КУРСОВ ПРОЕКТ

ПО СИСТЕМИ, ОСНОВАНИ НА ЗНАНИЯ

Тема:

Система за генериране на препоръка за закупуване на книги (recommender system)

Студенти:

Иван Иванов Петров, 3MI0700035

Петя Ангелова Личева, 3MI0700022

София, януари 2025 г.

Съдържание

[1. Формулировка на задачата 2](#_Toc5681)

[2. Използвани данни 4](#_Toc29353)

[3. Използвани алгоритми 6](#_Toc1865)

[4. Описание на програмната реализация 15](#_Toc30144)

[5. Примери, илюстриращи работата на програмната система 16](#_Toc26114)

[6. Литература 16](#_Toc24632)

### Формулировка на задачата

Система за препоръки на книги, като препоръките се базират на рейтинг от читателите/потребителите. На данните би могло да се приложи ре-филтрация по възраст на читателите. Основен проблем, който би следвало да се реши преди ре-филтрацията е, че около 30% от читателите не са посочили възрастта си, което води до нужда от поправки/зачистване в данните преди ре-филтрацията, което на по-късен етап установихме, че води до загуба на информация и неточности в модела. Естествено, в следната документация предлагаме пълния процес, през който преминахме в изграждането на система за препоръки на книги.

Забележка: Преди да дсотигнем крайната реализация на задачата си преминахме през няколко различни потенциални подхода за решаване на подобна задача, но основен подход, на който се спряхме, за реализация на задачата ни е k-NN алгоритъма във варианта му на a-NN (k-NN алгоритъм с приближение). За да изведем съображенията, които ни насочиха да тръгнем в тази посока би било редно да опишем подходите и експериментите, които проведохме, заеднос техните резултати и изводи, както и да покажем с какво a-NN подхода превъзхожда всички останали, изпробвани алгоритми в условностите на задачата ни. Повече информация за процеса на имплементация, както и за преценката за нужния ни алгоритъм може да намерите в **графа Използвани алгоритми** на документацията.

Забележка: Кодът е разработен на Python със съответно необходимите му библиотеки като:

#### **Numpy**

**Numpy** е библиотека за научни изчисления с Python, която предоставя поддръжка за работа с многомерни масиви и набор от математически функции за бързи операции с тези масиви. В кода **Numpy** се използва за:

* Преобразуване на колони в масиви (np.array()), които се използват като входни данни за изчисления и модели.
* Комбиниране на колони в масиви (np.column\_stack()) за създаване на матрици, използвани като характеристики (features) в модела.

#### **pandas**

**pandas** е библиотека за манипулация и анализ на данни. Тя предлага структури от данни като DataFrame, които улесняват работата с таблични данни. В кода **pandas** се използва за:

* Зареждане на CSV файлове с информация за книги, потребители и рейтинги (pd.read\_csv()).
* Обединяване на данни от различни източници (pd.merge()).
* Филтриране и попълване на липсващи стойности в данните (dropna() и fillna()).

#### **sklearn.neighbors**

Модулът **sklearn.neighbors** предоставя алгоритми за машинно обучение на базата на съседи на текущо тествания пример. В кода е използван класът **KNeighborsRegressor**, който:

* Използва метода на най-близките съседи за регресия.
* Предсказва стойност на база сходството (напр. близост по рейтинг на книга и възраст на читател).

#### **sklearn.metrics**

Модулът **sklearn.metrics** предлага функции за оценка на качеството на машинно обучение. В кода се използват:

* mean\_absolute\_error и mean\_squared\_error за изчисляване на грешките в модела.
* Допълнителни функции за оценка като precision\_at\_k и recall\_at\_k се имплементират ръчно, за да измерват качеството на препоръките.

#### **sklearn.model\_selection**

Модулът **sklearn.model\_selection** осигурява инструменти за разделяне на данни и валидиране на модели. В кода **train\_test\_split**:

* Разделя данните на тренировъчен и тестов набор, което гарантира, че моделът се тренира върху тренировъчното множество данни и се оценява коректно.

#### **scipy.stats**

Модулът **scipy.stats** предоставя статистически функции и разпределения. В кода се използва **pearsonr:**

* За изчисляване на коефициента на корелация (по метода на Пиърсън) между рейтингите на книги и възрастта на читателите.
* Това помага да се оцени дали съществува линейна връзка между тези две променливи.

#### **warnings**

Модулът **warnings** е предназначен за управление на warning-ите в Python. В кода **warnings.filterwarnings('ignore')**:

* Заглушава предупрежденията, които биха могли да разсейват при анализа на резултатите или изпълнението на кода.

Тези библиотеки съвместно осигуряват основата за предварителна обработка, статистически анализ, машинно обучение и оценка на резултатите.

### Използвани данни

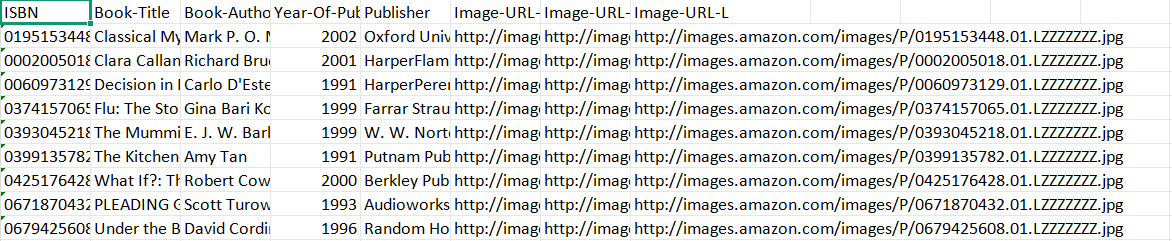
Описаните по-долу **CSV таблици** са взети от **dataset-а в** <https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset,> а този ресурс е предоставен и в **графа Литература** на документацията **под наименованието Сетове с данни**.

