knn-book-recommendation

August 21, 2025

1 Book Recommendation Engine Using KNN

Project Structure

1. Data Loading & Exploration
2. Data Preprocessing
3. Model Building
4. Recommendation Function
5. Testing

2 01- Import Libraries & Load Dataset

[8]:
```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import seaborn as sns

# Load data with updated parameters
try:
    books = pd.read_csv('BX-Books.csv', sep=';', on_bad_lines='skip',␣
 ↪encoding="latin-1")
    users = pd.read_csv('BX-Users.csv', sep=';', on_bad_lines='skip',␣
 ↪encoding="latin-1")
    ratings = pd.read_csv('BX-Book-Ratings.csv', sep=';', on_bad_lines='skip',␣
 ↪encoding="latin-1")
except FileNotFoundError:
    # If files aren't in current directory, try loading from URL
    import io
    import requests

    def load_from_url(url):
        response = requests.get(url)
```

```
        return pd.read_csv(io.StringIO(response.text), sep=';',␣
↪on_bad_lines='skip', encoding="latin-1")

    books_url = "https://raw.githubusercontent.com/zygmuntz/goodbooks-10k/
↪master/samples/bx-books.csv"
    users_url = "https://raw.githubusercontent.com/zygmuntz/goodbooks-10k/
↪master/samples/bx-users.csv"
    ratings_url = "https://raw.githubusercontent.com/zygmuntz/goodbooks-10k/
↪master/samples/bx-book-ratings.csv"

    books = load_from_url(books_url)
    users = load_from_url(users_url)
    ratings = load_from_url(ratings_url)

# Explore data shapes
print(f"Books shape: {books.shape}")
print(f"Users shape: {users.shape}")
print(f"Ratings shape: {ratings.shape}")

# Show sample data
print("\nBooks sample:")
print(books.head())
print("\nRatings sample:")
print(ratings.head())
print("\nUsers sample:")
print(users.head())
```

```
/tmp/ipython-input-2941081801.py:10: DtypeWarning: Columns (3) have mixed types.
Specify dtype option on import or set low_memory=False.
  books = pd.read_csv('BX-Books.csv', sep=';', on_bad_lines='skip',
encoding="latin-1")
```

```
Books shape: (271360, 8)
Users shape: (278858, 3)
Ratings shape: (1149780, 3)

Books sample:
         ISBN                                          Book-Title  \
0  0195153448                                Classical Mythology
1  0002005018                                       Clara Callan
2  0060973129                               Decision in Normandy
3  0374157065  Flu: The Story of the Great Influenza Pandemic…
4  0393045218                              The Mummies of Urumchi

            Book-Author Year-Of-Publication                   Publisher  \
0    Mark P. O. Morford                2002     Oxford University Press
1  Richard Bruce Wright                2001       HarperFlamingo Canada
2          Carlo D'Este                1991             HarperPerennial
```

```
3      Gina Bari Kolata                1999        Farrar Straus Giroux
4       E. J. W. Barber                1999  W. W. Norton &amp;amp; Company

                                          Image-URL-S  \
0  http://images.amazon.com/images/P/0195153448.0…
1  http://images.amazon.com/images/P/0002005018.0…
2  http://images.amazon.com/images/P/0060973129.0…
3  http://images.amazon.com/images/P/0374157065.0…
4  http://images.amazon.com/images/P/0393045218.0…

                                          Image-URL-M  \
0  http://images.amazon.com/images/P/0195153448.0…
1  http://images.amazon.com/images/P/0002005018.0…
2  http://images.amazon.com/images/P/0060973129.0…
3  http://images.amazon.com/images/P/0374157065.0…
4  http://images.amazon.com/images/P/0393045218.0…

                                          Image-URL-L
0  http://images.amazon.com/images/P/0195153448.0…
1  http://images.amazon.com/images/P/0002005018.0…
2  http://images.amazon.com/images/P/0060973129.0…
3  http://images.amazon.com/images/P/0374157065.0…
4  http://images.amazon.com/images/P/0393045218.0…

Ratings sample:
   User-ID        ISBN  Book-Rating
0   276725  034545104X            0
1   276726  0155061224            5
2   276727  0446520802            0
3   276729  052165615X            3
4   276729  0521795028            6

Users sample:
   User-ID                            Location   Age
0        1                  nyc, new york, usa   NaN
1        2           stockton, california, usa  18.0
2        3     moscow, yukon territory, russia   NaN
3        4           porto, v.n.gaia, portugal  17.0
4        5  farnborough, hants, united kingdom   NaN
```

