BOOK Recommender system

Group members

- Xunzhi He
- Cassidy Lu
- William Zhao
- Huiyi He

Abstract

In our project, we're on a quest to transform book discovery on Goodreads using Julian McAuley's lab's ebook shelf data. To develop a sophisticated, genre-specific recommendation system that connects readers with books they'll adore, based on the reading habits of others with similar tastes. To achieve this, we're utilizing advanced clustering techniques such as K-means, Gaussian Mixture Models, and Hierarchical Clustering. These methods allow us to efficiently categorize users' preferences, from general likes to specific interests, ensuring highly personalized book recommendations. K-means helps identify the core preferences within the data, GMM adds depth by accounting for the variability of these preferences, and Hierarchical Clustering offers a detailed structure of readers' interests. To validate the effectiveness of our system, we're using the adjusted Rand index, which gauges the accuracy of our recommendations against real user data. Additionally, silhouette plots visually affirm the fit of each recommendation to user preferences, providing a clear picture of our system's performance. Our ultimate aim is to make finding your next favorite book on Goodreads a personalized, enjoyable, and effortless experience.

Background

The utilization of recommendation systems (RS) has become increasingly prevalent across diverse platforms such as Netflix, Spotify, and e-commerce portals like Amazon and eBay. These intelligent computer-based techniques can analyze user preferences to predict users' preferences and provide suggestions for items that may interest them, thereby enhancing user experience and achieving business benefits. For instance, Spotify can smartly curate playlists based on users' listening history and personal playlists, while e-commerce platforms recommend products tailored to individual preferences which could potentially increase customers' desire to purchase certain products[1].

RS systems have evolved from the earliest algorithms like Grundy, which allow users to query for items in an online information domain, to complex models applying advanced

machine learning techniques. These systems use various recommendation approaches, such as content-based filtering and demographic segmentation[1], to create personalized suggestions for users. Content-based recommender systems, for instance, analyze users' historical data to recommend items similar to their past preferences. Demographic recommendation systems utilize demographic profiles to personalize recommendations.

However, despite the advancements in RS systems, there are still some limits that persist in current recommendation systems. Traditional recommender systems might suffer the problem of poor scalability, data sparsity, and recommending similar items repeatedly[2]. Our project is considered a content-based recommender system and we will analyze users' ebook shelves data on the Goodreads website. We will apply several appropriate clustering methods such as the K-Means algorithm, and Gaussian Mixture Models (GMM) in developing high-quality recommendation systems that determine similarities to define a people group to build a book recommender system for users. Through the integration of advanced machine learning algorithms, our system will provide more accurate and diverse recommendations, thereby enhancing user experience on the Goodreads platform.

Problem Statement

We're diving into a project that's all about making it super easy to find awesome books on Goodreads. You know how it can be a bit overwhelming with all those books to choose from? Well, we're tackling that by developing a personalized book recommendation system. Our game plan involves using some fancy clustering algorithms—K-means for starters, and then stepping it up with Gaussian Mixture Models (GMM) and a hierarchy model for the heavy lifting. These algorithms will help us group books into clusters C_i based on cool features like genre, who wrote it, how much readers liked it, and what it's about $F = \{f_1, f_2, \ldots, f_n\}$. The idea is to make finding books that match what you like super straightforward, cutting down on that endless scrolling.

We're gonna check how awesome our system is by looking at things like the silhouette score, which pretty much tells us how tight our book clusters are. A high silhouette score means our clusters are spot on, making it easier to recommend books that are right up your alley. Plus, we'll use some dimension reduction magic like PCA to see how well our clusters stand out from each other.

By figuring out what users dig through these clustering methods, we can recommend books that hit the mark. Since this approach works for all sorts of users and platforms, it's perfect for making the digital book world a better place. We can keep tweaking our system with stats and machine learning tricks to keep up with what users are into, making sure our recommendations always stay fresh.

Data

We would like to use total of two dataset to create a recommendation system of books for the users. Information regarding data:

- link of the data: https://mengtingwan.github.io/data/goodreads#datasets[4][5]
- The first file we use as our dataset is goodreads_book_genres_initial.json.gz file containing all the book information for the goodreads website. It has total of 2360655 rows and 2 columns: book_id and genres. Each of the row correspond to one book. The book_id is a string of numbers which each of the id is unique in this dataframe. The column genres is a dictionary where the key is a string of genre and the value is an integer which is unknow for us what the integer represent since the dataset description did not specify what the integer represent.
- The second file is goodreads_interactions.csv file containing the reader information. It has total of 228648342 rows and 5 columns. The first column is user_id where each unique user has one unqiue id using a string of numbers to represent. The second column is book_id which will match the book_id from the first goodreads_book_genres's book_id. The third column is_read contain a boolean value 1 and 0. 1 represent the reader is currently reading the book and 0 otherwise. The 4th column rating uses an integer from range 0-5 to represent the user rating of a book. The last column is_reviewed also contain a boolean value 1 and 0. 1 represent the reader have left a review for this book and 0 otherwise.
- We will preform some special handling for both of the data. Since both data are huge and in order to prevent dead kernel while cleaning the dataframe and training the model, we have reduce the size of the goodreads_book_genres dataframe to 800000 books. We have also reduced the size of goodreads_interactions.csv to the first 200000 rows. We would merge those two dataframe using left mrege using book_id columns by merging on to the user dataframe(goodreads_interactions.csv). We would then filter out all the np.nan values in the dataframe by deleting the row that contains any np.nan values. For the rating column, we might want to standardize it to see if there are user giving extremly high or low rating. We would then change the genres into a list of strings representing the genre the book belongs in. Later we would use MultiLabelBinarizer from sklearn to one-hot encode the genre list for each book. We would also groupby the dataframe by user_id and standardize the number of book the read for each genre.

Preprocessing

In [1]:
```
# imports
import pandas as pd
import numpy as np
import json
import gzip
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# import ML packages
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import MultiLabelBinarizer, StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_samples, silhouette_score
```

In [2]:
```
# file path for Karl
# base_path = "C:/Users/karl2/Downloads/"
# base_path = "/Desktop/cogs118B/data/"
```

In [4]:
```
# load in json file
file_path = '/Users/mac/Documents/学习资料/UCSD/Winter24/COGS118B/Project/goodre

with gzip.open(file_path, 'rt', encoding = 'utf-8') as f:
    df = pd.read_json(f, lines = True)
```

In [6]:
```
# load in csv file with first 200000 row
csv_path = '/Users/mac/Documents/学习资料/UCSD/Winter24/COGS118B/Project/goodrea

user_df = pd.read_csv(csv_path,nrows=200000)
user_df
```

Out[6]:

	user_id	book_id	is_read	rating	is_reviewed
0	0	948	1	5	0
1	0	947	1	5	1
2	0	946	1	5	0
3	0	945	1	5	0
4	0	944	1	5	0
...	...	...	...	...	...
199995	437	51332	0	0	0
199996	437	51329	0	0	0
199997	437	98589	0	0	0
199998	437	98588	0	0	0
199999	437	83026	0	0	0

200000 rows × 5 columns

In [7]:
```
# only include 800000 books
book_df = df.sample(800000, random_state=1)
book_df
```

Out[7]:

	book_id	genres
1479845	6884978	{}
2032903	6070478	{'history, historical fiction, biography': 8, ...
1406890	6343285	{}
183904	17058517	{'non-fiction': 23}
425144	34284404	{}
...	...	...
686872	24840238	{'non-fiction': 20, 'fiction': 3, 'history, hi...
45306	32128623	{'children': 4}
1254359	35174758	{'comics, graphic': 42}
2183803	19166163	{'fantasy, paranormal': 1, 'romance': 1}
1083679	10354854	{'comics, graphic': 6, 'romance': 1, 'fiction'...