#### Users

#### Users

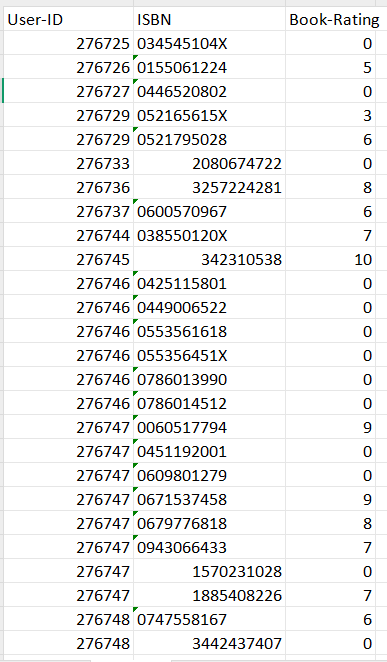
Таблица, която съдържа информация за потребителите, наричани още читатели в рамките на нашата система. Информацията, която се съхранява за читателите е техните уникални идентификационни номера (User-ID), локациите им (Location) и възрастта им. Потребителските идентификатори са анонимизирани и съпоставими с цели числа. Данните за местоположението на читателите и тяхната възраст се предоставят, ако има такива. В противен случай тези полета приемат NULL стойности. Това предполага, че ако те бъдат включвани в методите за категоризация и препоръки на книги на други потребители, трябва да се подсигурим за коректността на данните и евентуално да ги почистим от NULL стойности, тъй като те не внасят никаква информация и пречат на процеса по препоръка на книги.

#### Books



Таблица, която предоставя информация за книгите, от които имаме в наличност и ще можем да препоръчваме на нашите читатели. Книгите са уникално идентифицируеми от техните ISBN номера като трябва предварително да се погрижим, че всички ISBN номера са уникални и валидни. Таблицата също така предоставя информация за съдържанието на книгата като загавие на книгата (Book-Title), автор на книгата (Book-Author), година на публикация на книгата (Year-Of-Publication) и издателство на книгата (Publisher). **Тази информация е предоставена от Amazon Web Services.** Трябва да се отбележи също така, че при наличие на съавторство над някоя книга, само имената на първият автор са предоставени. В таблицата също така се съхранява информация за линка към корицата на съответната книга (линк към уеб сайта на Amazon). Тези линкове може да се показват под различни форми (Image-URL-S или къси, Image-URL-M или средно дълги, Image-URL-L или дълги).

1. Ratings

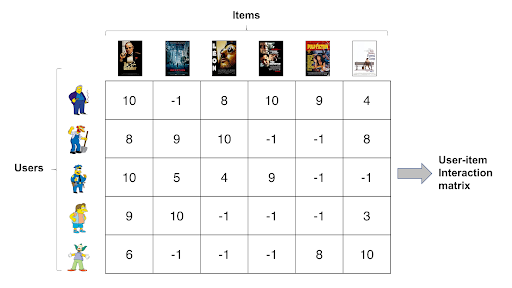


Таблица, която описва оценките на книгите. В нея се съдържа информация кой потребител (User-ID) коя книг (ISBN) а е оценил и каква е била самата оценка (число в интервала от 1 до 10, вписана в полето Book-Rating). Ако някоя книга все още не е била оценена от даден потребител, оценката, която автоматично се вписва в полето за оценка (Book-Rating) е 0.

### Използвани алгоритми

За решаването на задачата сме изолзвали няколко алгоритъма - разновидности на k-NN алгоритъма с цел проучване на потенциални възможности за модела и избиране на най-добрия възможен в конкретиката на задачата ни и dataset-овете, описани по - горе. В следващите редове ще опишем всеки един от изпробваните модели, заедно с проблемите, които сме срещнали при тяхната имплементация, възможните им решения и скрийншоти, показващи нагледно тяхната работа.

#### k-NN алгоритъм



##### Описание на алгоритъма

k-NN алгоритъма е класификационен алгоритъм, който се базира на изчисления на разстоянията между текущо изследвания тестови пример и неговите k на брой най-близки съседи, така че да се открият най-добрите примери от някаква извадка данни и според тази класификация, те да бъдат препоръчани на потребителя на системата (в случая говорим за книги, които трябва да се препоръчат на определена група читатели със сходства помежду си). За тази цел ни трябва определена метрика, която често пъти е Евклидово или Мнахатанско разстояние между дадените тестови примери и техните най-близки съседи (за простота).

* Евклидово разстояние: Ако точката ( A(x\_1, y\_1) ) и точката ( B(x\_2, y\_2) ), то
* Манхатанско разстояние: Ако точката ( A(x\_1, y\_1) ) и точката ( B(x\_2, y\_2) ), то

Тези метрики обаче не са достатъчно добри за нашата задача, защото те допускат дори и вземането на всички примери (в случая книги), които са на горе-долу еднакво разстояние от текущо тествания, а **cousine метриката** - взема предвид и ъгъла, под който се намира съседа на текущо тествания пример спрямо текущо тествания пример и въз основа на това колко фолям е ъгълът решава кой съсед (в случая книга) да добави в препоръчаните.

##### Алгоритъмът в стъпки

1. **Зареждане и обработка на данни**

* Зареждане на данните от файлове Ratings.csv, Books.csv и Users.csv в **Pandas DataFrame**.
* Обработка на липсващи стойности:
* Попълване на липсващи стойности с фиксирани стойности като "NaN" или -1.
* Изтриване на редове с липсващи стойности в оценките (df\_ratings.dropna()).

