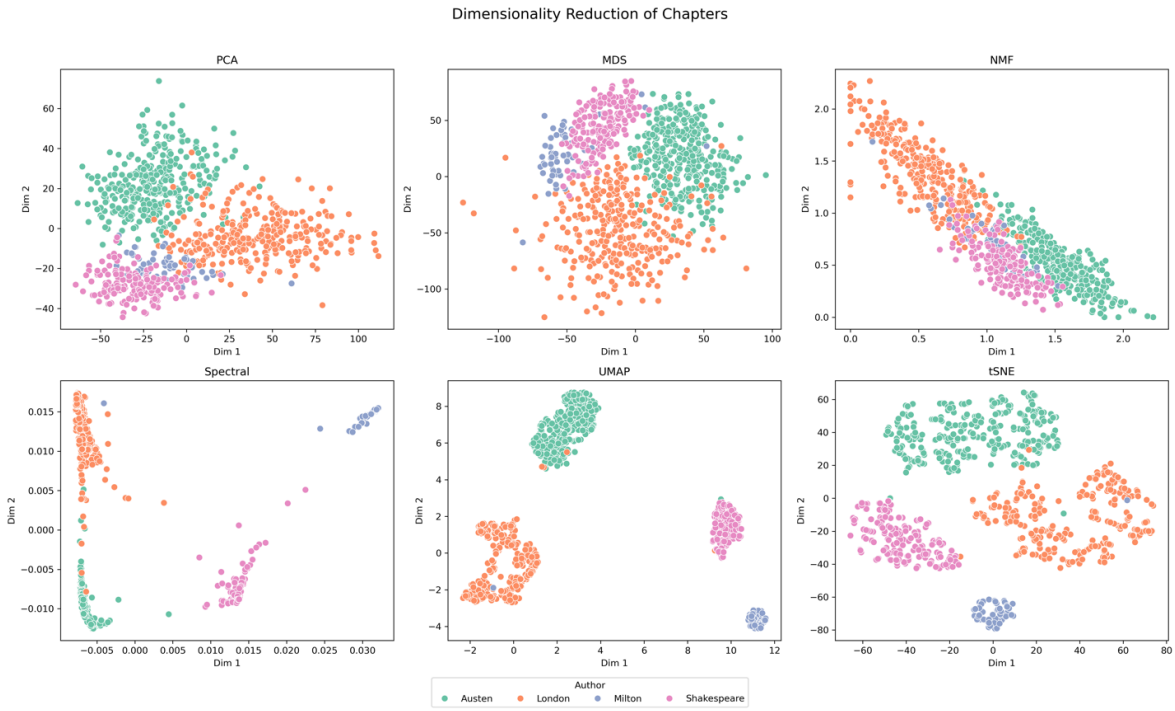
Visualizations

■ *observations (book chapters)*

Implementing PCA, MDS, NMF, Spectral Embedding, UMAP, t-SNE and validating with author labels (observations = chapters) gives the following visualizations.

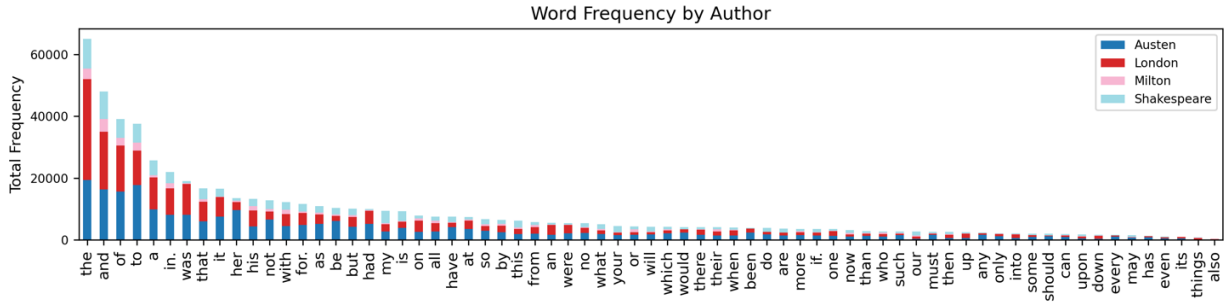


The linear methods perform worse in creating distinguishable clusters (without labels to validate PCA, MDS, and NMF would just look like "blobs" of observations) leading to more overlap and less informative projections; the structure of the data appears to be non-linear, evidenced by the superior performance of non-linear dimensionality reduction techniques. Among these, UMAP stands out—it produces well-separated, spherical clusters that significantly enhance interpretability and visual clarity. Compared to other non-linear approaches like t-SNE and Spectral Embedding, UMAP strikes a balance between global and local structure, revealing distinct author-based groupings.

In terms of interpretability, the visual above supports the claim that generally for this dataset (among the tested methods) the ***'best' interpretation = UMAP***.

■ *features (words)*

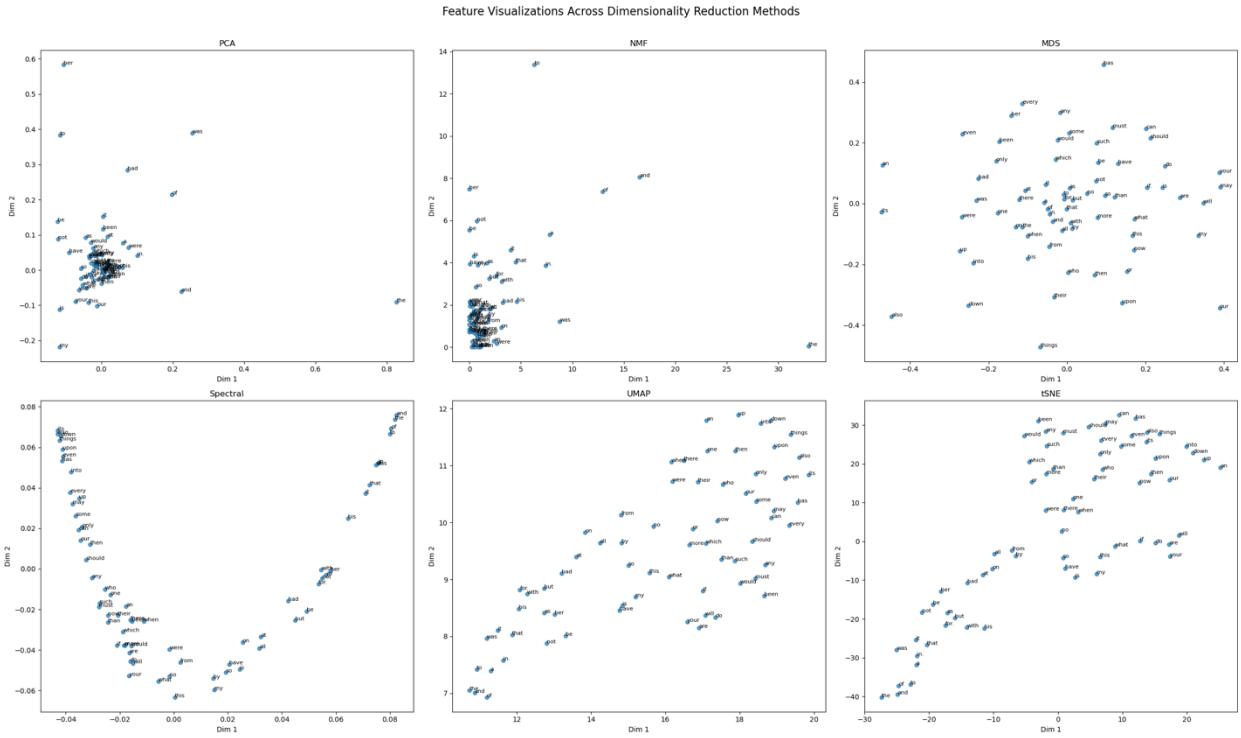
First off, the simplest visualization for the features/words is just a bar plot of word count across words!



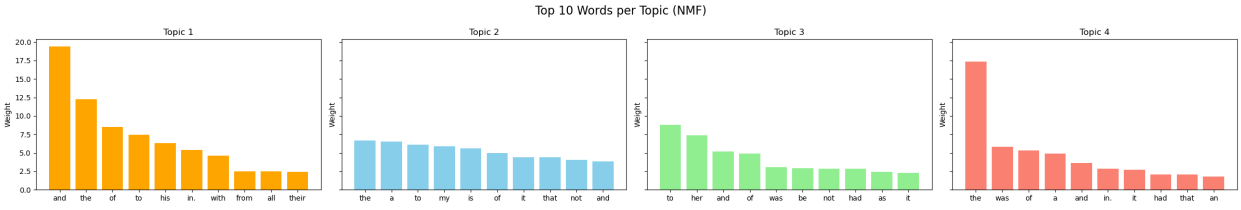
To further analyze patterns among features, I used their co-occurrences across chapters, without knowledge of authorship, to explore whether semantically or stylistically coherent word groups emerge. NMF is particularly well-suited for analyzing features (words) in an unsupervised

setting because it produces a parts-based, non-negative decomposition of the document-term matrix, allowing each topic to be represented as an additive combination of real words. This leads to intuitive and interpretable results, where the top contributing words for each topic reveal coherent semantic or stylistic themes—such as narrative tone, reflective voice, or dialogue. In contrast, methods like PCA yield components with mixed signs that are harder to interpret, and nonlinear methods like UMAP and Spectral Embedding embed words in low-dimensional space without preserving clear word-level semantics. Unlike these methods, NMF enables both dimensionality reduction and human-interpretable insights, making it the most effective choice for visualizing and understanding word-level structure in the data.

For example, visualizations on the features can be given by the following (below). Although this provides some information it is not evident and easily interpreted. So this method is **not** preferable for visual interpretation!