800000 rows × 2 columns

In [8]:
```
# clean the genres to list
book_df['genres_lst'] = book_df['genres'].apply(lambda x: list(x.keys()))
book_df = book_df[book_df['genres_lst'].str.len() != 0]
book_df
```

Out[8]:

	book_id	genres	genres_lst
2032903	6070478	{'history, historical fiction, biography': 8, ...	[history, historical fiction, biography, child...
183904	17058517	{'non-fiction': 23}	[non-fiction]
1584276	6885606	{'romance': 4, 'fiction': 1}	[romance, fiction]
636742	10961964	{'history, historical fiction, biography': 265...	[history, historical fiction, biography, roman...
234337	260335	{'non-fiction': 2, 'history, historical fictio...	[non-fiction, history, historical fiction, bio...
...	...	...	...
686872	24840238	{'non-fiction': 20, 'fiction': 3, 'history, hi...	[non-fiction, fiction, history, historical fic...
45306	32128623	{'children': 4}	[children]
1254359	35174758	{'comics, graphic': 42}	[comics, graphic]
2183803	19166163	{'fantasy, paranormal': 1, 'romance': 1}	[fantasy, paranormal, romance]
1083679	10354854	{'comics, graphic': 6, 'romance': 1, 'fiction'...	[comics, graphic, romance, fiction]

661288 rows × 3 columns

In [9]:
```
# merge book and users to merged_df
merged_df = user_df.merge(book_df, how = 'left', left_on = 'book_id', right_on
```

In [10]:
```
# check if there are any np.nan value left
merged_df.isna().all()
```

Out[10]:
```
user_id        False
book_id        False
is_read        False
rating         False
is_reviewed    False
genres         False
genres_lst     False
dtype: bool
```

In [11]:
```
merged_df
```

Out[11]:

	user_id	book_id	is_read	rating	is_reviewed	genres	genres_lst
6	0	942	1	5	0	{'non-fiction': 199, 'children': 33}	[non-fiction, children]
15	0	933	1	4	0	{'fiction': 17490, 'history, historical fictio...	[fiction, history, historical fiction, biograp...
17	0	931	1	5	0	{'fiction': 17490, 'history, historical fictio...	[fiction, history, historical fiction, biograp...
33	0	915	1	5	1	{'mystery, thriller, crime': 273, 'fiction': 94}	[mystery, thriller, crime, fiction]
35	0	913	0	0	1	{'non-fiction': 1466, 'fiction': 26, 'history,...	[non-fiction, fiction, history, historical fic...
...	...	...	...	...	...	...	...
199977	437	44789	0	0	0	{'non-fiction': 9, 'history, historical fictio...	[non-fiction, history, historical fiction, bio...
199987	437	51331	0	0	0	{'history, historical fiction, biography': 1}	[history, historical fiction, biography]
199991	437	98592	0	0	0	{'non-fiction': 74, 'fiction': 3, 'history, hi...	[non-fiction, fiction, history, historical fic...
199995	437	51332	0	0	0	{'non-fiction': 235, 'history, historical fict...	[non-fiction, history, historical fiction, bio...
199999	437	83026	0	0	0	{'non-fiction': 3388, 'history, historical fic...	[non-fiction, history, historical fiction, bio...

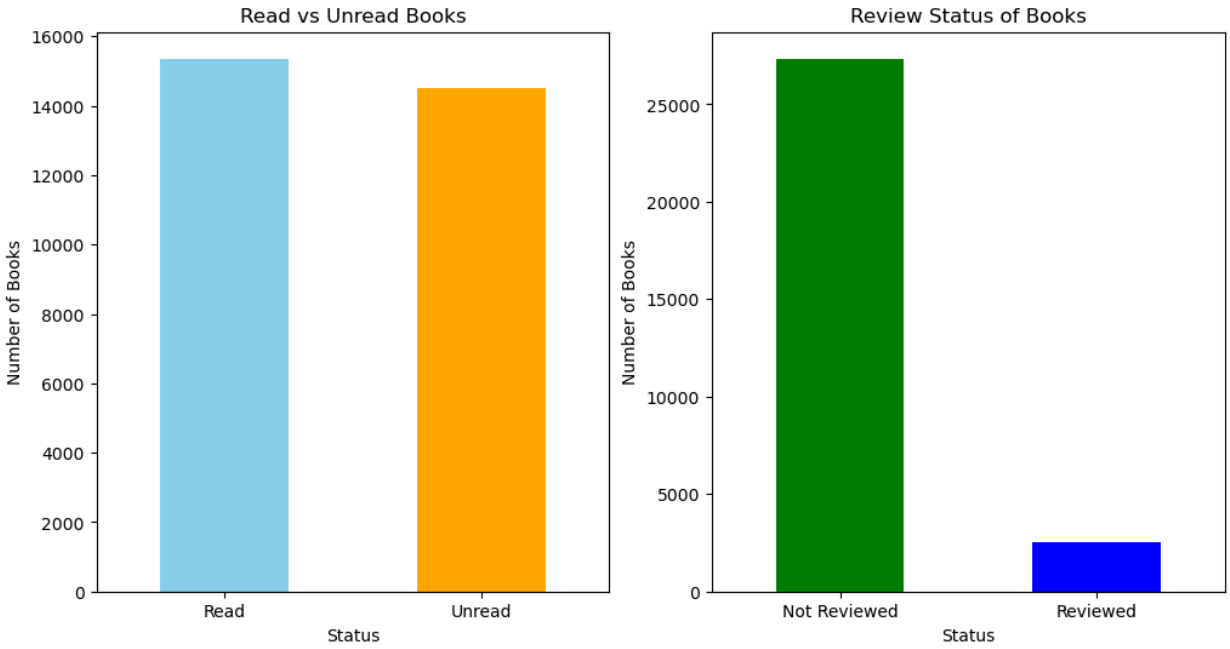
29859 rows × 7 columns

EDA

In [10]:
```
df = merged_df
general_stats = df.describe()

ratings_distribution = df[df['is_read'] == 1]['rating'].value_counts().sort_in

general_stats, ratings_distribution
```

Out[10]:
```
(            user_id       book_id       is_read        rating   is_reviewed
 count  29859.000000  29859.000000  29859.000000  29859.000000  29859.000000
 mean     221.035634  31098.412238      0.514384      1.892897      0.084095
 std      133.453416  28128.206487      0.499801      2.055101      0.277535
 min        0.000000      1.000000      0.000000      0.000000      0.000000
 25%      112.000000   7351.000000      0.000000      0.000000      0.000000
 50%      220.000000  20751.000000      1.000000      0.000000      0.000000
 75%      338.000000  50452.000000      1.000000      4.000000      0.000000
 max      437.000000  99500.000000      1.000000      5.000000      1.000000,
 rating
 0     848
 1     219
 2     844
 3    3469
 4    5689
 5    4290
 Name: count, dtype: int64)
```

