('num', StandardScaler(), numerical\_cols),  
 ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_cols),  
 ]  
)  
  
pipeline = Pipeline(  
 steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', RandomForestClassifier(random\_state=4))  
 ]  
)  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Точність – 0.52.

SVC:

pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', SVC(kernel='linear', C=1, class\_weight='balanced', random\_state=17))  
])  
  
pipeline.fit(X\_train, y\_train)  
y\_pred = pipeline.predict(X\_test)

Точність – 0.43.

Gradient boosting:

pipeline = Pipeline(steps=[  
 ('preprocessor', preprocessor),  
 ('classifier', GradientBoostingClassifier(random\_state=11))  
])  
  
pipeline.fit(X\_train, y\_train)  
  
y\_pred = pipeline.predict(X\_test)

Точність – 0.52.

**4.9 Загальне порівняння використаних алгоритмів класифікації для різних наборів вхідних даних**

Можна скласти таблицю, яка наочно демонструє отриману точність для різних варіацій вибірок і різних алгоритмів класифікації.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1604 + NumPg + Rat - Title | 1604 + NumPg + Rat + Title | 10250 – NumPg – Rat - Title | 10250 – NumPg – Rat + Title |
| Random forest | 0.62 | 0.63 | 0.52 | 0.58 |
| SVC | 0.41 | 0.50 | 0.43 | 0.54 |
| Gradient boosting | 0.54 | 0.53 | 0.52 | 0.49 |

**ВИСНОВКИ**

В даній роботі була здійснена класифікація та кластеризація книг на прикладі бібліотечного фонду міста Сієтл.

При дослідженні та очищенні даних із вибірки книг стовпчик видавців вимагав свого роду нормалізації. Навіть проста K-means кластеризація за ортографічною відстанню давала дуже хороші результати, але потребувала людської супервізії в плані визначення кластерів-сміття. Ймовірно, за умов масштабування варто обрати більш вишуканий алгоритм кластеризації. В цьому плані перспективним виглядяє DBSCAN і для подальших досліджень варто оцінити його можливості щодо виконання поставлених задач.

Для збагачення даних додатковими параметрами в роботі використовувалось лише одне зовнішнє джерело. З цього джерела відповідників вдалося знайти лише приблизно для 16% вибірки. Для більш детальних досліджень цього явно замало, а викидати 84% даних є нераціональним рішенням. Для покриття більшого об'єму вибірки слід розглянути способи поєднувати більшу кількість джерел даних між собою та систему визначення істини у випадку конфліктів між джерелами. Хорошим рішенням може також стати використання API, наприклад, GoogleBooks API. Це дозволить запитувати доступ тільки до необхідних додаткових даних на противагу завантаження в пам'ять велитенських баз даних, з яких для нас можуть бути корисними лише кілька записів.

Порівняння класифікації для різних наборів типів вхідних даних дала нам можливість зрозуміти, що використання назви книги, навіть у найпримітивнішому форматі без використання спеціалізованих алгоритмів для аналізу текстових даних, для класифікації спільно з іншими даними про книги для даної вибірки зазвичай покращує точність алгоритмів класифікації. Це стосується як даних, для яких були додані поля із зовнішніх джерел, так і даних з базової вибірки. Найбільшої точності у передбаченні колекції до якої належить книга за іншими даними про неї вдалося досягти з використанням алгоритму Random forest.

Для подальших досліджень в предметній області книг з використанням обраного основного джерела доречно буде звернути увагу на алгоритми класифікації та кластеризації які більшою мірою заточені на обробку мови. [5]

**ПЕРЕЛІК ПОСИЛАНЬ**

1. <https://lithub.com/how-many-books-will-you-read-before-you-die/>
2. <https://www.isbn-international.org>
3. <https://www.diva-portal.org/smash/get/diva2:1194266/FULLTEXT01.pdf>
4. <https://www.goodreads.com/api>
5. <https://towardsdatascience.com/nlp-with-lda-latent-dirichlet-allocation-and-text-clustering-to-improve-classification-97688c23d98>