1. Комбиниране на данни

* Обединяване на данни за оценки (df\_ratings) и книги (df\_books) чрез общата колона ISBN.
* Премахване на ненужни колони, като например Book-Author.

1. Изчисляване на популярност на книгите

* Групиране на данни, за да се преброи броят на рейтингите за всяка книга.
* Обединяване на тези резултати с оригиналния набор от данни.

1. Филтриране на популярни книги

* Изчисляване на прагово ниво за популярност на книгите въз основа на 90-ия перцентил от броя на рейтингите.

1. Създаване на матрица за препоръки

* Преобразуване на данните в **pivot таблица**, където редовете са заглавия на книги, колоните са потребителски идентификатори, а стойностите са рейтинги.
* Преобразуване на тази матрица в рядко представяне (**CSR matrix**), за да се спести памет.

1. Създаване и обучение на модел

* Инициализиране на модел **NearestNeighbors** с метрична функция "косинусова подобие".
* Обучение на модела с данните от CSR матрицата.

1. Препоръчване на книги

* Дефиниране на функцията get\_recommends, която приема заглавие на книга като вход, изчислява най-близките съседи на тази книга, и връща списък от препоръчани книги.
* Използване на функцията за препоръки.

##### Алгоритъмът в код

import pandas as pd

import numpy as np

from pprint import pprint as pp

from sklearn.neighbors import NearestNeighbors

from scipy.sparse import csr\_matrix

# This code snippet is reading data from CSV files into pandas DataFrames. Here's a breakdown of what

# each line is doing:

df\_ratings = pd.read\_csv("src/data/Ratings.csv", na\_values=["null", "nan", ""])

df\_books = pd.read\_csv(

    "src/data/Books.csv",

    na\_values=["null", "nan", ""],

    usecols=["ISBN", "Book-Title", "Book-Author"],

)

df\_users = pd.read\_csv("src/data/Users.csv", na\_values=["null", "nan", ""])

df\_books = df\_books.fillna("NaN")

df\_ratings = df\_ratings.dropna()

df\_users = df\_users.fillna(-1)

combine\_book\_ratings = pd.merge(df\_ratings, df\_books, on="ISBN")

combine\_book\_ratings = combine\_book\_ratings.drop(["Book-Author"], axis="columns")

book\_rating\_count = (

    combine\_book\_ratings.groupby(by=["ISBN"])["Book-Rating"]

    .count()

    .reset\_index()

    .rename(columns={"Book-Rating": "RatingCount"})

)[["ISBN", "RatingCount"]]

book\_rating\_with\_total\_count = combine\_book\_ratings.merge(

    book\_rating\_count, on=["ISBN"], how="left"

)

pp(book\_rating\_with\_total\_count["RatingCount"].quantile(np.arange(0.9, 1, 0.01)))

# Top 10% of rating counts

popularity\_threshold = 136

rating\_popular\_books = book\_rating\_with\_total\_count.query(

    "RatingCount >= @popularity\_threshold"

)

pivot = (

    rating\_popular\_books.drop\_duplicates(["Book-Title", "User-ID"])

    .pivot(index="ISBN", columns="User-ID", values="Book-Rating")

    .fillna(0)

)

model\_knn = NearestNeighbors(metric="cosine", algorithm="auto")

model\_knn.fit(csr\_matrix(pivot.values))

def get\_recommends(isbn="", k\_neighbors=5):

    try:

        x = pivot.loc[isbn].array.reshape(1, -1)

        distances, indices = model\_knn.kneighbors(x, n\_neighbors=k\_neighbors)

        R\_books = []

        for distance, indice in zip(distances[0], indices[0]):

            if distance != 0:

                R\_book = combine\_book\_ratings[

                    combine\_book\_ratings["ISBN"] == pivot.index[indice]

                ]["Book-Title"].values[0]

                R\_books.append([R\_book, distance])

        recommended\_books = [isbn, R\_books[::-1]]

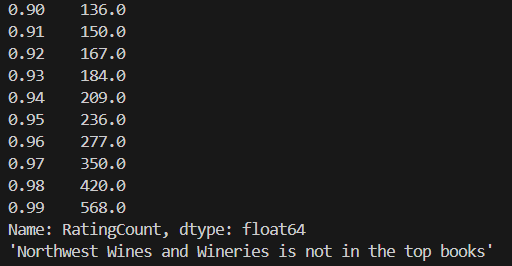
        return recommended\_books

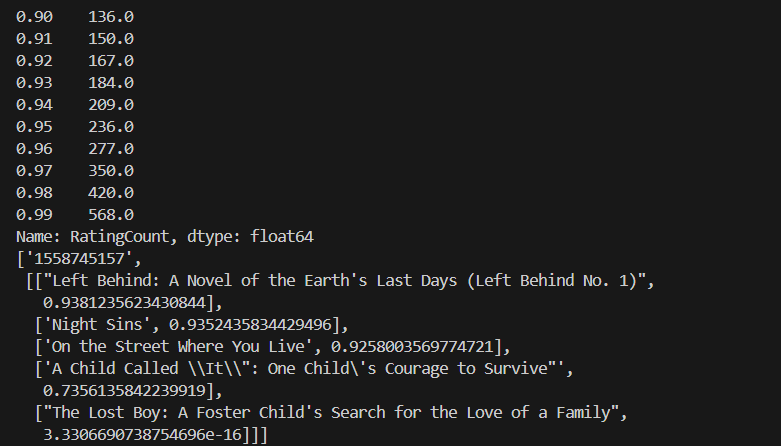
    except:

        return f"{isbn} is not in the top books"

pp(get\_recommends("1558745157"))

##### Резултати



На горния скрийншот са показани броят рейтинги на топ 10 най-четени книги и съобщение, че няма препоръчани книги. няма.