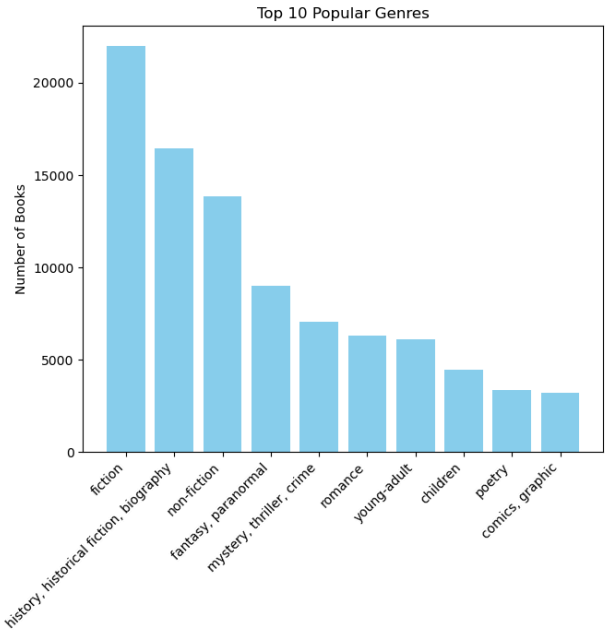
In [11]:
```
read_unread_distribution = df['is_read'].value_counts()
review_status_distribution = df['is_reviewed'].value_counts()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
read_unread_distribution.plot(kind='bar', color=['skyblue', 'orange'])
plt.title('Read vs Unread Books')
plt.xticks(ticks=[0, 1], labels=['Read', 'Unread'], rotation=0)
plt.xlabel('Status')
plt.ylabel('Number of Books')

plt.subplot(1, 2, 2)
review_status_distribution.plot(kind='bar', color=['green', 'blue'])
plt.title('Review Status of Books')
plt.xticks(ticks=[0, 1], labels=['Not Reviewed', 'Reviewed'], rotation=0)
plt.xlabel('Status')
plt.ylabel('Number of Books')

plt.show()
```

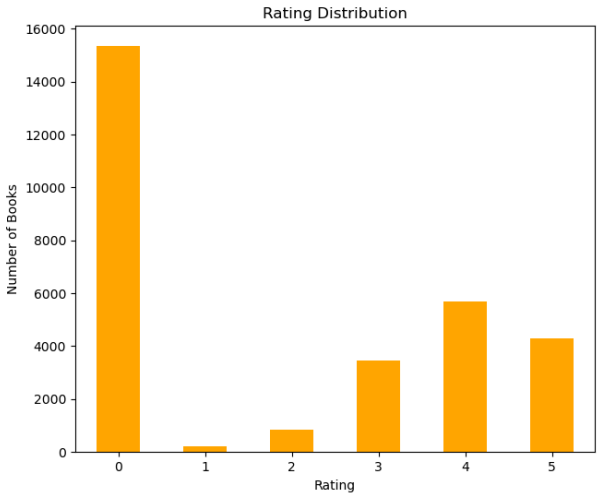


The exploratory data analysis (EDA) provided valuable insights into user interactions with books, highlighting active reading behavior but significantly lower levels of ratings and reviews. Most read books receive high ratings, yet the vast majority are not reviewed, indicating room for increased engagement. This analysis suggests the potential for improving recommendation systems and strategies to encourage more comprehensive user feedback. Understanding these user engagement patterns is crucial for tailoring content and enhancing the overall user experience.

In [12]:
```
genre_counts = {}
for genres in df['genres_lst']:
    for genre in genres:
        if genre in genre_counts:
            genre_counts[genre] += 1
        else:
            genre_counts[genre] = 1

sorted_genres = sorted(genre_counts.items(), key=lambda x: x[1], reverse=True)

top_genres = sorted_genres[:10]
genres, counts = zip(*top_genres)

plt.figure(figsize=(16, 6))

plt.subplot(1, 2, 1)
plt.bar(genres, counts, color='skyblue')
plt.title('Top 10 Popular Genres')
plt.xticks(rotation=45, ha='right')
plt.ylabel('Number of Books')

plt.subplot(1, 2, 2)
df['rating'].value_counts().sort_index().plot(kind='bar', color='orange')
plt.title('Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Number of Books')
plt.xticks(rotation=0)
```

```
plt.show()
```





Feature Selection

In [13]:
```
# One-hot encode the genres_lst columns (total unique 10 genres)
mlb = MultiLabelBinarizer()
genres_encoded = mlb.fit_transform(merged_df['genres_lst'])
genres_encoded
```

Out[13]:
```
array([[1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 1, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0]])
```

In [14]:
```
# create df with encoded genres
genres_df = pd.DataFrame(genres_encoded, columns=mlb.classes_)
genres_df['ident'] = [i for i in range((genres_df).shape[0])]
genres_df.head()
```

Out[14]:

	children	comics, graphic	fantasy, paranormal	fiction	history, historical fiction, biography	mystery, thriller, crime	non-fiction	poetry	romance	young adul
0	1	0	0	0	0	0	1	0	0	
1	0	0	0	1	1	0	1	0	1	
2	0	0	0	1	1	0	1	0	1	
3	0	0	0	1	0	1	0	0	0	
4	0	0	0	1	1	0	1	0	0	

In [15]:
```
# create df with only useful features
only_users = merged_df[['user_id', 'is_read', 'rating']]
only_users['ident'] = [i for i in range((only_users).shape[0])]
```

```
/var/folders/qt/13zmztf10v78004sfclyhp4m0000gn/T/ipykernel_1736/3880962739.py:
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  only_users['ident'] = [i for i in range((only_users).shape[0])]
```

In [16]:
```
# contains user and books info
user_genres = only_users.merge(genres_df, how = 'right', left_on = 'ident', ri
user_genres = user_genres.drop(['ident'], axis = 1)
user_genres.head()
```

Out[16]:

	user_id	is_read	rating	children	comics, graphic	fantasy, paranormal	fiction	history, historical fiction, biography	mystery, thriller, crime	non-fiction
0	0	1	5	1	0	0	0	0	0	1
1	0	1	4	0	0	0	1	1	0	1
2	0	1	5	0	0	0	1	1	0	1
3	0	1	5	0	0	0	1	0	1	0
4	0	0	0	0	0	0	1	1	0	1

In [17]:
```
# contains unique users with normalized genres, mean of rating, sum of is_read
user_df = user_genres.groupby('user_id').agg({**{'rating': 'mean',
                                                 'is_read': 'sum'},
                                              **{genre: 'sum' for genre in

row_sums = user_df[mlb.classes_].sum(axis=1)
user_df[mlb.classes_] = user_df[mlb.classes_].div(row_sums, axis=0)
user_df = user_df.drop(['is_read'], axis = 1)
user_df = user_df.drop(['rating'], axis = 1)
user_df.head()
```

Out[17]:

user_id	children	comics, graphic	fantasy, paranormal	fiction	history, historical fiction, biography	mystery, thriller, crime	non-fiction	poetry
0	0.036364	0.027273	0.079545	0.259091	0.197727	0.072727	0.190909	0.025000
1	0.000000	0.000000	0.073171	0.146341	0.170732	0.146341	0.317073	0.000000
2	0.050000	0.083333	0.033333	0.216667	0.200000	0.033333	0.300000	0.033333
3	0.011494	0.000000	0.120690	0.229885	0.224138	0.028736	0.149425	0.126437
4	0.000000	0.000000	0.141732	0.244094	0.220472	0.023622	0.133858	0.188976

Proposed Solution

We utilize three main clustering algorithms to analyze and categorize user preferences: K-means Clustering: Serves as the baseline for our clustering approach, grouping users by their preferences into k clusters. The KMeans function from sklearn.cluster is employed with n_clusters=10 and random_state=42, ensuring reproducibility.