Going back to analyzing the NMF topics for semantic themes we can provide the following visualization of the top 10 words per topic (highest weight). This visualization in combination with the topic weights per chapter

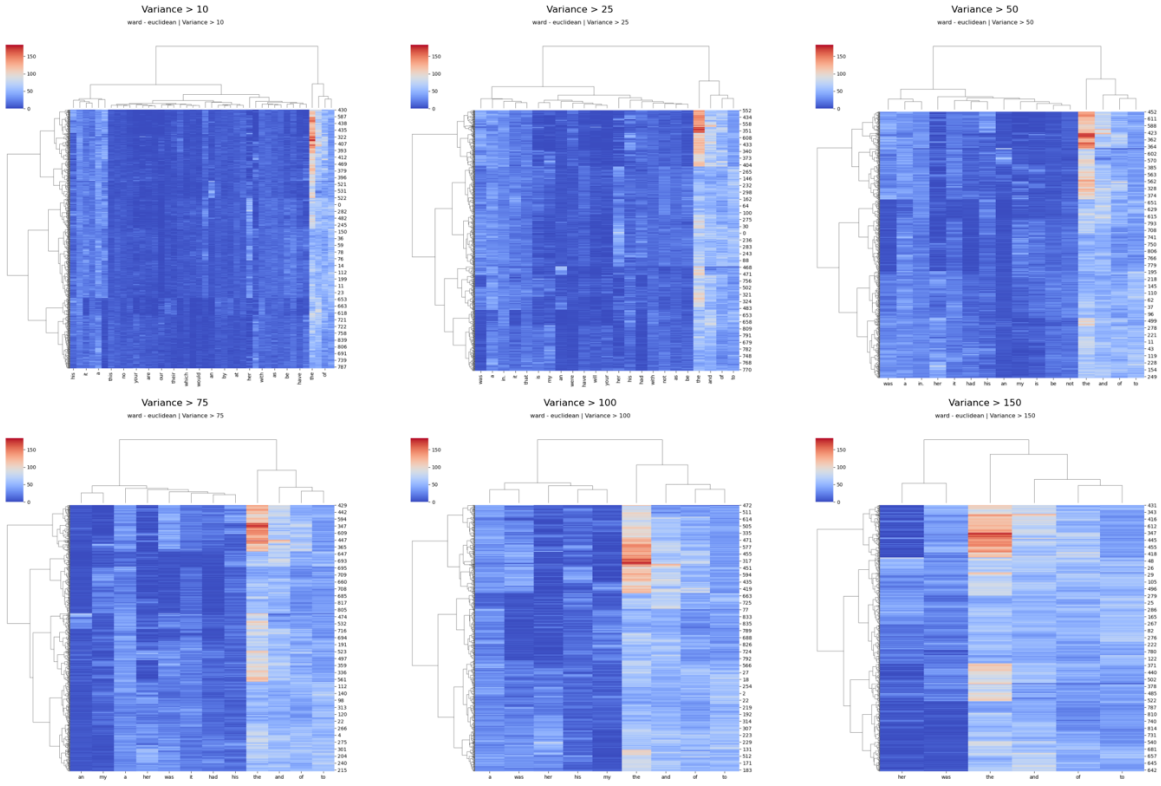


Using this it is far easier to establish semantic themes and distinguish the chapters from another according to which chapters are composed of which topics! The four topics extracted via NMF reveal distinct semantic patterns. **Topic 1** appears to reflect structural scaffolding of narrative prose, with high-weighted function words like *and*, *the*, *of*, *to*, and *with* suggesting continuous sentence construction, descriptive flow, and background exposition. **Topic 2** leans into a more introspective or active narrative stance, driven by frequent use of *my*, *is*, *to*, and *that*, pointing toward subjectivity, dialogue, or first-person accounts. **Topic 3** evokes personal and relational storytelling, emphasized by words such as *to*, *her*, *was*, *be*, and *had*, which are often found in emotionally driven or character-focused scenes. Lastly, **Topic 4** is anchored in definitive and declarative language—*the*, *was*, *and*, *had*, and *it*—potentially signaling reflective, philosophical, or climactic exposition. Together, these topic-word distributions highlight nuanced stylistic layers across authors, helping distinguish thematic patterns embedded in the text. This visual alone provides significant interpretation about the features.

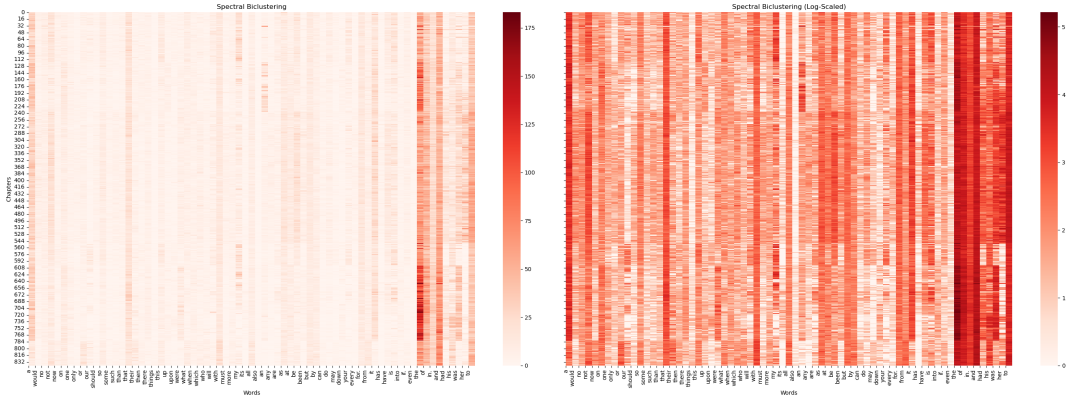
In terms of interpretability, the ***'best/great' interpretation*** is provided by ***NMF***. Now this leads to **1Ac** where we tie together these trends in words to the features which provides further value to interpretation!

■ *both - observations & features*

To visualize both observations and features I used Spectral Bi-clustering and Hierarchical Bi-clustering. To reduce noise in the heatmaps, I filtered out the highest variance features based on some threshold. I have plotted the features retained for each threshold across multiple thresholds below. The x-axis contains the words with high variance across chapters where significant chapters are plotted on the y-axis. If we look at the sparsest heatmap, for the highest threshold of variance > 150 (bottom right) we see bands emerge which provide significant interpretations. For example, if we go back to the NMF topics, for the word '*the'* we can see the red band for chapters 347 and 445 meaning it is likely that these chapters contain topic 1 or topic 4!



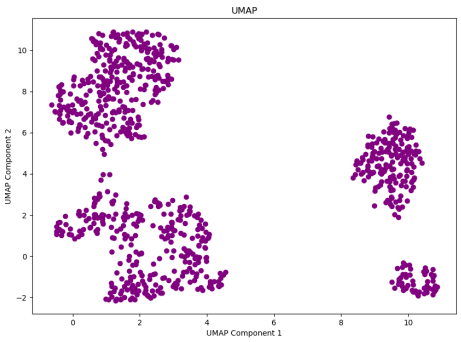
In other words, the features that have high contract among chapters allows us to interpret potential themes/topics among the chapters! I have also produced a heatmap using the Spectral Bi-clustering method which provided a decent visualization. However, clearly Hierarchical Bi-clustering outperforms this method.

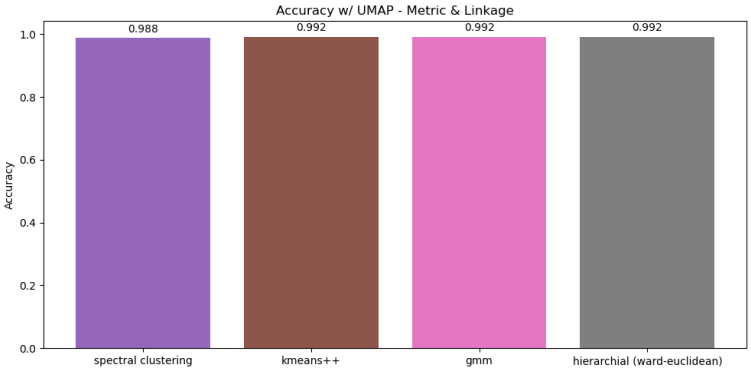


In terms of interpretability, the ***'best/great' interpretation*** is provided by ***Hierarchical Bi-clustering'***.

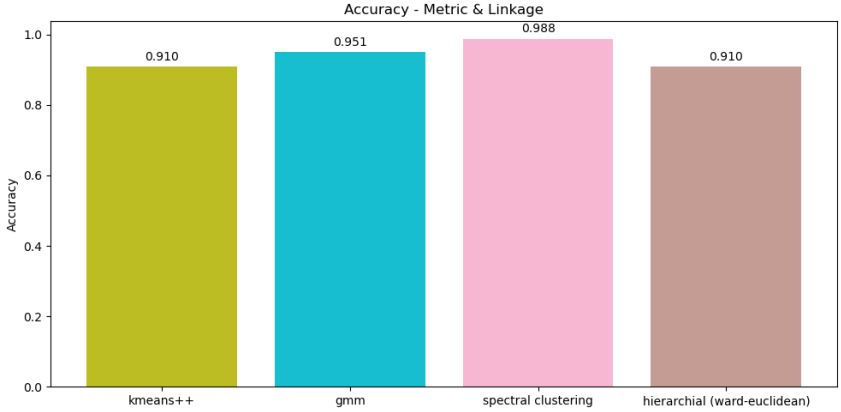
Clustering Methods

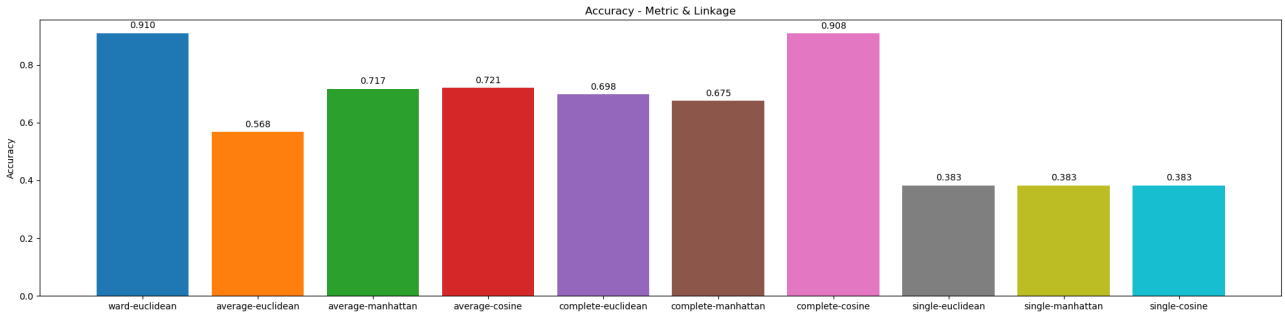
Since UMAP proved to be the superior dimension reduction technique for this dataset, I compared the clustering accuracy across methods on the UMAP transformed data (except for spectral clustering as this already has a built-in dimension reduction – spectral embedding). The results were not surprisingly the same across all methods at 0.992 because UMAP did such a good job at creating clusters that there were no points where the methods would diverge in predicting label outcomes! To make it evident, look at the UMAP data (without labels) – there are 4 clusters, and which points belongs to which clusters! Therefore, apply any methods to UMAP will yield the same labels for each point.