На горния скрийншот са показани броят рейтинги на топ 10 най-четени книги и препоръки за други книги, базирани на дадена от потребителя книга.

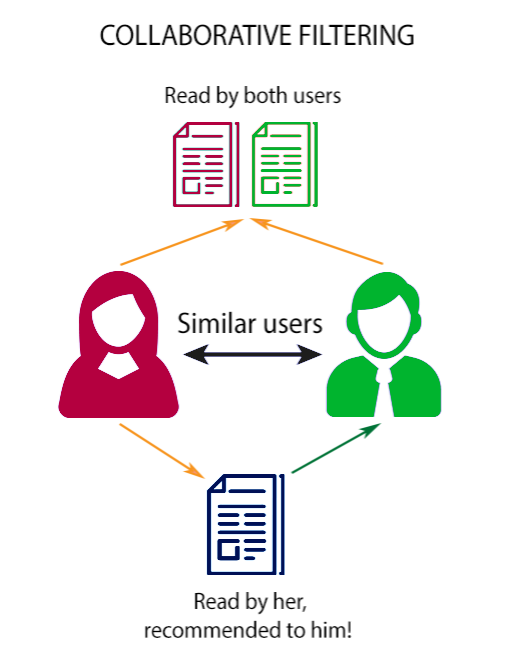
##### Изводи

При наблюдение на големи масиви от данни, каквито са предстваените във въпросните CSV таблици, алгоритъмът значително се забавя в изчисленията си и преопръките не са достатъчно персонализирани, за да можем да твърдим, че този модел е най-доброто решение. Това ни наведе на няколко оптимизационни идеи. На първо място решихме да използваме автоматичен подбор на структурата (на база колко мерни са наборите данни), която да използва k-NN модела, вместо възможностите:

* brute - brute-force алгоритъм, който изчислява разстоянията между всяка двойка обекти, за оптималност използва векторизирани изчисления и е ефективен за всякакъв вид метрика. Подходящ е при много малки или много големи обеми данни.
* ball-tree - балансирано дърво, което разделя пространството на сфери и е подходящо за работа с големи или високоразмерни набори от данни. Този алгоритъм е подходящ за Евклидова или друга подобна метрика.
* kd-tree - оптимизиран алгоритъм за работа с големи или високоразмерни набори от данни. Може да бъде по-бърз от другите алгоритми при тези размерности. Добре се справя с Евклидови и Манхатънски разстояния, но не работи добре с неквадратни метрични функции, като косинусово подобие.

На второ място, решихме да търсим начин за персонализирана препоръка на книги на база рейтингите на книгите и възрастовите групи, които се оформят като тенденции сред читателите, използващи нашата система. Естествено тук се породи въпроса: Ако досега можехме да представим алгоритъма в двумерна матрица, в която индексите на редовете са идентификационните номера на потребителите/читателите ни, а индексите на колоните са ISBN номерата на книгите, а стойностите в матрицата са оценките, които потребителите са давали за дадена книга, то сега как да изобразим положението, при което ще отправяме препоръки по 2 показателя? Отговорът, който намерихме беше свързан с представянето на линейната връзка между двата показателя, по които ще препоръчваме книги и така стигнахме до идеята за следващия ни експериментален алгоритъм.

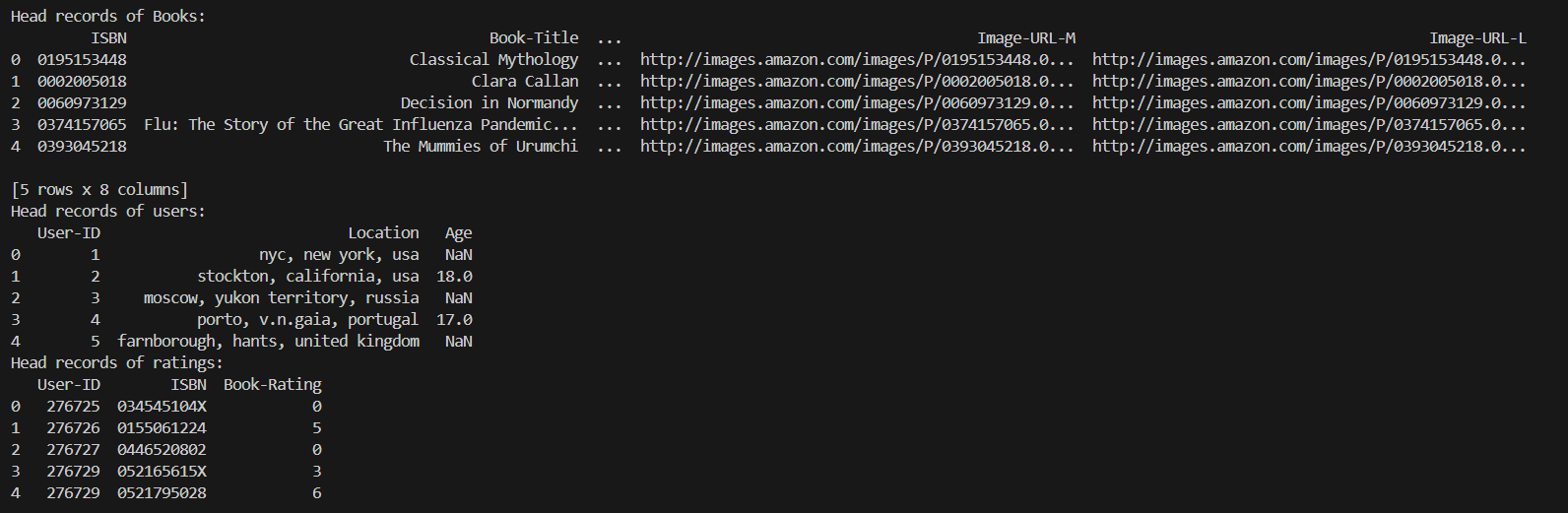
#### k-NN алгоритъм с регресорна препоръка на книги и грануларно изчисление на дистанциите (между текущо изследвания пример и неговите k на брой най-близки съседи)