1. **Data Preparation**: We will use the top 20,000 rows from `goodreads_interactions.csv`, as these users have read the most books and thus can provide more information for us to train our model. We will also randomly sample 80,000 books from `goodreads_book_genres_initial.json` to ensure a sufficiently large sample. The `rating` and `has_read` fields from `goodreads_interactions.csv` will be dropped. Since we need to use `has_read` for our book recommendation algorithm later, and `rating` to test how effective is our model at the end. The two datasets will then be merged to correlate users' interactions with specific book genres, enabling a comprehensive view of user preferences.

2. **Feature Engineering**: We will focus on the genres of books a user has read to summarize their book preferences.We belive the genres feature can capture the user preference for books since users tend to prefer certain genres depending ontheir personalities or experiences, and they tend to keep this preference for genres for long term.The aggregated data will be normalized to ensure uniformity and facilitate meaningful comparison between users.

3. **Gaussian Mixture Models (GMM)**: Offers a more flexible clustering technique by considering the distribution within each cluster. This is implemented using the GaussianMixture function from sklearn.mixture, allowing the model to accommodate the varied density of data points.

4. **Hierarchical Clustering**: Adds another dimension by organizing user preferences into a multi-level hierarchy. This approach uses the linkage and f cluster functions from scipy.cluster.hierarchy, creating a dendrogram that visually represents the hierarchical clustering.

Testing the Solution

Evaluation Metrics: The effectiveness of our recommendation system is quantified using the silhouette score, which measures how similar an object is to its own cluster compared to other clusters. A higher silhouette score indicates better-defined clusters. The silhouette_score function from sklearn.metrics is used to calculate this metric for each clustering method.

Dimensionality Reduction: To facilitate visualization and improve clustering performance, we apply Principal Component Analysis (PCA) using the PCA function from sklearn.decomposition. This reduces the dataset to two principal components, preserving the essential data structure while minimizing information loss.

Visualization: To illustrate the clustering results and the distribution of users within clusters, we employ scatter plots and dendrograms. This visual feedback is crucial for assessing the performance and fine-tuning our clustering algorithms.

Our system will be benchmarked against a basic genre-popularity model, providing a comparison to highlight our system's personalized recommendation capabilities. By integrating advanced clustering with detailed evaluation, our approach aims to significantly improve personalized book discovery on Goodreads.

In [18]:
```
#imports
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import umap
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import MultiLabelBinarizer
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.hierarchy import fcluster
from sklearn.preprocessing import StandardScaler
```

K-means cluster

In [19]:
```
# create and fit kmeans
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(user_df)
```

```
/Users/mac/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:1
412: FutureWarning: The default value of `n_init` will change from 10 to 'aut
o' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[19]:
```
KMeans(n_clusters=10, random_state=42)
```

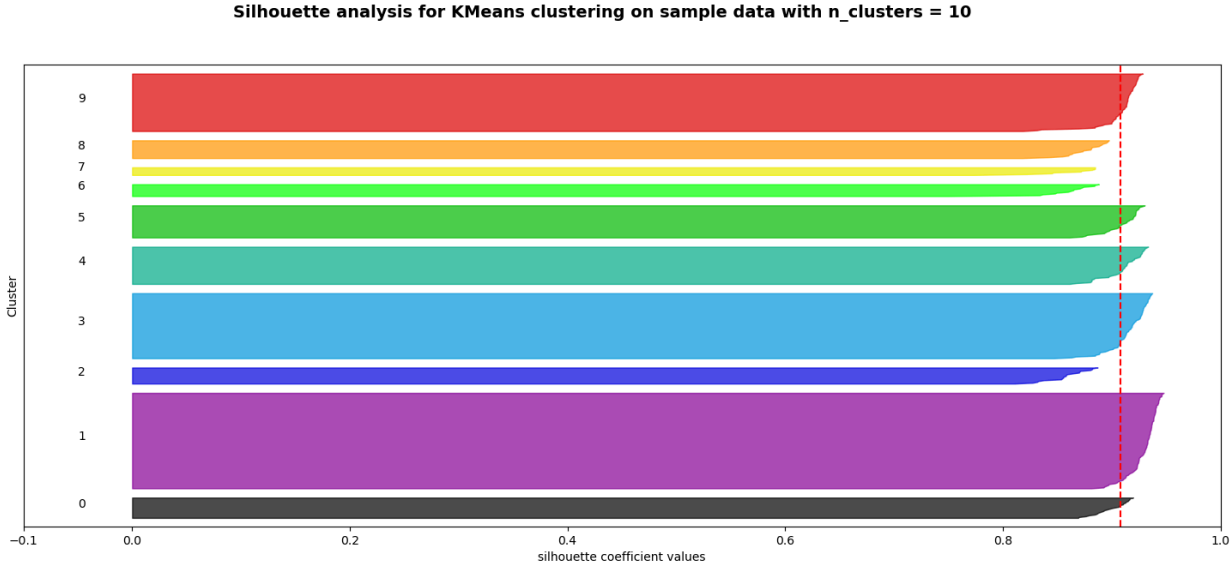
In [20]:
```
# add label in clusters
user_df['k_means_cluster'] = kmeans.labels_
```

In [21]:
```
# eval matrix
silhouette_avg = silhouette_score(user_df, kmeans.labels_)
silhouette_avg
```

Out[21]:
```
0.9079045762494306
```

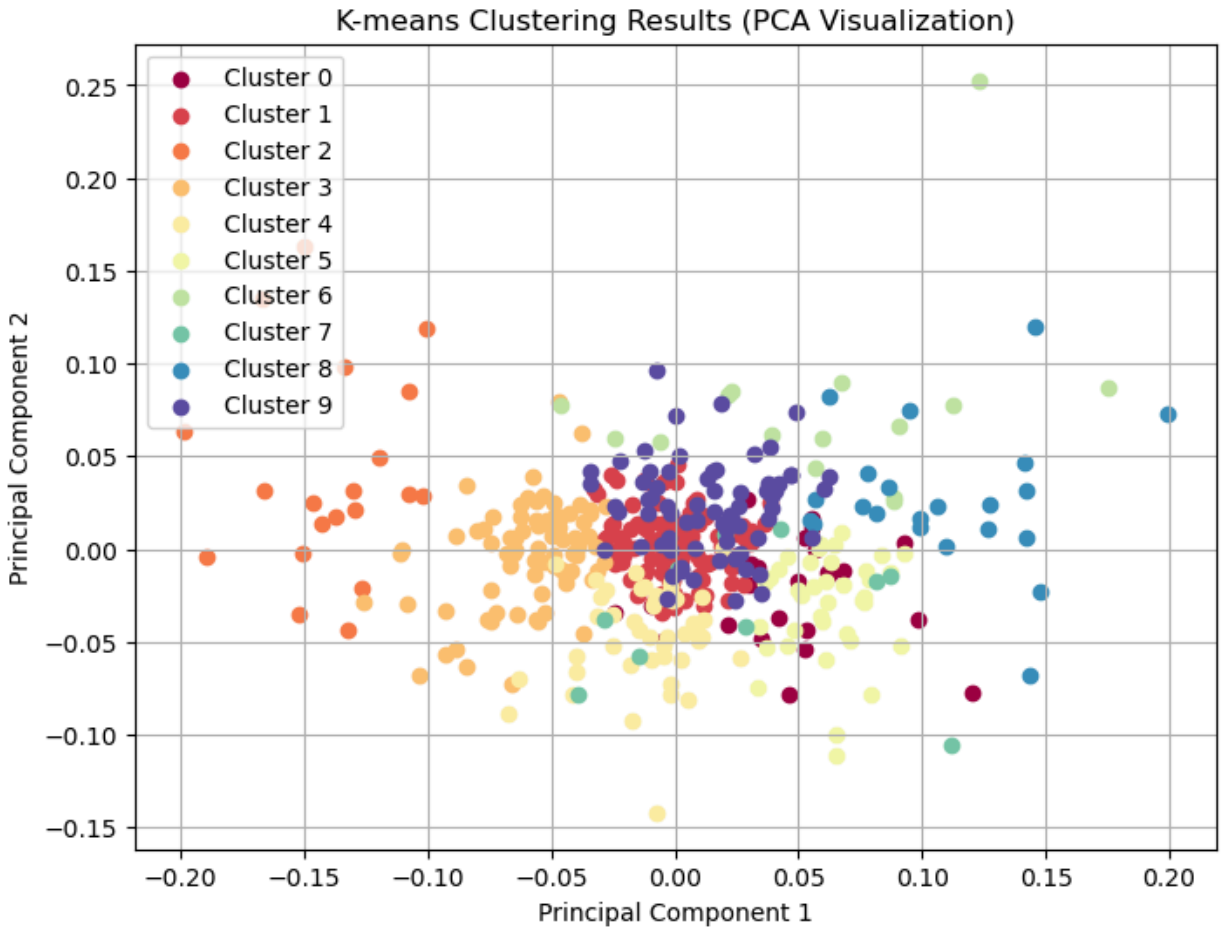
silhouette illustration for k-means

In [22]:
```
# graph silhouette
silhouette_avg = silhouette_score(user_df, kmeans.labels_)
sample_silhouette_values = silhouette_samples(user_df, kmeans.labels_)