Now spectral clustering performs spectral embedding, so the method accuracy diverges from the methods processed on UMAP transformed data. Since UMAP dimension reduction made comparison redundant, I applied the same techniques to the raw dataset! Furthermore, I ran hierarchical clustering across many hyperparameter values for metrics and linkage.

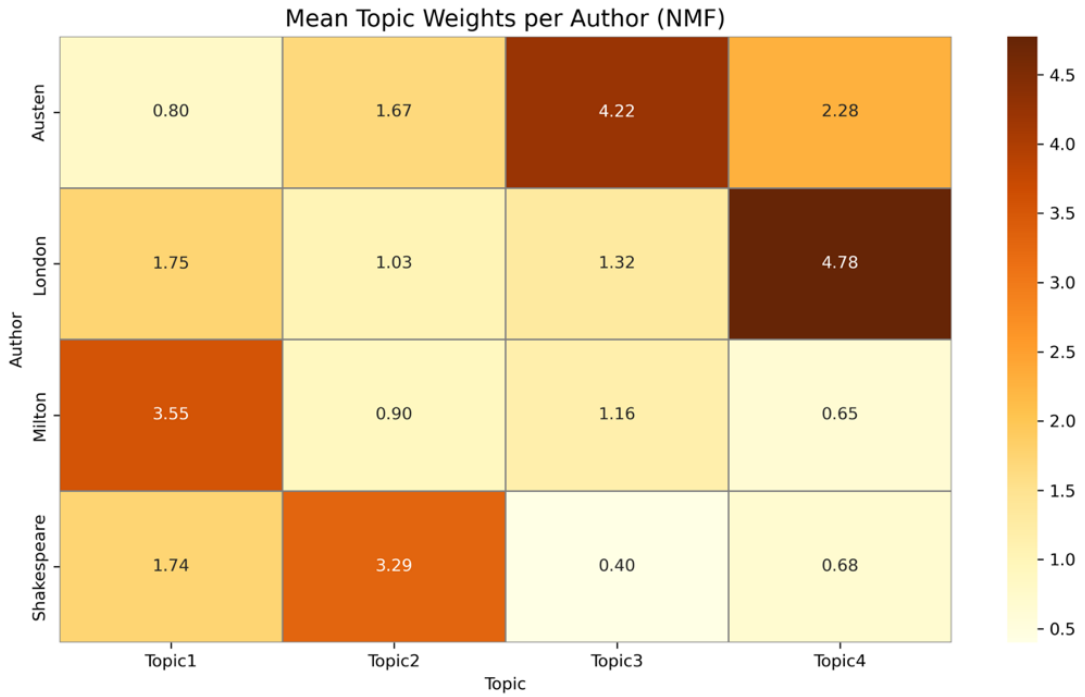




Without UMAP, performance dropped slightly yet still maintained strong results (of course spectral embedding yielded the same results). Among the metric and linkage parameters, Ward linkage with Euclidean distance yielded the highest accuracy, rivaling the performance of UMAP-based methods, while others like single-linkage performed poorly. Overall, UMAP significantly improves clustering accuracy and consistency, though the choice of metric and linkage remains critical when UMAP is not used

Pattern Recognition

Now using supervised learning to predict the author given the distribution of words per chapter! For this task I went back to the NMF topics because they revealed a lot about the dataset. I aimed to identify stylistic or feature-based patterns across the authors using an unsupervised learning approach. Recall that I applied NMF to uncover latent topics from the word frequency data, generating a document-topic matrix representing how strongly each chapter aligns with each topic. To determine which words were important for each topic, I visualized the top 10 words for each component, revealing clear semantic themes. Now using that work, I grouped by author and computed the mean topic weights and visualized them as a heatmap.



This shows that while no topic is exclusive to a single author, there are strong preferences — for instance, ***Milton* scores highest on Topic 1, *Shakespeare* on Topic 2, *Austen* on Topic 3, and *London* on Topic 4**. Since each author has a distinct #1 topic, I am inclined to predict the chapter based on this simple mapping of author to highest topic weight, i.e,

```
mapping = {'Topic1': 'Milton', 'Topic2': 'Shakespeare', 'Topic3': 'Austen', 'Topic4': 'London'}
```

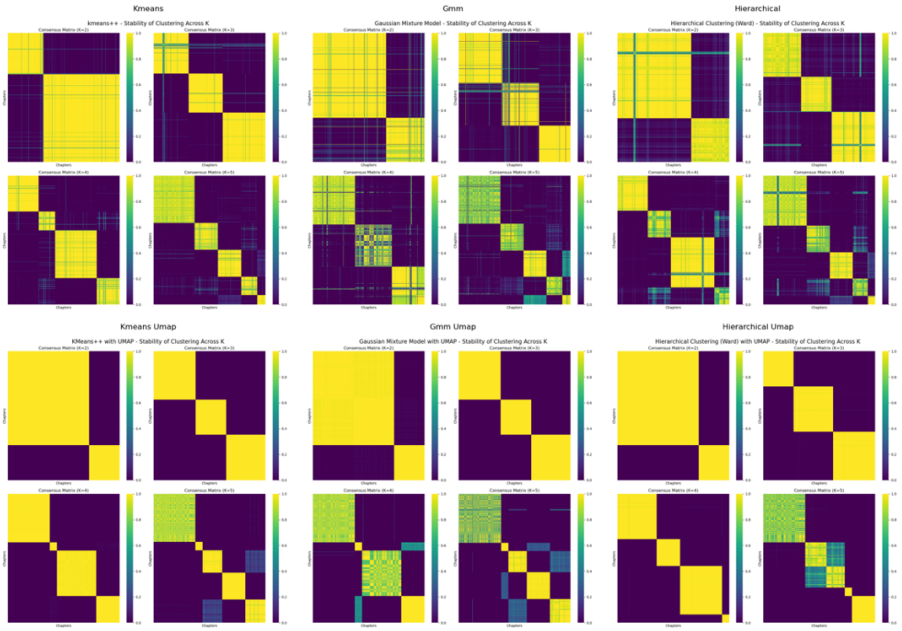
This representation suggests distinct stylistic or grammatical tendencies across authors. To evaluate how informative these patterns are, I split the data into **80% training** and **20% testing**, learned a mapping from dominant topic to author based on training data, and predicted authors on the test set using only the highest-weighted topic. This method correctly identified the author **91.7% of the time**, demonstrating that the unsupervised topic distributions captured meaningful, distinguishable patterns in writing style across authors. Now, we can claim that if the word distribution within a chapter aligns strongly with a topic, we can map that chapter to one of the known authors!

```
top_10_words_per_topic = {'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all', 'their'],
                          'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not', 'and'],
                          'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as', 'it'],
                          'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that', 'an']
}
```

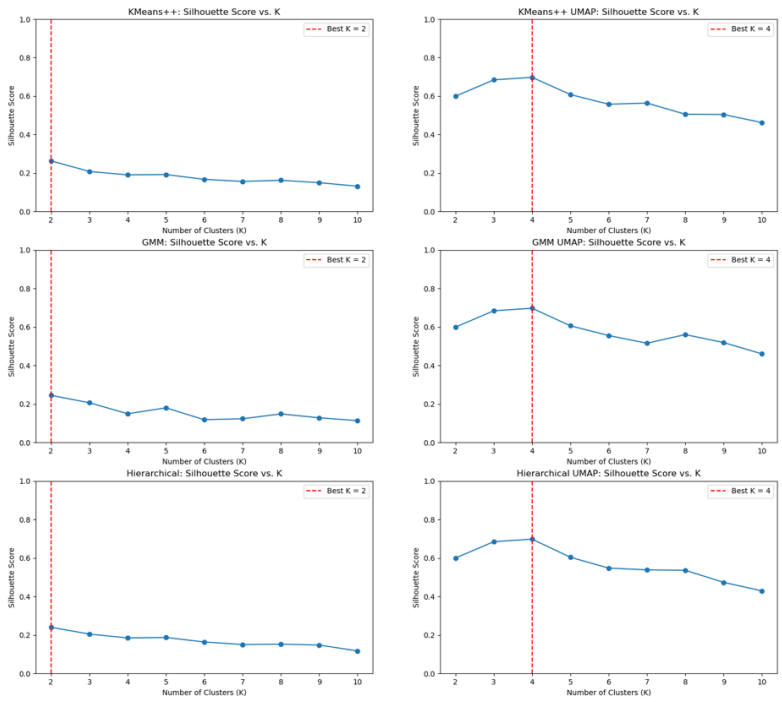
The words that are significant within topics and topics significant across authors provide predictive capabilities!

Validation

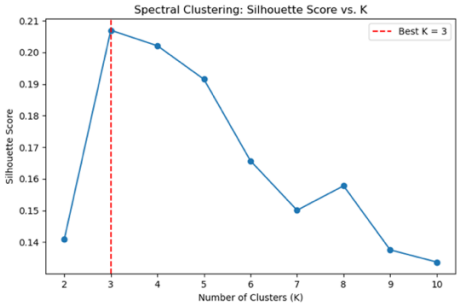
To validate our hyperparameter K without author labels, I used generalizability and stability as metrics.