##### Описание на алгоритъма

* Идеята зад този алгоритъм е да се използва колаборативно филтриране за препоръчване на книги на потребителите, като се вземат предвид техните оценки на книги и възраст. Това се постига чрез изчисляване на сходството между потребителите, като се използва Евклидово разстояние, което комбинира две характеристики: оценките на книгите и възрастта на потребителите. Целта е да се намерят потребители със сходни предпочитания (като се сравняват оценките им за книги) и да се препоръчат книги, които тези сходни потребители са харесали. Освен това, възрастта на потребителя е допълнителен фактор, който може да повлияе на препоръките, защото различни възрастови групи могат да имат различни литературни предпочитания. Алгоритъмът използва метода на най-близките съседи (k-NN), за да определи кои потребители имат най-сходни вкусове на базата на оценките на книги и възрастта им. Също така, чрез Евклидовото разстояние, се придава по-голямо значение на оценките на книгите, отколкото на възрастта, което отразява важността на самите предпочитания в сравнение с демографската информация. Идеята е, че чрез тази комбинирана оценка на сходствата, моделът може да предостави по - точни и персонализирани препоръки за книги на потребителите, които може да не са срещали, но са им сходни с други потребители, които имат подобни литературни вкусове. k-NN алгоритъм с линейна регресия за оценката влиянието на годините на читателите върху рейтинга на книгите и грануларни изчисления на разстоянията между текущо изследваната книга и нейните k на брой най-близки съседи чрез определена метрика (Евклидова, Манхатански разстояния, cousine и други) - стъпките са сходни като тези на горния алгоритъм, но с включвне на корелация между рейтингите и възрастите на читателите, регресия при положителна оценка за линейна връзка между двата компонента и грануларни изчисления, базирани на следната формула:

##### Скрийншот на препоръчаните книги за 2-ма читатели с изцяло различни характериситки:



##### Алгоритъмът в стъпки

1. Включване на необходимите библиотеки и пакети.
2. Зареждане на данните.
3. Обединяване на данните за рейтингите на книгите (всички данни от таблицата с рейтингите) с годините на читателите.
4. Почистване на данните от невалидни такива (NaN стойности).
5. Подравняване на размерите на данните.
6. Изчисляване на корелацията по Пиърсън за определяне колко смислена би била препоръка на книга на базата на възрастта на читателя и рейтинга на книгите.
7. Подготовка на данните и разделянето им на тренировъчни и тестови множества данни.
8. Създаване и трениране на модела с тренировъчните множества данни.
9. Грануларно изчисление на разстоянията на текущо исзледвания пример и неговите k на брой най-близки съседи.
10. Препоръка на книга.
11. Оценка на качеството на модела.

##### Алгоритъмът в код

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

from scipy.stats import pearsonr

import warnings

warnings.filterwarnings("ignore")

# Load the data.

all\_ratings = pd.read\_csv("src\\data\\Ratings.csv")

all\_books = pd.read\_csv("src\\data\\Books.csv")

all\_users = pd.read\_csv("src\\data\\Users.csv")

print(

    f"Head records of Books:\n{all\_books.head()}\nHead records of users:\n{all\_users.head()}\nHead records of ratings:\n{all\_ratings.head()}\n"

)

merged\_data = pd.merge(

    all\_ratings.dropna(), all\_users[["User-ID", "Age"]], on="User-ID", how="inner"

)

print(f"Merged data:\n{merged\_data}\n")

print(f"Merged data info:\n{merged\_data.info()}\n")

initial\_len = len(merged\_data)

with\_nan\_len = merged\_data.isna().any(axis=1).sum()

# Drop NaN values from merged data (both Book-Rating and Age)

# Here, around 30% of readers' data is permanently lost because we do not know the age of these readers.

merged\_data = merged\_data.fillna(value=-1)

print(f"Cleaned of NaN values merged data:\n{merged\_data}\n")

print(f"Cleaned of NaN values merged data info:\n{merged\_data.info()}\n")

# Calculate percentage of rows without NaN values

without\_nans\_len\_percentage = (initial\_len - with\_nan\_len) / initial\_len \* 100

print(f"Percentage of records without NaN values: {without\_nans\_len\_percentage:.2f}%")

# Dimensions to be checkd for correlation by Pearson method before going

# to regression recommendation approach.

x\_ratings = np.array(merged\_data["Book-Rating"])

y\_ages = np.array(merged\_data["Age"])

# Calculate Pearson correlation coefficient

correlation, p\_value = pearsonr(x\_ratings, y\_ages)

print(f"Pearson Correlation: {correlation}")

print(

    f"P-value: {p\_value}"

)  # equals to zero => no linear relationship between readers' age and books' rating.