2.1 02- Data Preprocessing

[9]:
```python
def preprocess_data(ratings, users, books, min_book_ratings=100,␣
↪min_user_ratings=200):
    """Preprocess data by filtering based on rating thresholds"""
    # Count ratings per user and book
    user_rating_counts = ratings['User-ID'].value_counts()
```

```
    book_rating_counts = ratings['ISBN'].value_counts()

    # Filter users and books
    valid_users = user_rating_counts[user_rating_counts >= min_user_ratings].
↪index
    valid_books = book_rating_counts[book_rating_counts >= min_book_ratings].
↪index

    # Apply filters
    filtered_ratings = ratings[
        (ratings['User-ID'].isin(valid_users)) &
        (ratings['ISBN'].isin(valid_books))
    ]

    # Merge with book data
    final_data = filtered_ratings.merge(books, on='ISBN')

    # Create pivot table
    book_pivot = final_data.pivot_table(
        index='Book-Title',
        columns='User-ID',
        values='Book-Rating'
    ).fillna(0)

    # Convert to sparse matrix
    book_sparse = csr_matrix(book_pivot.values)

    return book_pivot, book_sparse

book_pivot, book_sparse = preprocess_data(ratings, users, books)
```

2.2 03- Model Building

[10]:
```
def build_knn_model(sparse_matrix, n_neighbors=5, metric='cosine'):
    """Build and fit KNN model"""
    model = NearestNeighbors(
        n_neighbors=n_neighbors+1,  # +1 to exclude the book itself
        algorithm='auto',
        metric=metric
    )
    model.fit(sparse_matrix)
    return model

knn_model = build_knn_model(book_sparse)
```

2.3 04- Recommendation Function

[12]:
```
def get_recommends(book_title, model, pivot_table):
    """Get book recommendations for given title"""
    try:
        # Find book index
        book_idx = np.where(pivot_table.index == book_title)[0][0]

        # Get distances and indices of nearest neighbors
        distances, indices = model.kneighbors(
            pivot_table.iloc[book_idx, :].values.reshape(1, -1))

        # Prepare recommendations
        recommended_books = []
        for i in range(1, len(indices.flatten())):  # Skip first (itself)
            recommended_books.append([
                pivot_table.index[indices.flatten()[i]],
                distances.flatten()[i]
            ])

        # Sort by distance (closest first)
        recommended_books = sorted(recommended_books, key=lambda x: x[1],␣
 ↪reverse=True)

        return [book_title, recommended_books]

    except IndexError:
        return f"Book '{book_title}' not found in dataset"

# Test function
print(get_recommends("The Queen of the Damned (Vampire Chronicles␣
 ↪(Paperback))", knn_model, book_pivot))
```

```
['The Queen of the Damned (Vampire Chronicles (Paperback))', [['Catch 22',
np.float64(0.7939835419270879)], ['The Witching Hour (Lives of the Mayfair
Witches)', np.float64(0.7448657003312193)], ['Interview with the Vampire',
np.float64(0.7345068863988313)], ['The Tale of the Body Thief (Vampire
Chronicles (Paperback))', np.float64(0.5376338446489461)], ['The Vampire Lestat
(Vampire Chronicles, Book II)', np.float64(0.5178411864186413)]]]
```