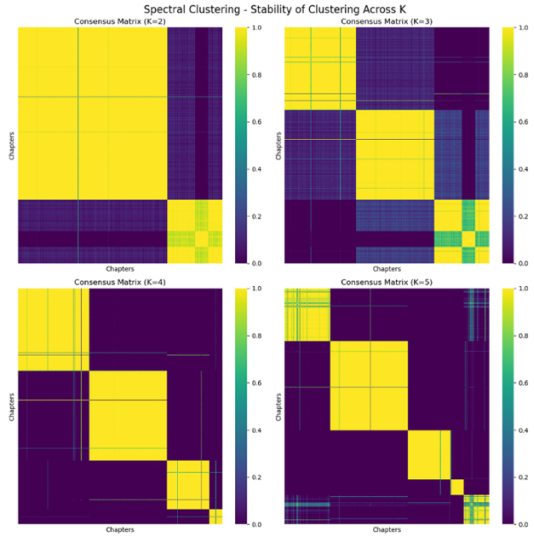


To gain insights to the generalizability, I calculated the silhouette scores across all methods (with and without the dimension reduction UMAP). From the visualization (right) you can see that before applying UMAP the silhouette score was very low across all K indicating but after applying UMAP, the silhouette score drastically increased and peaked at $K = 4$ across Kmeans++, GMM, and Hierarchial Clustering! To evaluate the stability, I used a consensus matrix, visualized above. GMM yielded a poor consensus matrix (likely due to problems in the initialization) but in combination with the silhouette scores we can confidently support the claim that $K = 4$! Furthermore, if we base our hyperparameter validation across all methods the we see a trend in stability score peaking at $K = 4$ and the consensus matrix showing strong stability with 4 clusters.



Spectral clustering can perform poorly under stability analysis because it is highly sensitive to small perturbations in the data or similarity matrix. Unlike centroid-based methods such as K-Means, which rely on explicit distance metrics and generally converge to consistent partitions, spectral clustering depends on the eigen structure of a graph Laplacian derived from data affinities. Even slight changes in the dataset can significantly alter the eigenvectors, leading to different cluster assignments across runs. As a result, traditional stability metrics may underestimate the effectiveness of spectral clustering, not due to poor clustering performance, but due to the method's inherent sensitivity to initialization and graph construction parameters. This is demonstrated below, as the silhouette scores all appeared very low and that it "peaked" at $K = 3$ but very marginally. Inspecting the $K = 3$ consensus matrix makes it clear that the stability for $K = 3$ is very poor and so it makes more sense to select the next highest silhouette score at $K = 4$ (which is essentially the same value as the stability score for $K = 3$). This deduction aligns with all the other methods.





In conclusion, Hierarchial Clustering and Kmeans++ are the most stable and generalize the best for this dataset! The results of analyzing the stability and generalizability lead to a clear hyperparameter of $K = 4$ clusters.

EM Algorithm

The EM algorithm for the Multivariate Gaussian Mixture Model

Start - Initialize

$\mu_k^{(0)} \sim$ Uniform sample from $\{x_1, \dots, x_N\}$, $\forall k \in \{1, \dots, K\}$

$\Sigma_k^{(0)} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$, $\forall k \in \{1, \dots, K\}$

$\pi_k^{(0)} = \frac{1}{K}$, $\forall k \in \{1, \dots, K\}$

E-step

$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$ where $\mathcal{N}(x_i \mid \mu_j, \Sigma_j)$ is the multivariate Gaussian density $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik} x_i$

Update covariance: $\Sigma_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

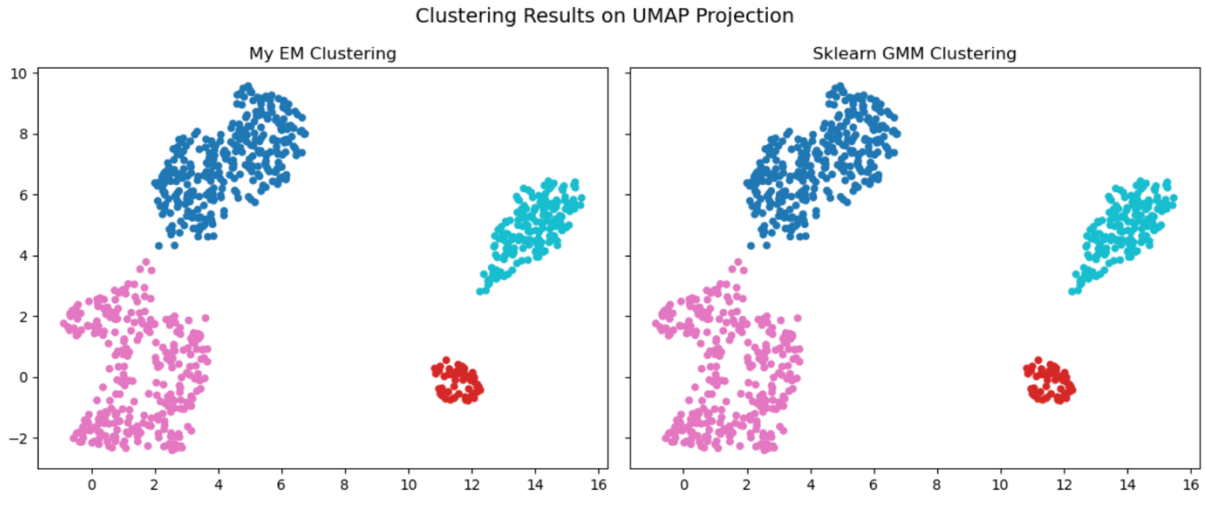
Update mixing weights: $\pi_k = \frac{N_k}{N}$

Running this on the authors algorithm on the 00_authors.csv dataset with the following code (fit_slow uses for loops and then I vectorized it using numpy for speed into fit_fast). See code here (alternatively I have attached a screenshot below).

```python
class GaussianMixtureEM:
    def __init__(self, K, num_iterations, allow_singular=True):
        self.K = K
        self.num_iterations = num_iterations
        self.allow_singular = allow_singular

    def fit_slow(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # E-step
            for i in range(n_rows):
                xi = X.iloc[i].values
                denom = 0
                for k in range(self.K):
                    numerator = pis[k] * multivariate_normal.pdf(xi, mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                    gamma[i, k] = numerator
                    denom += numerator
                gamma[i, :] /= denom

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]
            means = [np.sum([gamma[i, k] * X_array[i, :] for i in range(n_rows)],axis=0) / Nk[k] for k in range(self.K)]
            cov = [np.sum([gamma[i, k] * np.outer(X_array[i, :] - means[k], X_array[i, :] - means[k]) for i in range(n_rows)],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]
            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma
        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}

    def fit_fast(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # Vectorized E-step
            log_pdf_matrix = np.zeros((n_rows, self.K))
            for k in range(self.K):
                rv = multivariate_normal(mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                log_pdf_matrix[:, k] = np.log(pis[k] + 1e-12) + rv.logpdf(X_array)

            max_log = np.max(log_pdf_matrix, axis=1, keepdims=True)
            log_gamma = log_pdf_matrix - max_log
            gamma = np.exp(log_gamma)
            gamma /= gamma.sum(axis=1, keepdims=True)

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]

            means = [np.sum(gamma[:, k][:, np.newaxis] * X_array,axis=0) / Nk[k]for k in range(self.K)]

            cov = [np.sum(gamma[:, k][:, np.newaxis, np.newaxis] *(X_array - means[k])[:, :, np.newaxis] @ (X_array - means[k])[:, :,np.newaxis],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]

            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma

        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}
```

The results of my EM-algorithm against Sklearn EM-algorithm for GMM on the authors data set after applying UMAP yielded the same results!

```
Adjusted Rand Index: 1.0000
```



Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!

01_linear_dimension_reductionV2

April 17, 2025

Download the necessary libraries.

Link to data - https://raw.githubusercontent.com/DataSlingers/clustRviz/master/data/authors.rda

Install these if necessary.

[48]:
```
# %pip install scikit-learn --quiet
# %pip install adjustText --quiet
# %pip install umap-learn --quiet
# %pip install wordcloud
```

[49]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.decomposition import FastICA, NMF, KernelPCA, PCA, TruncatedSVD
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE
from sklearn.model_selection import train_test_split
from sklearn.manifold import MDS

## Msc
from adjustText import adjust_text
from itertools import combinations
from wordcloud import WordCloud
```

Load dataset.

[50]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values
df
```

[50]:
```
           Author   a  all  also  an  and  any  are  as  at  …  was  were  \
0          Austen  46   12     0   3   66    9    4  16  13  …   40    11
1          Austen  35   10     0   7   44    4    3  18  16  …   27    13
2          Austen  46    2     0   3   40    1   13  11   9  …   24     6
3          Austen  40    7     0   4   64    3    3  20  13  …   26    10
4          Austen  29    5     0   6   52    5   14  17   6  …   23     5
..            …  ..  …   …  ..  …   …   …  ..  ..  …  …   …
836  Shakespeare  32    4     0   6   33    0    7   8   4  …    0     1
837  Shakespeare  16    5     0   5   49    1    6  10   3  …    1     1
838  Shakespeare  22   15     0   3   48    0    9  10   2  …    4     0
839  Shakespeare  25    4     0   8   59    3    6   7   3  …    3     4
840  Shakespeare  26    4     0   2   62    0    4   7   4  …    5     0

     what  when  which  who  will  with  would  your
0       7     5      6    8     4     9      1     0
1       5     7      7    3     5    14      8     0
2      10     4      6    4     5    15      3     9
3       3     6     10    5     3    22      4     3
4       8     4     13    2     4    21     10     0
..    …   …     …  …   …   …     …   …
836    13     2      3    3    11    17      5    10
837     6     5      6    0    11    20      2     7
838    16     2      2    0    12    15      1    10
839    11     2      2    2    22    23      4     5
840    13     2      5    3    11    19      0     3

[841 rows x 70 columns]
```

[51]:
```
book_id = df['Author']
book_id.value_counts() # 4 different books ; w/ 317 - Austen, ..., 55 - Milton.
```

[51]:
```
Author
Austen         317
London         296
Shakespeare    173
Milton          55
Name: count, dtype: int64
```

- Unsupervised learning: drop columns [‘Authors’] and determine patterns with words across chapters using unsupervised learning methods.
- We will later come back to these labels we dropped to validate our results.
- Note, a row represents a book chapter with each column representing the word counts of key words in that chapter.