# Create a subplot
fig, ax1 = plt.subplots(1, 1)
fig.set_size_inches(18, 7)

cluster_n = 10
ax1.set_xlim([-0.1, 1])
ax1.set_ylim([0, len(user_df) + (cluster_n + 1) * 10])

y_lower = 10

#5 clusters
for i in range(cluster_n):
    ith_cluster_silhouette_values = sample_silhouette_values[kmeans.labels_ ==
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = cm.nipy_spectral(float(i) / cluster_n)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)


    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))


    y_lower = y_upper + 10


ax1.set_xlabel("silhouette coefficient values")
ax1.set_ylabel("Cluster")


ax1.axvline(x=silhouette_avg, color="red", linestyle="--")


ax1.set_yticks([])
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# Display
plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
              f'with n_clusters = {cluster_n}'),
             fontsize=14, fontweight='bold')

plt.show()
```



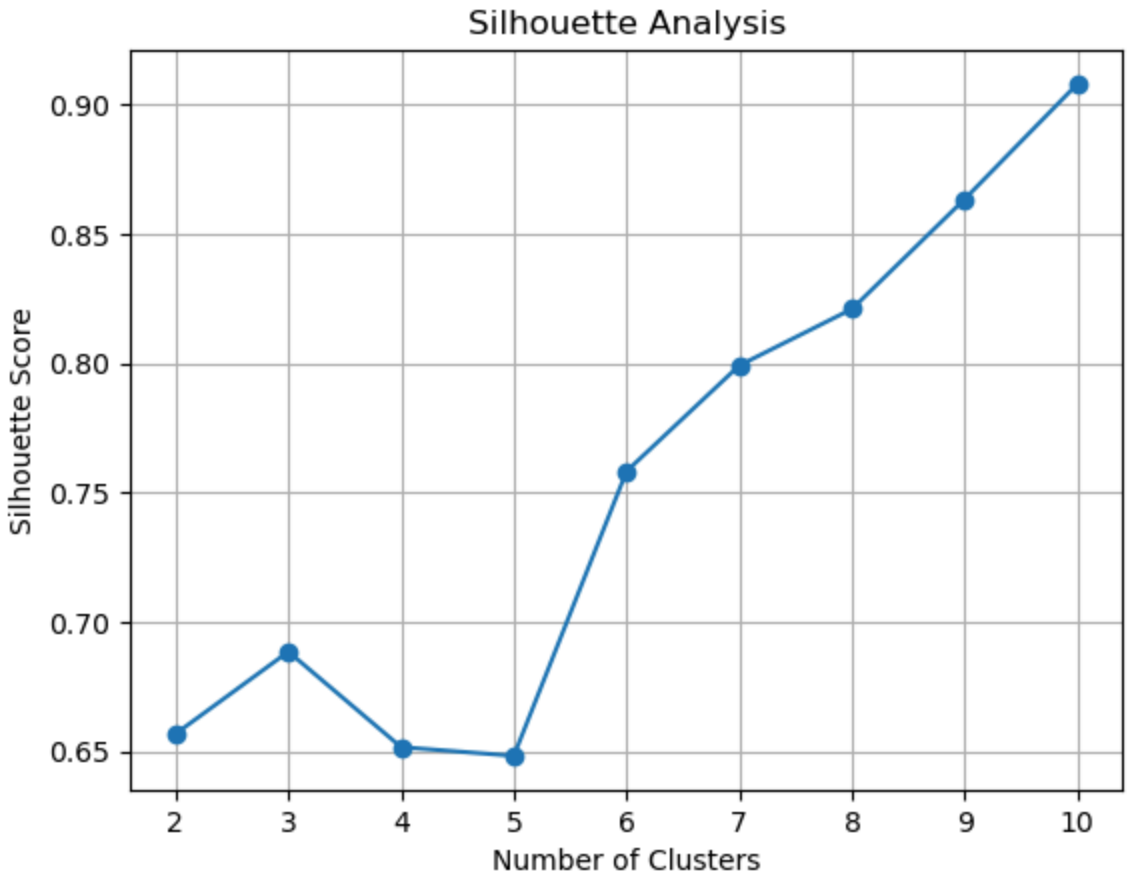
PCA analysis for k-means

In [23]:
```
# Extract data points and cluster assignments
data_points = user_df.drop('k_means_cluster', axis=1).values
cluster_labels = user_df['k_means_cluster'].values

# Number of unique clusters
unique_clusters = np.unique(cluster_labels)

# Run PCA 2 components
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_points)


colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_clusters))]

# Plot data points
plt.figure(figsize=(8, 6))
for i, cluster_label in enumerate(unique_clusters):
    color = colors[i]
    cluster_points = pca_result[cluster_labels == cluster_label]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=[color], label=f

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('K-means Clustering Results (PCA Visualization)')
plt.legend()
plt.grid(True)
plt.show()
```



Gaussian Mixture Model

In [24]:
```
# Search thought the different number of culsters
silhouette_scores = []

min_clusters = 2
max_clusters = 10

for n_clusters in range(min_clusters, max_clusters+1):
    # fit the GMM model
    gmm = GaussianMixture(n_components=n_clusters, random_state=42)
    cluster_labels = gmm.fit_predict(user_df)

    # silhouette score
    silhouette_avg = silhouette_score(user_df, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot
plt.plot(range(min_clusters, max_clusters+1), silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis')
plt.xticks(np.arange(min_clusters, max_clusters+1, 1))
plt.grid(True)
plt.show()
print(silhouette_scores)
```




```
[0.6568289876609937, 0.6885543148887644, 0.6516871103268398, 0.648311805182538
3, 0.758253647712333, 0.7991045785240968, 0.8209395867905894, 0.86341737501049
59, 0.9079045762494306]
```

In [25]:
```
# Standardize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(user_df)