# Prepare features and target.

X = np.column\_stack(

    (merged\_data["Book-Rating"].values, merged\_data["Age"].values)

)  # Features: Book Rating and Age

y = merged\_data["Book-Rating"].values  # Target: Book Rating

print(f"X column stack of Book Rating and Age recommendation features:\n{X}\n")

print(f"y array of Book Ratings:\n{X}\n")

# Split into train and test datasets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

print(f"X\_train data set:\n{X\_train}\n")

print(f"y\_train data set:\n{y\_train}\n")

print(f"X\_test data set:\n{X\_test}\n")

print(f"y\_test data set:\n{y\_test}\n")

# Define the KNN regressor model with the Euclidien metric.

knn = KNeighborsRegressor(n\_neighbors=5, metric="euclidean")

# Train the model with the train data sets.

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

# Granular distance function.

def granular\_distance(weights, X1, X2):

    distances = (X1 - X2) \*\* 2

    weighted\_distances = weights \* distances

    return np.sqrt(np.sum(weighted\_distances))

# Recommend books function.

def recommend\_books(

    user\_data, k=5, weights=np.array([2, 1])

):  # address that the book's rating will be taken with

    # bigger priority than the reader's age

# Merge all\_ratings and all\_users data to get corresponding ratings and ages.

    merged\_data = pd.merge(

        all\_ratings, all\_users[["User-ID", "Age"]], on="User-ID", how="inner"

    )

    # Drop NaN values to avoid issues.

    merged\_data = merged\_data.dropna()

    # Calculate the features for all books (ratings, ages).

    all\_books\_features = np.column\_stack(

        (merged\_data["Book-Rating"].values, merged\_data["Age"].values)

    )

    # Calculate the granular distance for each book using the provided weights.

    distances = np.array(

        [

            granular\_distance(weights, user\_data, book\_data)

            for book\_data in all\_books\_features

        ]

    )

    # Find the k nearest books based on granular distance.

    indices = np.argsort(distances)[:k]

    # Use the indices to get book IDs from merged\_data.

    # The books unique identifiers are their ISBN numbers.

    book\_ids = merged\_data.iloc[indices]["ISBN"].values

    # Prepare recommendations ad book's title and author.

    recommendations = []

    for book\_id in book\_ids:

        book\_info = all\_books[all\_books["ISBN"] == book\_id].iloc[

            0

        ]  # Fetch the book details by ISBN

        book\_title = book\_info["Book-Title"]

        book\_author = book\_info["Book-Author"]

        recommendations.append((book\_title, book\_author))

    return recommendations

# Examples of recommending for a new user:

# Example 1: A user who is 33 years old wants to read a book with rating 3.

new\_user = np.array([[3, 33]])

recommendations = recommend\_books(new\_user)

print("Recommendations for a 33 years old user who wants to read a book with rating 3:")

for rec in recommendations:

    print(f"Book Title: {rec[0]}, Author: {rec[1]}")

print()

# Example 2: A user who is 25 years old wants to read a book with rating 5.

new\_user2 = np.array([[5, 25]])

recommendations2 = recommend\_books(new\_user2)

print("Recommendations for a 25 years old user who wants to read a book with rating 5:")

for rec in recommendations2:

    print(f"Book Title: {rec[0]}, Author: {rec[1]}")

print()

# Generate predictions for evaluating the model.

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

# Evaluation of recommendation quality in percentages.

def precision\_at\_k(y\_true, y\_pred, k):

    relevant = set(y\_true[:k])  # Top-k true ratings.

    recommended = set(y\_pred[:k])  # Top-k predicted ratings.

    return len(relevant & recommended) / k

def recall\_at\_k(y\_true, y\_pred, k):

    relevant = set(y\_true)  # All true ratings.

    recommended = set(y\_pred[:k])  # Top-k predicted ratings.

    return len(relevant & recommended) / len(relevant)

# Example metrics calculation.

y\_true = y\_test[:10]  # Top-10 true ratings (or actual ratings for evaluation).

y\_pred\_top\_k = y\_pred[:10]  # Top-10 predicted ratings.

precision = precision\_at\_k(y\_true, y\_pred\_top\_k, k=5)

recall = recall\_at\_k(y\_true, y\_pred\_top\_k, k=5)

# For example Precision@5 = 0.4 means that, among the top 5 recommendations made

# by the model, 40% were relevant (i.e., they matched the true ratings or user

# preferences).

print(f"Precision@5: {precision}")

# For example Recall@5 = 0.66666666666666666 means that, of all the relevant items for the user,

# around 67% were included in the top 5 recommendations.

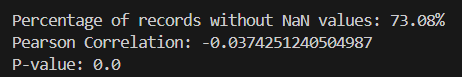
print(f"Recall@5: {recall}")