1 Linear Methods

1.1 PCA

PCA Theory

Given a centered data matrix $X \in \mathbb{R}^{n \times p}$ with: - n: observations (chapters), - p: features (words),

PCA uses the SVD: $X = U\Sigma V^\top$

- $U\Sigma \rightarrow$ principal component scores (embedding of chapters),
- V (or `pca.components_`) $\rightarrow$ principal axes (directions for words).

No need to transpose the data to get word embeddings.
Using `fit_transform(X)` for chapters, and `components_.T` for words — where both come from the same PCA fit.

Also note that when fitting a PCA in this dataset: **WE SHOULD NOT SCALE DATA SINCE IT IS WORD COUNT!!!**

1.1.1 Observations

[52]:
```
pca = PCA()
X_pca = pca.fit_transform(X)
word_loadings = pca.components_.T

pca_df = pd.DataFrame(X_pca)
cols = [f'PC{j+1}' for j in range(pca_df.shape[1])]
pca_df = pca_df.rename(columns = {i:cols[i] for i in range(pca_df.shape[1])})
pca_df
```

[52]:

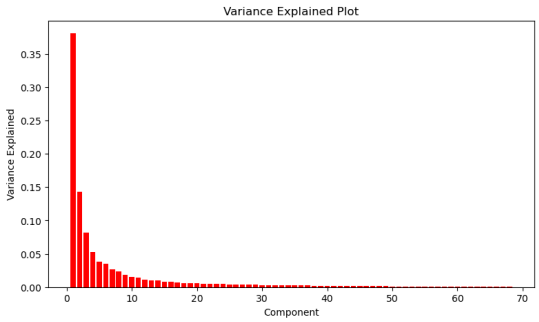
	PC1	PC2	PC3	PC4	PC5	PC6	\
0	-2.265044	43.499301	5.196950	-2.333575	23.359407	22.309224	
1	-2.604648	25.086417	-9.488717	7.748273	19.244916	1.052493	
2	-33.199533	8.667765	-14.833418	3.971572	-6.913595	4.173856	
3	8.098653	21.760546	6.962558	6.683136	15.262020	7.640936	
4	10.031814	6.801164	0.035520	22.731646	10.465248	-5.248171	
..	…	…	…	…	…	…	
836	-64.400961	-28.132705	-21.267908	0.131106	-0.337083	12.068621	
837	-58.313001	-23.417273	4.887866	2.462693	-1.275957	-2.603434	
838	-47.898865	-31.938566	-7.364020	-9.133520	10.442357	14.851150	
839	-39.844905	-29.936659	0.614003	-2.261258	5.007836	14.369278	
840	-34.807687	-38.141056	6.289341	-2.114649	1.435612	13.845645	

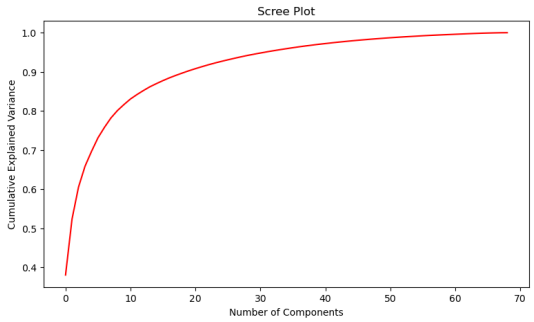
	PC7	PC8	PC9	PC10	…	PC60	PC61	\
0	13.585777	-0.891477	-2.178907	0.062718	…	-2.043904	-3.175850	
1	6.765152	3.650037	-0.746511	-1.753312	…	-2.269775	0.991903	
2	7.094662	-11.728066	1.002140	-4.270366	…	-2.055041	0.864406	
3	11.974431	-5.748289	-1.011280	-4.502115	…	0.590874	0.830834	
4	7.489147	-6.171808	-9.885234	-14.616535	…	1.616164	2.706626	
..	…	…	…	…	…	…	…	

```
836  -5.848328   3.920184 -7.233151   0.239586  … -0.089364 -0.674634
837 -11.452006   4.745220  7.397146  -1.948260  … -0.264725 -1.237238
838 -10.872174  10.228351 -5.141056   3.876664  …  0.938558  1.771321
839  -3.165040   3.620264  1.479972   2.361646  … -0.011889  1.010582
840   3.620534   3.312984 -2.660270   0.521845  …  1.273284  4.486729

         PC62      PC63      PC64      PC65      PC66      PC67      PC68  \
0    1.578725 -2.575981  1.463553 -1.107755 -0.341962  2.639943  3.214014
1    0.013898  1.037391 -1.077602  0.044514 -1.312650  0.800142  0.183083
2   -2.179206  2.564626 -3.672920  0.798756 -0.664101 -1.319190 -0.493611
3   -1.969229  0.937902 -3.603491 -1.244561  1.283435 -0.912942 -0.271062
4   -0.708168 -1.272716 -0.278721 -1.525001  1.746847 -1.004067 -1.272581
..        …         …         …         …         …         …         …
836 -1.165301 -0.470178 -1.348415 -0.240363  0.167233  0.514576 -0.123753
837 -0.170168  1.335993  0.455795  0.942986 -0.266152 -2.256417 -0.182997
838  1.542836 -1.795202  1.242780  2.648138 -0.017259 -2.002290  0.787364
839  1.514307 -0.011713  1.135751  0.803847 -0.028491 -0.803976 -1.307802
840  3.058903  0.497595  0.257034 -0.099739 -0.201420 -0.168980 -0.437193

         PC69
0   -0.594109
1   -0.194564
2    0.165242
3   -0.743841
4   -0.229321
..        …
836 -0.036760
837 -0.021365
838  0.071339
839 -0.026705
840 -0.209892

[841 rows x 69 columns]
```

[53]:
```
fig, ax = plt.subplots(1,2, figsize=(20,5))

ax[0].bar(np.arange(1, pca_df.shape[1]),pca.explained_variance_ratio_[0:pca_df.
 ↪shape[1]-1], color = 'red')
ax[0].set_xlabel('Component')
ax[0].set_ylabel('Variance Explained')
ax[0].set_title('Variance Explained Plot')

ax[1].plot(np.cumsum(pca.explained_variance_ratio_), color = 'red')
ax[1].set_xlabel('Number of Components')
ax[1].set_ylabel('Cumulative Explained Variance')
ax[1].set_title('Scree Plot')
```

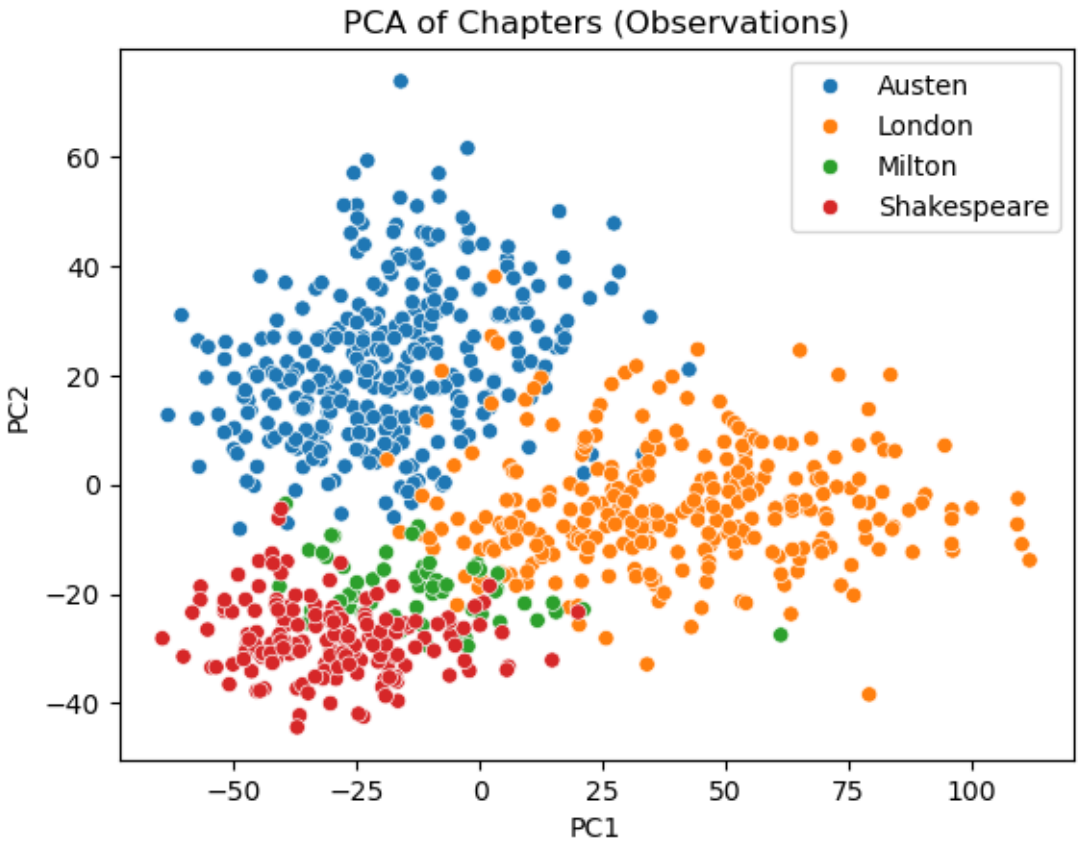
```
plt.savefig('Media/viz/01/01_pca_var_explained_and_screeplot')
plt.show()

print(f'PC1 and PC2 explain {np.sum(pca.explained_variance_ratio_[0:2])*100:.
  ↪3f}% of the variance.')
```






```
PC1 and PC2 explain 52.396% of the variance.
```