# Reduce dimensionality with PCA
pca = PCA(n_components=10)
pca_features = pca.fit_transform(scaled_features)

# Initialize the GMM model
num_clusters = 10
gmm = GaussianMixture(n_components=num_clusters, random_state=42)


gmm.fit(pca_features)
cluster_labels = gmm.predict(pca_features)
user_df['GMM_cluster'] = cluster_labels
```

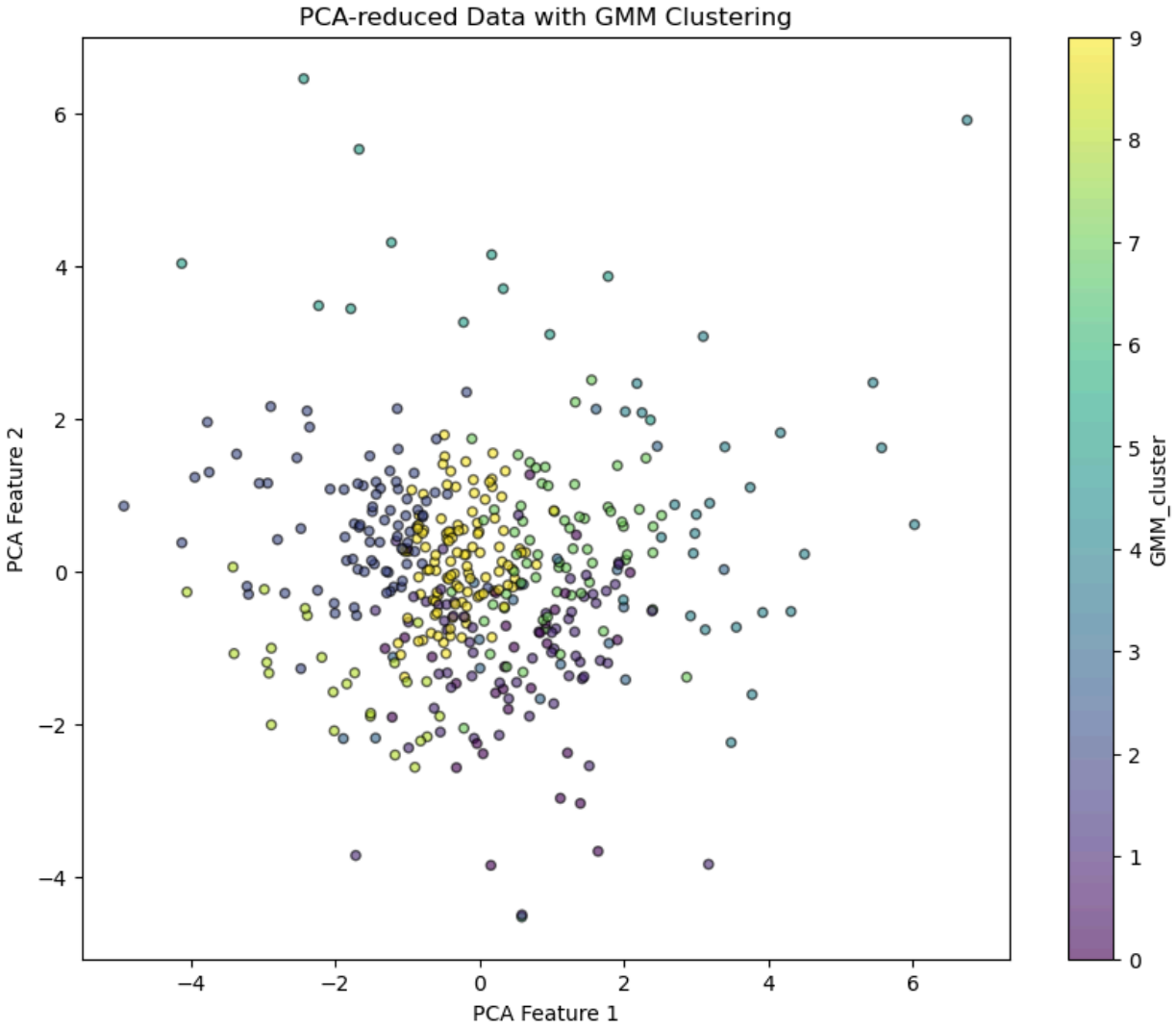
silhouette score for GMM

In [26]:
```
silhouette_avg = silhouette_score(user_df, cluster_labels)
silhouette_avg
```

Out[26]:
```
0.7456639009109439
```

PCA for GMM

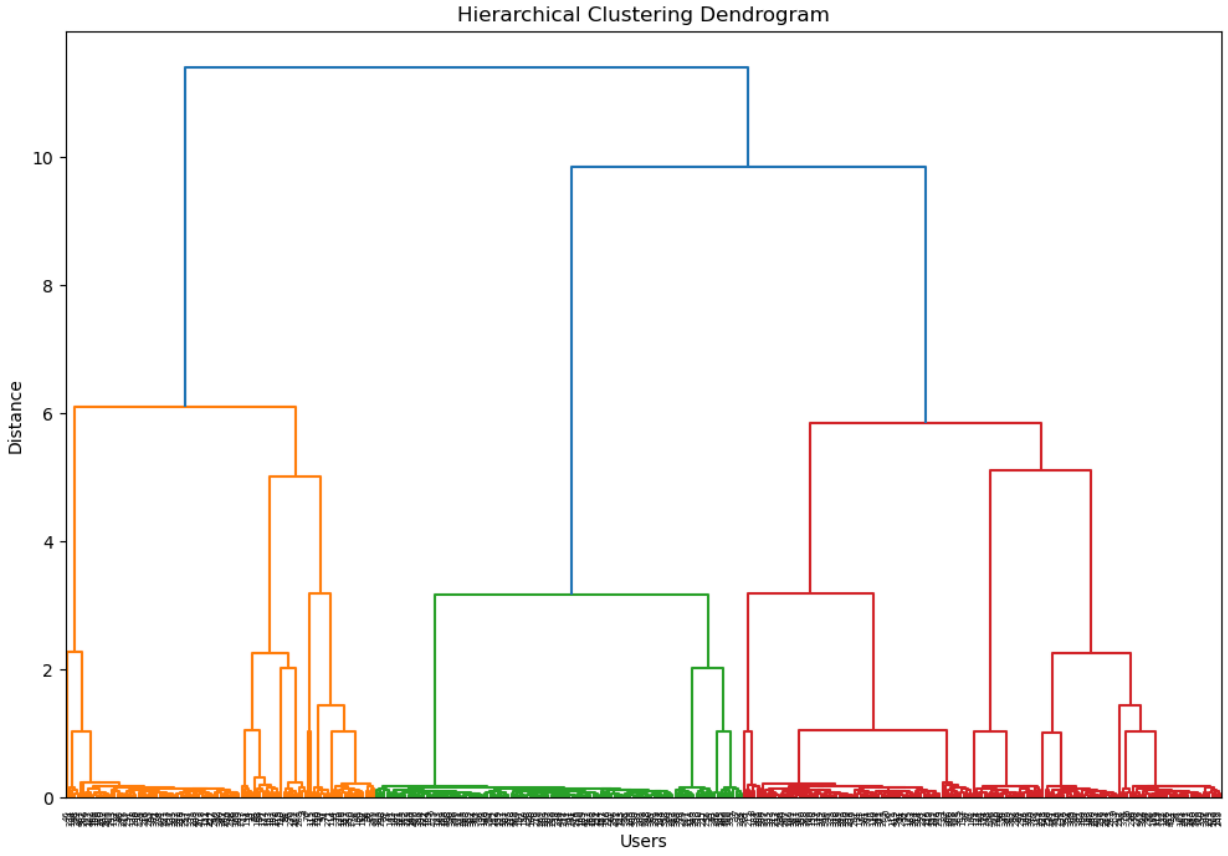
In [27]:
```
# Analyzing the clusters
# Distribution of users in each cluster
cluster_distribution = user_df['GMM_cluster'].value_counts()