[18]:
```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors

class BookRecommender:
    def __init__(self, ratings_path, books_path, users_path):
```

```
        self.ratings, self.books, self.users = self._load_data(ratings_path,␣
↪books_path, users_path)
        self.book_pivot, self.book_sparse = None, None
        self.model = None

    def _load_data(self, ratings_path, books_path, users_path):
        """Load and merge datasets using updated pandas parameters"""
        # Use on_bad_lines instead of error_bad_lines for newer pandas versions
        load_params = {
            'sep': ';',
            'encoding': 'latin-1',
            'on_bad_lines': 'skip'  # Changed from error_bad_lines
        }

        try:
            books = pd.read_csv(books_path, **load_params)
            users = pd.read_csv(users_path, **load_params)
            ratings = pd.read_csv(ratings_path, **load_params)
            return ratings, books, users
        except Exception as e:
            print(f"Error loading data: {str(e)}")
            # Try loading from URL if local files fail
            return self._load_backup_data()

    def _load_backup_data(self):
        """Load backup data from URLs if local files fail"""
        print("Attempting to load data from URLs...")
        base_url = "https://raw.githubusercontent.com/zygmuntz/goodbooks-10k/
↪master/samples/"
        urls = {
            'ratings': base_url + "bx-book-ratings.csv",
            'books': base_url + "bx-books.csv",
            'users': base_url + "bx-users.csv"
        }

        load_params = {
            'sep': ';',
            'encoding': 'latin-1',
            'on_bad_lines': 'skip'
        }

        books = pd.read_csv(urls['books'], **load_params)
        users = pd.read_csv(urls['users'], **load_params)
        ratings = pd.read_csv(urls['ratings'], **load_params)
        return ratings, books, users

    def preprocess(self, min_book_ratings=100, min_user_ratings=200):
```

```
        """Preprocess data with rating thresholds"""
        # Filter users and books
        user_counts = self.ratings['User-ID'].value_counts()
        book_counts = self.ratings['ISBN'].value_counts()

        valid_users = user_counts[user_counts >= min_user_ratings].index
        valid_books = book_counts[book_counts >= min_book_ratings].index

        filtered_ratings = self.ratings[
            (self.ratings['User-ID'].isin(valid_users)) &
            (self.ratings['ISBN'].isin(valid_books))
        ]

        # Create pivot table
        self.book_pivot = filtered_ratings.merge(
            self.books, on='ISBN'
        ).pivot_table(
            index='Book-Title',
            columns='User-ID',
            values='Book-Rating'
        ).fillna(0)

        self.book_sparse = csr_matrix(self.book_pivot.values)

    def train_model(self, n_neighbors=5, metric='cosine'):
        """Train KNN model"""
        if self.book_sparse is None:
            raise ValueError("Preprocess data first")

        self.model = NearestNeighbors(
            n_neighbors=n_neighbors+1,
            algorithm='auto',
            metric=metric
        )
        self.model.fit(self.book_sparse)

    def get_recommends(self, book_title):
        """Get recommendations for a book"""
        if self.model is None or self.book_pivot is None:
            raise ValueError("Model not trained or data not preprocessed")

        try:
            book_idx = np.where(self.book_pivot.index == book_title)[0][0]
            distances, indices = self.model.kneighbors(
                self.book_pivot.iloc[book_idx, :].values.reshape(1, -1))

            recommendations = [
```

```
                [self.book_pivot.index[i], d]
                for i, d in zip(indices.flatten()[1:], distances.flatten()[1:])
            ]

            return [book_title, sorted(recommendations, key=lambda x: x[1],␣
 ↪reverse=True)]

        except IndexError:
            return f"Book '{book_title}' not found in dataset"

# Usage example with error handling
try:
    recommender = BookRecommender(
        'BX-Book-Ratings.csv',
        'BX-Books.csv',
        'BX-Users.csv'
    )
    recommender.preprocess()
    recommender.train_model()

    test_book = "The Queen of the Damned (Vampire Chronicles (Paperback))"
    print(recommender.get_recommends(test_book))

except Exception as e:
    print(f"Error in recommendation system: {str(e)}")
```

```
/tmp/ipython-input-1533142611.py:22: DtypeWarning: Columns (3) have mixed types.
Specify dtype option on import or set low_memory=False.
  books = pd.read_csv(books_path, **load_params)
```

```
['The Queen of the Damned (Vampire Chronicles (Paperback))', [['Catch 22',
np.float64(0.7939835419270879)], ['The Witching Hour (Lives of the Mayfair
Witches)', np.float64(0.7448657003312193)], ['Interview with the Vampire',
np.float64(0.7345068863988313)], ['The Tale of the Body Thief (Vampire
Chronicles (Paperback))', np.float64(0.5376338446489461)], ['The Vampire Lestat
(Vampire Chronicles, Book II)', np.float64(0.5178411864186413)]]]
```

2.4 05- Testing

[19]:
```
# Test with the required book
test_book = "The Queen of the Damned (Vampire Chronicles (Paperback))"
print(recommender.get_recommends(test_book))

# Test with random books
for book in np.random.choice(recommender.book_pivot.index, 3):
    print("\nRecommendations for:", book)
    print(recommender.get_recommends(book))
```

['The Queen of the Damned (Vampire Chronicles (Paperback))', [['Catch 22', np.float64(0.7939835419270879)], ['The Witching Hour (Lives of the Mayfair Witches)', np.float64(0.7448657003312193)], ['Interview with the Vampire', np.float64(0.7345068863988313)], ['The Tale of the Body Thief (Vampire Chronicles (Paperback))', np.float64(0.5376338446489461)], ['The Vampire Lestat (Vampire Chronicles, Book II)', np.float64(0.5178411864186413)]]]

Recommendations for: Southern Cross
['Southern Cross', [['Cradle and All', np.float64(0.7611771486505596)], ['Unnatural Exposure', np.float64(0.7282497349087003)], ['The Body Farm', np.float64(0.7245286615411648)], ['Black Friday', np.float64(0.6070886378513438)], ['Isle of Dogs', np.float64(0.5439673619663369)]]]

Recommendations for: Seabiscuit
['Seabiscuit', [['Midwives: A Novel', np.float64(0.7740154179350087)], ['STONES FROM THE RIVER', np.float64(0.7728723983662207)], ['Midnight in the Garden of Good and Evil', np.float64(0.7532707322489147)], ['The Mulberry Tree', np.float64(0.748992565581007)], ['The Hundred Secret Senses', np.float64(0.7123004857457995)]]]

Recommendations for: Eats, Shoots & Leaves: The Zero Tolerance Approach to Punctuation
['Eats, Shoots & Leaves: The Zero Tolerance Approach to Punctuation', [['The Temple of My Familiar', np.float64(0.7929697117192669)], ['Tears of the Giraffe (No.1 Ladies Detective Agency)', np.float64(0.7845751024872697)], ["Ender's Game (Ender Wiggins Saga (Paperback))", np.float64(0.7826599851598155)], ['The Hours: A Novel', np.float64(0.7790514147795367)], ["Big Cherry Holler: A Big Stone Gap Novel (Ballantine Reader's Circle)", np.float64(0.769486277669488)]]]