[54]:
```
pca_chapters_df = pca_df.loc[:,['PC1','PC2']] # Nested and ordered; extract␣
  ↪first 2 Principal Components
pca_chapters_df['Author'] = df['Author']

sns.scatterplot(data=pca_chapters_df, x='PC1', y='PC2', hue='Author',␣
  ↪palette='tab10')
plt.title('PCA of Chapters (Observations)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.savefig('Media/viz/01/01_pca_chapters')
plt.show()
```



1.1.2 Features

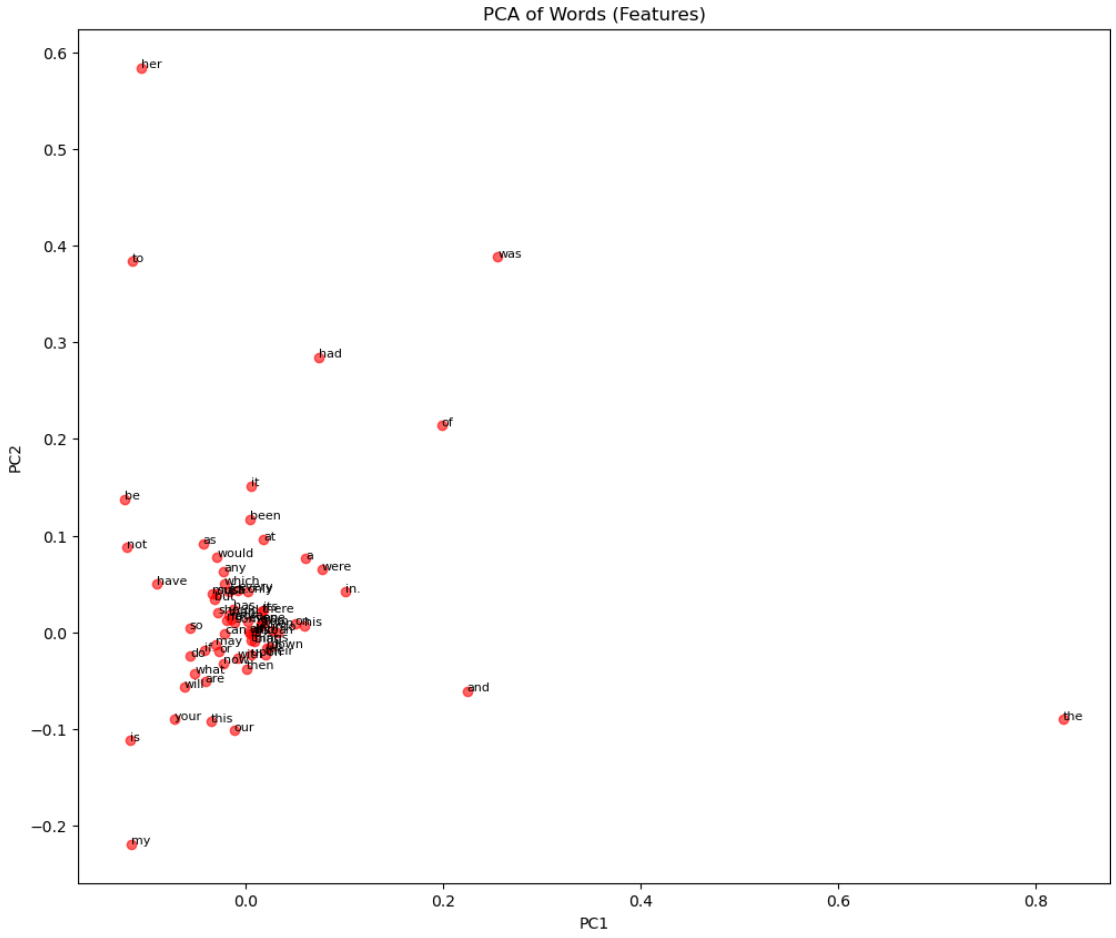
[55]:
```
pca_words_df = pd.DataFrame(word_loadings)
cols = [f'PC{j+1}' for j in range(pca_words_df.shape[1])]
pca_words_df = pca_words_df.rename(columns = {i:cols[i] for i in␣
 ↪range(pca_words_df.shape[1])})
pca_words_df.index = X.columns
pca_words_df
```

[55]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	\
a	0.060676	0.076522	-0.315737	-0.055225	-0.078926	0.263739	0.484181	
all	0.003003	0.000487	0.053015	-0.032618	0.066769	-0.029175	-0.009633	
also	0.007791	-0.000515	0.007859	0.001355	-0.008722	0.000714	0.000660	
an	0.032094	-0.000843	-0.326638	-0.223061	0.188812	-0.163141	0.137205	
and	0.223970	-0.060692	0.667811	-0.081825	-0.015979	0.543313	-0.035569	
…	…	…	…	…	…	…	…	
who	0.003630	-0.001030	0.052789	0.002280	0.002378	-0.013840	-0.007167	
will	-0.062244	-0.056680	-0.001381	0.124291	0.004152	0.024512	-0.127001	
with	-0.008282	-0.026201	0.128355	-0.102281	0.099969	-0.044243	0.033596	
would	-0.029127	0.078012	-0.025317	0.047541	-0.050614	0.034905	-0.062669	

```
your  -0.072387 -0.090066 -0.053952  0.107079 -0.015571  0.009243 -0.028564

             PC8       PC9      PC10  …      PC60      PC61      PC62  \
a      -0.525683  0.264041  0.219213  … -0.005026  0.002127 -0.020590
all     0.042661  0.058567 -0.200501  …  0.007117 -0.043006  0.021078
also    0.007541  0.003267 -0.006968  … -0.002700  0.018310 -0.007202
an      0.060527  0.235075 -0.152501  … -0.012955 -0.026053 -0.005756
and    -0.145092  0.219987 -0.003553  … -0.002065  0.003563  0.004656
…          …        …        …  …        …         …         …
who    -0.006857  0.004959 -0.064097  … -0.030093 -0.166349  0.078814
will   -0.070483 -0.042921  0.024465  …  0.029968  0.013908  0.020761
with   -0.015710  0.041990  0.047793  … -0.006375 -0.009951  0.001977
would  -0.037895 -0.032398 -0.054165  … -0.024179 -0.008643 -0.011250
your    0.019719  0.015591  0.159339  …  0.010928 -0.015083 -0.011472

            PC63      PC64      PC65      PC66      PC67      PC68      PC69
a      -0.020622 -0.005487 -0.001468  0.006522 -0.002964  0.020486  0.001386
all     0.017012 -0.001458  0.008575  0.012527  0.033792 -0.014598  0.001157
also   -0.008381 -0.001825  0.006400 -0.019838 -0.093516  0.056643  0.989992
an      0.020056  0.008385 -0.027075  0.000523  0.005918  0.007497  0.005771
and    -0.002794  0.001242 -0.004503  0.005302 -0.000100  0.000272 -0.005046
…          …         …         …         …         …         …         …
who     0.125654 -0.021114 -0.080313  0.096907  0.061265 -0.000161 -0.017311
will    0.013775  0.006232 -0.059988 -0.017531 -0.003613  0.022391 -0.003819
with   -0.007391  0.009002  0.006421 -0.005216  0.012884  0.008480 -0.000474
would   0.021813  0.022624 -0.031247  0.009858 -0.018687 -0.006684 -0.001789
your   -0.023488 -0.021720  0.023567  0.006809 -0.026835  0.019442 -0.000493

[69 rows x 69 columns]
```

[56]:
```python
plt.figure(figsize=(12, 10))
plt.scatter(pca_words_df['PC1'], pca_words_df['PC2'], alpha=0.6, color = 'red')

for _, row in pca_words_df.iterrows():
    plt.text(row['PC1'], row['PC2'], row.name, fontsize=8)  # ← fixed here
plt.title('PCA of Words (Features)')
plt.xlabel('PC1')
plt.ylabel('PC2');
plt.savefig('Media/viz/01/01_pca_words')
```



This method offers poor visualization and little interpretability. There are other methods that will provide more interpretability on the features such as NMF.

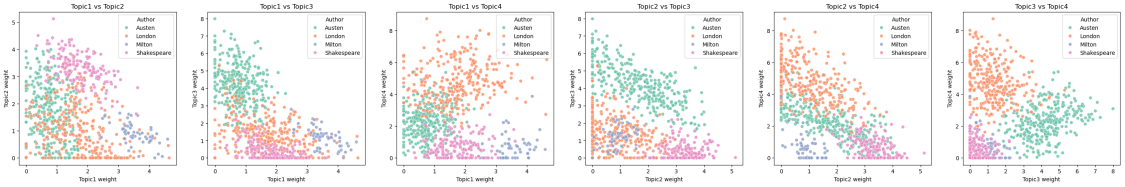
1.2 NMF

NMF may perform strongly here because our word frequencies are non-negative! Furthermore, this method will provide semantic intuition by creating topics.