# Visualize
plt.figure(figsize=(10, 8))
plt.scatter(pca_features[:, 0], pca_features[:, 1], c=cluster_labels, cmap='vi
            marker='o', edgecolor='k', s=20, alpha=0.6)
plt.title('PCA-reduced Data with GMM Clustering')
plt.xlabel('PCA Feature 1')
plt.ylabel('PCA Feature 2')
plt.colorbar(label='GMM_cluster')
plt.show()
```



Hierarchical clustering

In [28]:
```
linkage_matrix = linkage(user_df, method='complete', metric='euclidean')

plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, leaf_rotation=90)
```

```
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Users')
plt.ylabel('Distance')
plt.show()
```



silhouette score for Hierarchical

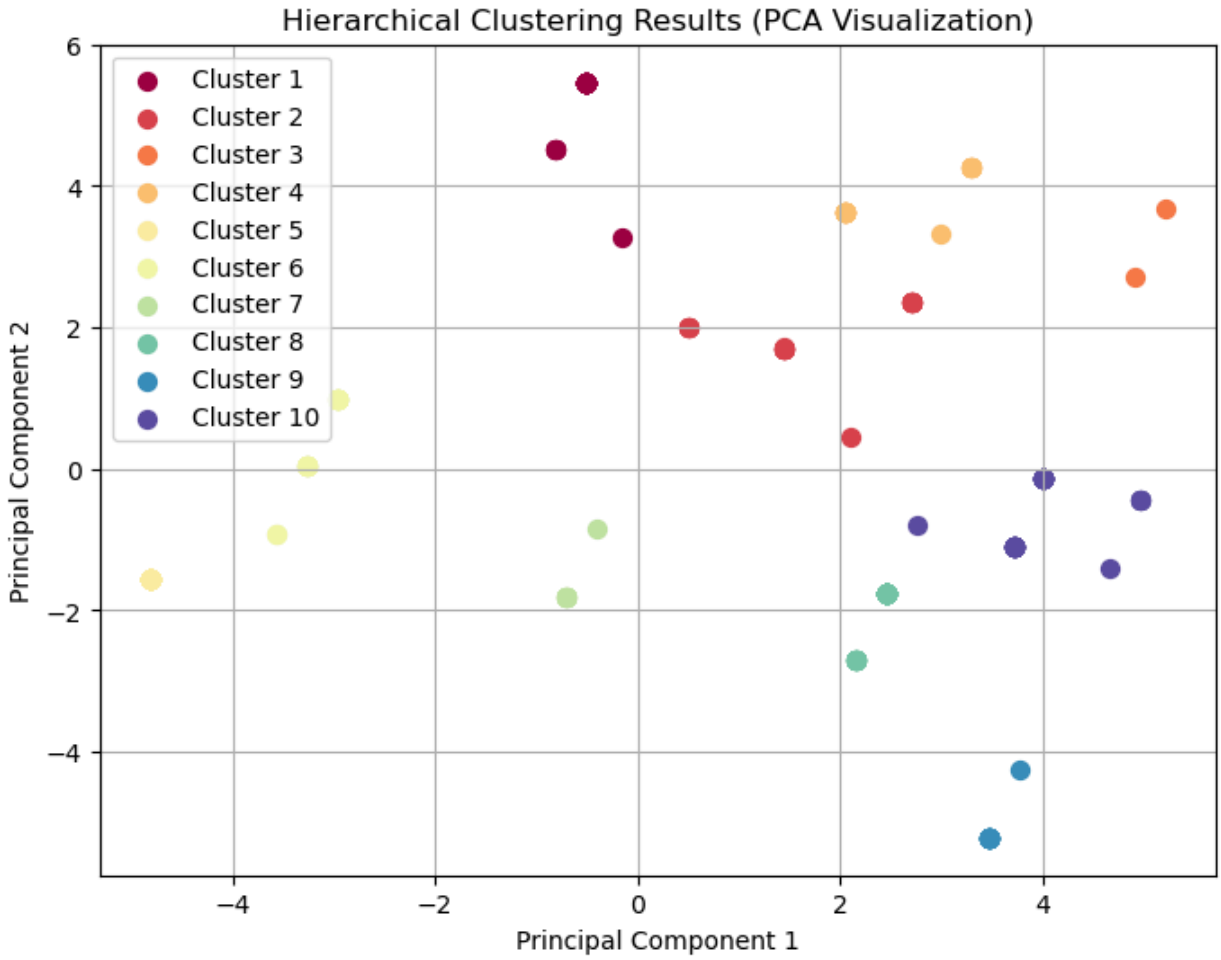
In [29]:
```
num_clusters = 10
cluster_labels = fcluster(linkage_matrix, num_clusters, criterion='maxclust')

# Compute the silhouette score
silhouette_avg = silhouette_score(user_df, cluster_labels)
silhouette_avg
```

Out[29]:
```
0.825655916534086
```

PCA for Hierarchical

In [30]:
```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(user_df)

# Plotting
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, num_clusters)]

plt.figure(figsize=(8, 6))
for i in range(1, num_clusters+1):

    cluster_points = pca_result[cluster_labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], s=50, c=[colors[i-
```

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Hierarchical Clustering Results (PCA Visualization)')
plt.legend()
plt.grid(True)
plt.show()
```



Evaluation Metrics

To evaluate the effectiveness of different clustering algorithms, we used silhouette score, t

since our project involves clustering users based on their preferences to improve book recommendations, and the decision to utilize Gaussian Mixture Models (GMM) K-means, and Hierarchy for clustering, it's appropriate to consider evaluation metrics that can quantitatively assess the effectiveness of these clustering approaches. Therefore, the silhouette score is an excellent metric for this purpose, and combining it with visualizations from dimensionality reduction techniques like PCA (Principal Component Analysis) offers both quantitative and qualitative insights into our model's performance.

Silhouette Score

Definition: The silhouette score is a measure of how similar an object is to its own cluster compared to other clusters. The silhouette score for a set of samples is given as the mean of the silhouette coefficient for each sample, where the silhouette coefficient for a single sample is defined as:

$$s = \frac{b - a}{\max(a, b)}$$

- Here, (a) is the mean distance between a sample and all other points in the same cluster (cohesion).
- (b) is the smallest mean distance from the sample to all points in any other cluster, of which the sample is not a part (separation).

Range: The silhouette score ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

Mathematical Representation: The overall silhouette score for the dataset is the average of all individual sample scores:

$$\text{Silhouette Score} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

where (N) is the number of samples and (s_i) is the silhouette score of the (i)th sample.

Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. The transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component, in turn, has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components.

Mathematics behind PCA:

1. Standardize the Dataset

Given a dataset (X) of dimensions (n \times m) (where (n) is the number of observations and (m) is the number of variables), standardize each variable to have a mean of 0 and a standard deviation of 1. The standardized data is denoted as (Z).