# As we saw earlier this model is not suitable for aour task because there is no linear

# relationship between the considered books' recommendation components.

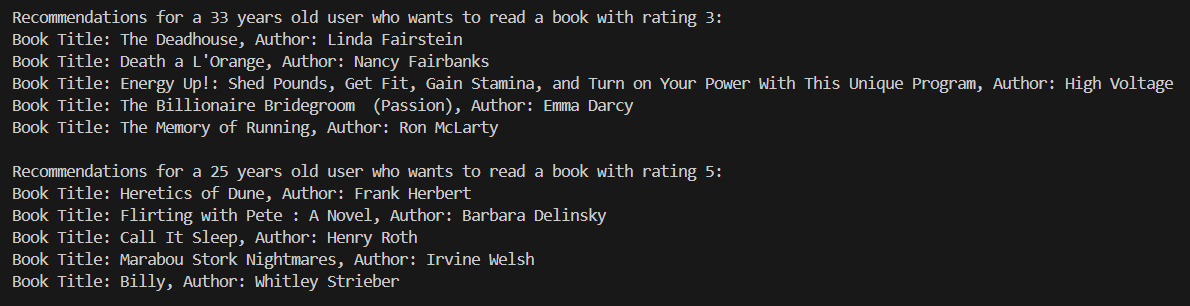
##### Резултати

* **Наблюдавани проблеми**: Основен проблем, който срещнахме при този подход е, че трябваше да се установи дали между двата параметъра - години на читателя и рейтинг на книгата съществува линейна връзка, която да ни позволява да използваме регресия за оценка на книгите и потенциални препоръки към читателите. За целта обаче първо трябваше да зачистим данните от незначещи стойности (като NaN стойности). При анализ на данните от трите таблици открихме, че около 30% от читателите не са посочили своята възраст, което доста намали шансовете да постигнем извода за връзка между > въпросните два параметъра, на които искахме да базираме нашите препоръки. Все пак проведохме този експеримент като проверихме коефициента за корелация по метода на Пиърсън.
* **Корелация по Пиърсън**: След като проверихме връзката на рейтинга на книгите и възрастта на читателите, установихме, че p-value-то е равно на 0.0, което от своя страна показва, че между двата > компонента не еналична линейна връзка и следователно регресията тук не би довела до добри препоръки на книги.



* **Скрийншот на препоръките на книги на базата на рейтингите на книгите и възрастта на читателите:**

Снимката по-долу предоставя препоръките на книги за двама читателя с тотално различни характеристики - единият читател е на 33 години и предпочита книги с рейтинг 3 / 10, а другият читател е на 25 години и предпочита книги с рейтинг 5 / 10. Информацията, която се предоставя на читателя при препоръка е името и автора на книгата.



* **Оценка на качеството на модела**: След като открихме, че този подход не би бил удачен в нашата задача, ние решихме да продължим с него с напълно експериментални цели, за да затвърдим или евентуално опровергаем това ни заключение. Завършихме подхода по описаните по-горе стъпки и в последната стъпка при оценката на качеството на модела достигнахме до извода, че едва 40% от препоръките са релевантни на предпочитанията на читателите и че около 67% от релевантните препоръки са всключени в топ 5 преопръчвани на отделния читател книги, тоест между 1 и 2 от 5 препоръчани на читателя книги наистина съответства на неговите предпочитания.

recommendations_knn_reg_granular_compute_eval

##### Изводи

Този резултат само затвърждава по-ранно изведения (чрез корелацията по метода на Пиърсън) извод, че един такъв модел не би бил достатъчно ефективен при препоръките на книги, следователно следва да се мисли в друга посока, например оптимизирания k-NN алгоритъм (, познат още като Approximate K Nearest Neighbour - а-NN алгоритъм, който ще разгледаме малко по-надолу).

#### а-NN алгоритъм (оптимизиран k-NN алгоритъм)

##### Описание на алгоритъма

Приблизителният най-близък съсед (ANN) е важен проблем в машинното обучение и има множество приложения в различни области. Значението на търсенето на ANN се крие в способността му да намира ефективно приблизителни решения на проблема с най-близките съседи, което може да доведе до значително ускоряване и спестяване на памет в сравнение с търсенето на точни най-близки съседи. Очевидно има компромис между точността на приблизителните резултати и времето, което се изразходва за изчисления. В някои задачи наборът от данни е толкова голям, че напълно точното търсене е дори невъзможно в разумно време, а в други - разликата между точния най-близък съсед и приблизителния съсед, който е близо, може да не бъде забележима. ANN се използва в различни приложения:

* **Компютърно зрение**
* **Извличане на информация**
* **Системи за препоръки**
* **Изчислителна биология**

##### Основни разновидности на ANN алгоритъма

###### HNSW

### Основна идея

**Алгоритъмът HNSW (Hierarchical Navigable Small World)** е разширена версия на алгоритъма NSW (Navigable Small World), който се базира на концепцията за "малък свят". Тази концепция предполага, че в графовете всяка връзка е краткосрочна към съседите (около логаритмично количество стъпки), но също така има и няколко дългосрочни връзки, които позволяват ефективно глобално търсене. Алгоритъмът HNSW изгражда многопластова йерархична структура от графове, която позволява ефективно индексиране и търсене на най-близки съседи в високоизмерни пространства.

###### Faiss

### Основна идея

FAISS ускорява търсенето на най-близки съседи в големи набори от данни, като използва характеристиките на високомерните пространства. Традиционните методи като изчерпателното търсене (brute-force) или K-D дърветата са бавни и паметно интензивни при обработка на такива данни. FAISS предлага алгоритми, създадени специално за ефективно търсене в подобни пространства.

###### ANNOY

### Основна идея

Annoy изгражда йерархична дървовидна структура, наречена Annoy дърво, за да организира данните. Това позволява ефективно търсене на приблизително най-близки съседи. Алгоритъмът жертва малка част от точността на резултатите, за да постигне значително по-бързо време за търсене и по-ниска консумация на памет.