[57]:
```
k = 4 # hyperparameter -> clusters
model = NMF(n_components=k, init='random', random_state=0, max_iter=500)

# dimensions -> (841 chapters, k topics)
W = model.fit_transform(X)

# dimensions -> (k topics, 69 words)
H = model.components_
```

1.2.1 Observations

[58]:
```
W_df = pd.DataFrame(W, index=X.index, columns=[f'Topic{i+1}' for i in␣
↪range(k)]) # Turn W into a DataFrame with word labels
topic_df = W_df.reset_index(drop=True)
topic_df["Author"] = df["Author"].reset_index(drop=True)
topics = W_df.columns.tolist() # List of topic names
topic_pairs = list(combinations(topics, 2)) # All pairs of topics

n_plots = len(topic_pairs)
fig, axes = plt.subplots(nrows=1, ncols=n_plots, figsize=(6 * n_plots, 5))

if n_plots == 1:
    axes = [axes]

for i, (x_topic, y_topic) in enumerate(topic_pairs):
    ax = axes[i]
    sns.scatterplot(data=topic_df, x=x_topic, y=y_topic, hue="Author",␣
↪palette="Set2", alpha=0.8, ax=ax)
    ax.set_title(f"{x_topic} vs {y_topic}")
    ax.set_xlabel(f"{x_topic} weight")
    ax.set_ylabel(f"{y_topic} weight")
    ax.legend().set_title("Author")

plt.savefig('Media/viz/01/01_nmf_observations')
```



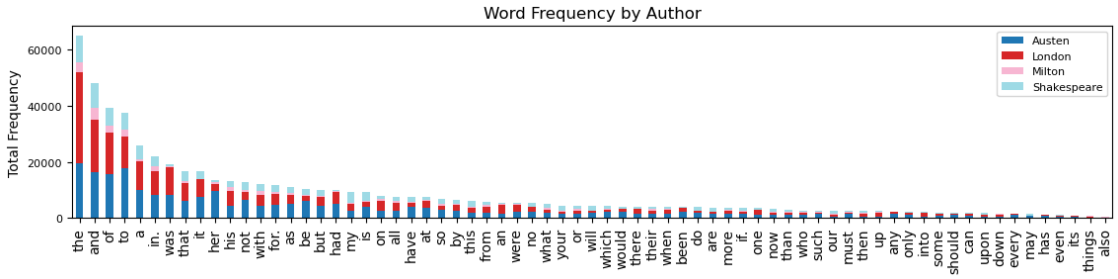
This provides poor interpretability so it is clearly not our first choice when visualizing chapters/observations.

1.2.2 Features

First of all the most simple and easy/inuitive vizualization of the features/words is just a simple total count!

[59]:
```
X_df = X.copy()
X_df['Author'] = authors
author_word_freq = X_df.groupby('Author').sum().T
author_word_freq = author_word_freq.loc[author_word_freq.sum(axis=1).
↪sort_values(ascending=False).index]
```

```
fig, ax = plt.subplots(figsize=(len(author_word_freq)/6, 3))  # adjust based on␣
 ↪# of words
author_word_freq.plot(
    kind='bar',
    stacked=True,
    ax=ax,
    colormap='tab20'
)

plt.title("Word Frequency by Author", fontsize=12)
plt.ylabel("Total Frequency", fontsize=10)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=8)
plt.legend(loc='upper right', fontsize=8)
plt.tight_layout()
plt.savefig("Media/viz/01/01_all_word_freq_by_author_stackedbar_condensed.png",␣
 ↪dpi=300)
plt.show()
```



This is where NMF will truly shine.

[60]:
```
n = 10 # Set here -> top words per topic!
```

[61]:
```
word_topic_df = pd.DataFrame(H.T, index=X.columns, columns=[f"Topic{i+1}" for i␣
 ↪in range(H.shape[0])]) # Transpose H to get words as rows, topics as columns
columns = list(word_topic_df.columns)
highest_n_weighted_words_per_topic = {}
for col in columns:
    words = list(word_topic_df[col].sort_values(ascending=False).head(n).index)
    highest_n_weighted_words_per_topic[col] = words
print(highest_n_weighted_words_per_topic)
```

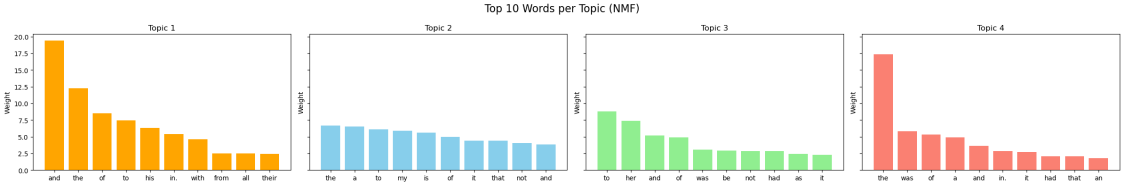
```
{'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all',
'their'], 'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not',
'and'], 'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as',
'it'], 'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that',
'an']}
```

Below are semantic interpretations of each topic based on the top contributing words:

- **Topic 1 – Structural & Possessive Language**
  Emphasizes grammatical connectors, possession, and sentence scaffolding — likely reflecting narrative structure or formal exposition.
- **Topic 2 – Determiners & Negation**
  Centers on articles, pronouns, and simple negations, suggesting basic sentence formation and assertive language.
- **Topic 3 – Past-Tense Narration**
  Highlights auxiliary verbs and past-tense forms, indicating descriptive or event-driven storytelling.
- **Topic 4 – Temporal & Descriptive Grammar**
  Focuses on narrative tense, articles, and function words often used in unfolding sequences or descriptive prose.

These topics reflect broad grammatical and stylistic features common in text, helping differentiate author styles or narrative structures.

[62]:
```
fig, axes = plt.subplots(1, H.shape[0], figsize=(6 * H.shape[0], 4),␣
 ↪sharey=True)

colors = ['orange', 'skyblue', 'lightgreen', 'salmon']  # or however many␣
 ↪topics you have
for topic_idx, ax in enumerate(axes):
    topic_weights = H[topic_idx]
    top_word_indices = topic_weights.argsort()[::-1][:n]
    top_words = [X.columns[i] for i in top_word_indices]
    top_scores = topic_weights[top_word_indices]
    ax.bar(top_words, top_scores, color=colors[topic_idx % len(colors)])
    ax.set_title(f"Topic {topic_idx + 1}")
    ax.set_ylabel("Weight")

fig.suptitle(f"Top {n} Words per Topic (NMF)", fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/01/01_nmf_top_words_per_topic')
```



Extra Feature Vizuals (not scientific)

[63]:
```
word_freq_dict = X.sum(axis=0).to_dict()

wordcloud = WordCloud(width=1000, height=500, background_color='white').
 ↪generate_from_frequencies(word_freq_dict)

plt.figure(figsize=(15, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of All Features", fontsize=16)
plt.savefig('Media/viz/01/01_word_cloud')
plt.show()
```

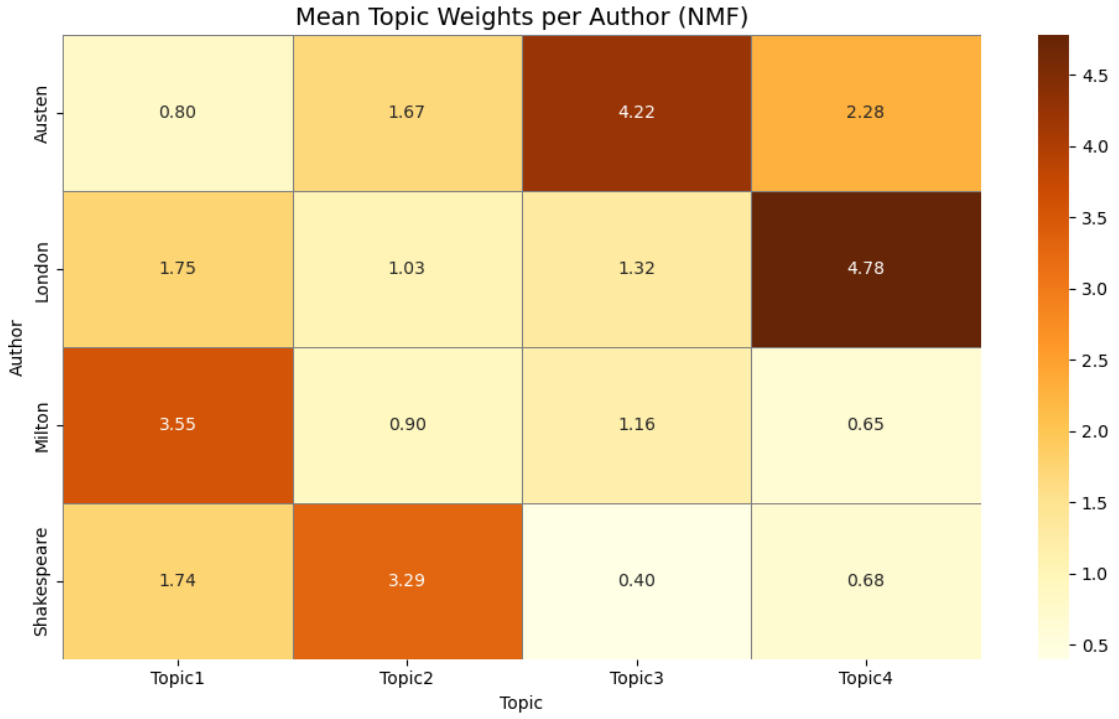
Now let us try and use this to determine which chapters correspond to which topics and predict the author based on the semantics of each topic.

Establish a mapping by grouping by using the labels, grouping by Author and then evaluating the mean topic weights per author! First we should split our data into a training and testing.

[64]:
```
W_df['Authors'] = df['Author']

# Split the data
train_df, test_df = train_test_split(W_df, test_size=0.2, random_state=42,␣
 ↪stratify=W_df['Authors'])

print(f"Training samples: {len(train_df)}")
print(f"Testing samples: {len(test_df)}")
```

```
Training samples: 672
Testing samples: 169
```

[65]:
```
author_topic_means = train_df.groupby('Authors').mean() # Compute mean topic␣
 ↪weights grouped by author
plt.figure(figsize=(10, 6))
sns.heatmap(author_topic_means, annot=True, fmt=".2f", cmap="YlOrBr",␣
 ↪linewidths=0.5, linecolor='gray')
plt.title("Mean Topic Weights per Author (NMF)", fontsize=14)
plt.xlabel("Topic")
plt.ylabel("Author")
plt.tight_layout()
plt.savefig("Media/viz/01/01_nmf_author_topic_heatmap", dpi=300)
```



Mean Topic Weights per Author (NMF)

The heatmap above shows the average topic weights across all chapters written by each author. It is computed by grouping the NMF document-topic matrix (`W`) by author and taking the mean.