2. Compute the Covariance Matrix

Calculate the covariance matrix (C) of the standardized dataset (Z). The covariance matrix is an (m \times m) matrix where each element represents the covariance between two variables. The covariance of a variable with itself is just its variance.

$$C = \frac{1}{n-1} Z^T Z$$

3. Compute the Eigenvalues and Eigenvectors of the Covariance Matrix

Solve for the eigenvalues (\lambda) and eigenvectors (v) of the covariance matrix (C). The eigenvectors represent the directions of the principal components, while the eigenvalues represent their magnitudes (i.e., the amount of variance explained by each principal component).

4. Sort the Eigenvalues and Corresponding Eigenvectors

Order the eigenvalues from largest to smallest, and sort the eigenvectors accordingly. This prioritizes the principal components in order of significance.

5. Select the Principal Components

Choose the top (k) eigenvectors as the principal components, where (k) is the number of dimensions you want to reduce your data to. This subset will capture the most variance in the data.

6. Transform the Original Data

Finally, transform the original standardized data (Z) into the new subspace using the selected eigenvectors. The transformed data (Y), representing the scores of the original data on the principal components, is given by:

$$Y = ZV$$

where (V) is the matrix of selected eigenvectors.

Mathematical Summary:

- **Standardization**:

$$(Z = \frac{X - \mu}{\sigma})$$

- **Covariance Matrix**:

$$(C = \frac{1}{n-1} Z^T Z)$$

- **Eigen Decomposition**:

$$(Cv = \lambda v)$$

- **Transformation**:

$$(Y = ZV)$$

PCA allows you to reduce the dimensionality of the data while preserving as much variability as possible. This is particularly useful for visualizing high-dimensional data, reducing computational load, and mitigating the "curse of dimensionality" in machine learning models.

Evaluation Using Silhouette Score and PCA Visualization

- **Quantitative Evaluation with Silhouette Score**: Provides a clear numerical value to judge the clustering performance, indicating how well each user fits within their cluster compared to other clusters. This helps in selecting the optimal number of clusters for both K-means and GMM.
- **Qualitative Evaluation with PCA Visualization**: Offers a visual insight into how well the clustering has been performed by showing the spatial distribution of different clusters. This can reveal overlap between clusters or indicate if some clusters are more dispersed than others, which might not be apparent from the silhouette score alone.

Combining these two methods gives a comprehensive view of the clustering performance, guiding improvements in the clustering process and, consequently, the effectiveness of the book recommendation system.

Results

1. Hierarchical clustering

Here, we demonstrate hierarchical clustering analysis on a dataset (user_df) using the complete linkage method and Euclidean distance metric. Subsequently, we visualize the resulting dendrogram. We define the number of clusters to 10 to allocate each data point to one of 10 clusters. Finally, we compute the silhouette score, which is equal to 0.8257, to measure the clustering effectiveness. Such a high silhouette score indicates that the clusters are distinct and well-defined. Thus, the hierarchical clustering algorithm here can effectively grouped the data points based on their similarities.

2. K-mean clustering

We use the k-means clustering in the construction of a recommender system. We have defined the number of clusters to 10 because we have 10 distinct genres of books in our dataframe. THe silhouette score for this k-mean clustering is 0.907 which indicates a highly cohesive clustering outcome. When we perform a two dimensional PCA plot, this

dimensionality reduction techniques help to separate between clusters which can help the recommendation by understanding the centroid of each cluster.

3. GMM

We perform Gaussian Mixture Model (GMM) clustering analysis here. It iterates through different numbers of clusters (ranging from 2 to 10). We plot the silhouette scores and find out the optimal number of clusters is equal to 10. Additionally, the data is standardized using StandardScaler and dimensionality reduced using Principal Component Analysis (PCA).The GMM model is then initialized and fit to the PCA-transformed data, assigning cluster labels to each data point. We got the silhouette score = 0.746. We also visualize the PCA-reduced data colored by GMM clustering.

Discussion

Interpreting the result

In developing a recommender system, three clustering algorithms—Hierarchical Clustering, K-Means Clustering, and Gaussian Mixture Model (GMM)—were employed, each demonstrating unique strengths in grouping data points. Hierarchical Clustering, using complete linkage and Euclidean distance, achieved a silhouette score of 0.8257, indicating well-defined clusters. K-Means, aimed at categorizing 10 distinct book genres, outperformed with a silhouette score of 0.907, showcasing highly cohesive clusters, further visualized effectively through PCA for clear separation. GMM, exploring clusters from 2 to 10, standardized data and applied PCA, finding an optimal cluster count of 10 with a silhouette score of 0.746, highlighting its flexibility in accommodating varied data distributions. All methods identified 10 as the optimal number of clusters, corresponding to the dataset's structure. K-Means presented the highest silhouette score, suggesting superior cluster cohesiveness and distinction, whereas GMM's approach with data standardization and PCA showcased a nuanced understanding of data complexity. This comparison reveals the nuanced capabilities of each method, with K-Means standing out for its effectiveness in genre-based recommendations, while Hierarchical Clustering provides intuitive navigation, and GMM offers adaptability to complex datasets, guiding the choice of technique based on specific recommender system needs and data characteristics.

Limitations

The dataset currently used in our project comprises solely user IDs and their corresponding ebook shelves, which significantly restricts the depth of information available for analysis. This omission of detailed user feedback, such as rating scores for each book, poses a challenge to accurately depicting users' reading preferences. Such a limitation is consequential, as it might not only lead to a less precise recommendation system but also potentially introduce biases. These biases arise from an over-reliance on genre

categorization as the primary means of understanding user preferences. While genre is a useful indicator of interest, it is insufficient to capture the full spectrum of factors that influence reading choices. User preferences can be deeply influenced by aspects beyond genre, including the quality of storytelling, the author's reputation, thematic depth, and emotional resonance of the books. Consequently, the absence of these nuanced data points may result in a recommendation system that does not fully reflect the diverse tastes and interests of the user base.

Ethics & Privacy

Our project faces not just technical challenges but also ethical and privacy concerns, notably the potential for users to feel surveilled by the use of their ebook shelf data for recommendations. Balancing personalization with privacy is crucial. To tackle these issues, we're committing to transparency about our system's capabilities and its limitations, including addressing any dataset biases. We're also working on diversifying our data to better capture the wide range of reading preferences and behaviors, all while prioritizing user privacy and autonomy by adhering to data protection best practices. Our goal is to create a recommendation system that's not just smart and useful but also respectful and trustworthy, ensuring user trust and satisfaction by being ethically responsible from the ground up.

Conclusion

Our project faces not just technical challenges but also ethical and privacy concerns, notably the potential for users to feel surveilled by the use of their ebook shelf data for recommendations. Balancing personalization with privacy is crucial. To tackle these issues, we're committing to transparency about our system's capabilities and its limitations, including addressing any dataset biases. We're also working on diversifying our data to better capture the wide range of reading preferences and behaviors, all while prioritizing user privacy and autonomy by adhering to data protection best practices. Our goal is to create a recommendation system that's not just smart and useful but also respectful and trustworthy, ensuring user trust and satisfaction by being ethically responsible from the ground up.

Footnotes

1.^: Lorenz, T. (9 Dec 2021) Birds Aren't Real, or Are They? Inside a Gen Z Conspiracy Theory. *The New York Times*. https://www.nytimes.com/2021/12/09/technology/birds-arent-real-gen-z-misinformation.html

2.^: Also refs should be important to the background, not some randomly chosen vaguely related stuff. Include a web link if possible in refs as above.

3.^: Perhaps the current state of the art solution such as you see on Papers with code. Or maybe not SOTA, but rather a standard textbook/Kaggle solution to this kind of problem