### Сравнение между ANNOY, HNSW и FAISS

| **Критерий** | **ANNOY** | **HNSW** | **FAISS** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| **Точност** | Висока, но контролирана от броя на дърветата. | Много висока, благодарение на малкия свят и дългосрочните връзки. | Много висока за големи и високодименсионални данни. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Скорост на търсене** | Бърза, но зависи от броя на дърветата. | Много бърза благодарение на оптимизираното търсене в графи. | Изключително бърза при големи данни с GPU оптимизация. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ефективност при добавяне на нови данни** | Изисква пълно изграждане на дърветата при нови данни. | Поддържа динамични актуализации (добавяне и премахване на точки). | Изисква частична или пълна прекалибрация на индексите за някои типове. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Използване на памет** | Ниско, създаден за работа с данни на диск. | Средно до високо, особено за големи графи. | Високо, изисква целият индекс да бъде в паметта за максимална ефективност. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Простота на внедряване** | Много лесен за настройка и използване. | Изисква по-сложна конфигурация и параметризация. | Сложна настройка, особено за специфични архитектури (GPU, специализирани индекси). |

|  |  |  |  |
| --- | --- | --- | --- |
| **Мащабируемост** | Подходящ за малки до средни набори от данни (до милиони елементи). | Добра за средни до големи набори (до десетки милиони точки). | Отлична за много големи набори (стотици милиони до милиарди точки). |

|  |  |  |  |
| --- | --- | --- | --- |
| **Поддръжка на специални функции** | Основен фокус върху баланс скорост/точност, без сложни функции. | Предоставя гъвкавост за настройка на точността и динамика на търсене. | Предлага разширени функции като мулти-GPU поддръжка, мулти-проба и индексиране на редки данни. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Приложимост за препоръки за книги** | Отличен за начални и средни по мащаб проекти с балансиран подход. | Подходящ за по-големи проекти, които изискват много висока точност и динамика. | Идеален за изключително мащабни приложения или сложни системи с високодименсионални данни. |

За повече информация, относно този алгоритъм и неговите разновидности, вижте следната [статия](https://rtriangle.hashnode.dev/approximate-nearest-neighbors-algorithms-and-libraries).

##### Изводи

След всички тези проведени експерименти над множествата от данните, разгледани и описнани в графа **Използвани алгоритми** и графа **Използвани данни** на документацията, се спираме на ANN (approximate nearest neighbour) алгоритъма като най-оптимално решение за идентифициране на най-добрата препоръка на база ..... По-надолу в документацията описваме крайното решение на проблема ни и основни моменти от кода в графа **Описание на програмната реализация** от документацията, а в графа **Примери, илюстриращи работата на програмната система** от документацията предоставяме скрийншоти на резултатите след изпълнението на конкретните примери заедно с разяснения по тях.

Забележка: Предвид средно големите данни, които се обработват в системата ни, неналичието на условие за изключителна точност на прогнозите (каквато би се изисквало в един медицински софтуер, свързан с прдсказания на симптоми на заболявания например) и неналичието на толкова ефективна изчислителна мощ (, каквато би била необходима за релизацията на някоя друга от разновидностите на ANN алгоритъма), ние предпочетохме да използваме ANN алгоритъма в смисъла на неговата ANNOY разновидност като най-уместна.

### Описание на програмната реализация

##### Алгоритъмът ANNOY в стъпки

1. **Индексиране (Indexing)**:

· Избира се случайна точка за разделяне по случайно избран размер на данни, която създава кореновия възел на дървото.

* Данните се разделят рекурсивно в по-малки региони чрез избиране на нови точки за разделяне на различни размери по всяко ниво на дървото.
* Стремежът е да се разпределят данните равномерно през дървото, с цел балансирана структура на дървото.

2. **Баланс (Balancing)**:

· Алгоритъмът гарантира, че дървото остава балансирано, като поддържа максимален размер за всеки възел.

* Ако броят на точките в даден възел надвишава максималния размер, възелът се разделя на два нови дъщерни възела чрез избиране на нова точка за разделяне.
* Процесът продължава рекурсивно, докато не се създадат всички листни възли в дървото.

3. **Запитвания (Querying)**:

· Алгоритъмът започва от кореновия възел и сравнява точката за запитване с точката за разделяне в текущия възел.

* Въз основа на това сравнение алгоритъмът избира дъщерния възел, който е по-близо до точката за запитване.
* Процесът продължава рекурсивно надолу през дървото, като се избира дъщерният възел, който е по-близо до точката за запитване, докато не се достигне листен възел.

4. **Приближени резултати (Approximate results)**:

· След като се достигне листен възел, алгоритъмът извлича данните в този възел като начален набор от приблизителни най-близки съседи.

* Алгоритъмът усъвършенства приближените резултати чрез брутално търсене в ограничен радиус около точката за запитване в листния възел.
* Радиусът се определя от максималното разстояние между точката за запитване и приблизителните най-близки съседи, като алгоритъмът актуализира резултатите с всякакви по-близки точки, открити по време на търсенето в радиуса.

#### Алгоритъмът ANNOY в код (финална версия на кода на системата за препоръки на книги)

### Примери, илюстриращи работата на програмната система

Тук се включват примери (в това число снимки на екрани и др.), които илюстрират основните случаи на работа/използване на създадената програмна система, съответните данни и получените резултати.

### Литература

* **Сетове с данни:** https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset
* **Ръководство за създаване на система за препоръки в 1 час (част 1):** https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-the-fundamentals-dfee054f978e
* **Ръководство за създаване на система за препоръки в 1 час (част 2):** https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c
* **Kaggle notebook за по-добро разбиране на данните:** https://www.kaggle.com/code/arashnic/recom-i-data-understanding-and-simple-recomm
* **Еволюция на алгоритмите за системи, отправящи препоръки (част 1):**  <https://medium.com/@anicomanesh/evolution-of-recommendation-algorithms-part-i-fundamentals-and-classical-recommendation-bb1c0bce78a9>
* **Снимки за илюстрация на използваните алгоритми:** <https://www.almabetter.com/bytes/articles/types-of-recommendation-systems-how-they-work-and-their-use-cases>
* **ANNOY алгоритъм:** <https://rtriangle.hashnode.dev/approximate-nearest-neighbors-algorithms-and-libraries>