Interpretation: - Each cell reflects how strongly a given author tends to express a specific topic. - Higher values indicate that the author frequently uses patterns or structures associated with that topic. - While no topic is exclusive to a single author, we observe distinct preferences: - **Milton** leans heavily on Topic 1 - **Shakespeare** shows strong usage of Topic 2 - **Austen** favors Topic 3 - **London** stands out on Topic 4

This unsupervised representation helps reveal stylistic or grammatical tendencies across authors.

[66]:
```
mapping = {'Topic3': 'Austen', 'Topic4': 'London', 'Topic1': 'Milton', 'Topic2':
 ↪ 'Shakespeare'}
```

Now lets evaluate this mapping on the test set!

[67]:
```
test_df['Highest Weighted Topic'] = test_df.iloc[:, :4].idxmax(axis=1).values
test_df['Likely Author'] = test_df['Highest Weighted Topic'].apply(lambda x:␣
 ↪mapping[x])
correct_proportion = (test_df['Likely Author'] == test_df['Authors']).mean()
print(f'This method was correct {correct_proportion*100:.4f}% of the time')
```

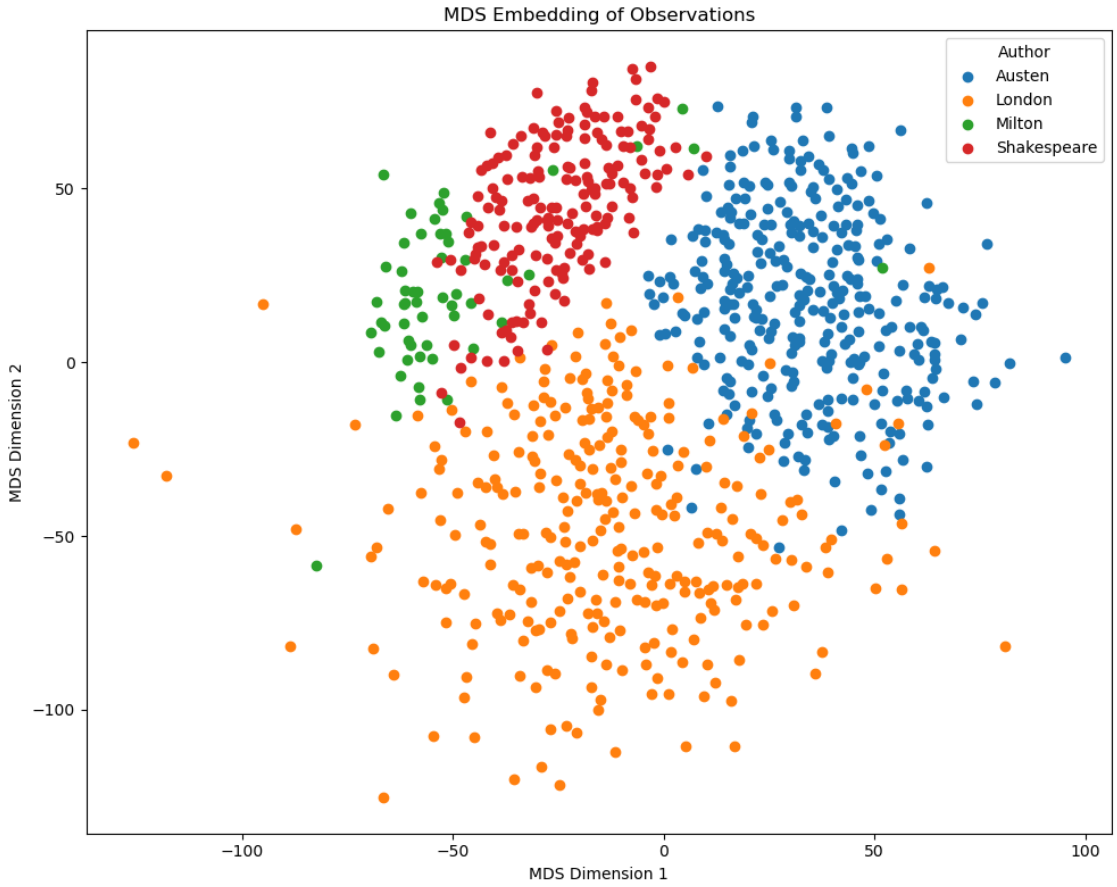
```
This method was correct 91.7160% of the time
```

This yields pretty good results!!

1.3 MDS

1.3.1 Observations

[68]:
```
model = MDS(n_components=2, random_state=42)
X_mds = model.fit_transform(X)

unique_authors = sorted(set(authors))

# Plot
plt.figure(figsize=(10, 8))
for i, author in enumerate(unique_authors):
    mask = authors == author
    plt.scatter(X_mds[mask, 0], X_mds[mask, 1], label=author)

plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.title("MDS Embedding of Observations")
plt.legend(title="Author", loc="best")
plt.tight_layout()
plt.savefig("Media/viz/01/01_mds_observation_viz")
plt.show()
```



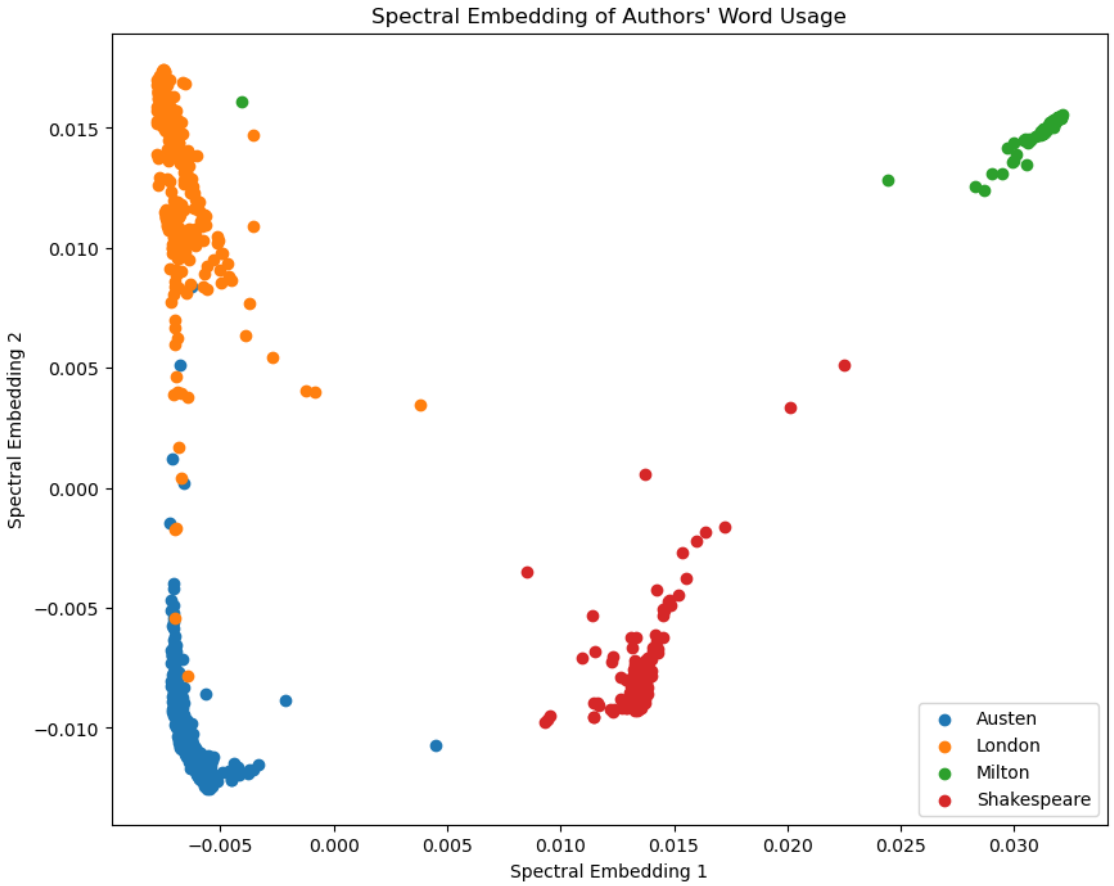
2 Non-Linear Methods

2.1 Spectral Embeddings

2.1.1 Observations

[69]:
```
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10)
X_spec = spectral.fit_transform(X)  # shape: (n_samples, 2)
spectral_df = pd.DataFrame(X_spec, columns=['Dim1', 'Dim2'])

plt.figure(figsize=(10, 8))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
plt.xlabel("Spectral Embedding 1")
plt.ylabel("Spectral Embedding 2")
plt.title("Spectral Embedding of Authors' Word Usage")
plt.legend(loc="lower right")
```

```
plt.savefig('Media/viz/01/01_spectal_obs_viz')
plt.show()
```



2.1.2 Features

Unlike PCA and NMF there are no components we can use to visualize the features in Spectral Embedding. However, we can apply a transpose to our data and then reapply Spectral Embedding.

This procedure will yield results where:

- Two words will be close in the embedding if they tend to appear in the same chapters.
- This gives you a co-occurrence-like structure, driven by chapter usage.
- This is conceptually similar to Latent Semantic Analysis, just via graph-based distance instead of SVD.

[70]:
```
X_transpose = X.T
X_transpose = X_transpose.rename(columns = {i:f'Chapter{i}' for i in range(df.
  ↪shape[0])})
X_transpose
```

[70]:

	Chapter0	Chapter1	Chapter2	Chapter3	Chapter4	Chapter5	Chapter6	\
a	46	35	46	40	29	27	34	
all	12	10	2	7	5	8	8	
also	0	0	0	0	0	0	0	
an	3	7	3	4	6	3	15	
and	66	44	40	64	52	42	44	
…	…	…	…	…	…	…	…	
who	8	3	4	5	2	6	4	
will	4	5	5	3	4	3	9	
with	9	14	15	22	21	18	11	
would	1	8	3	4	10	4	6	
your	0	0	9	3	0	5	4	

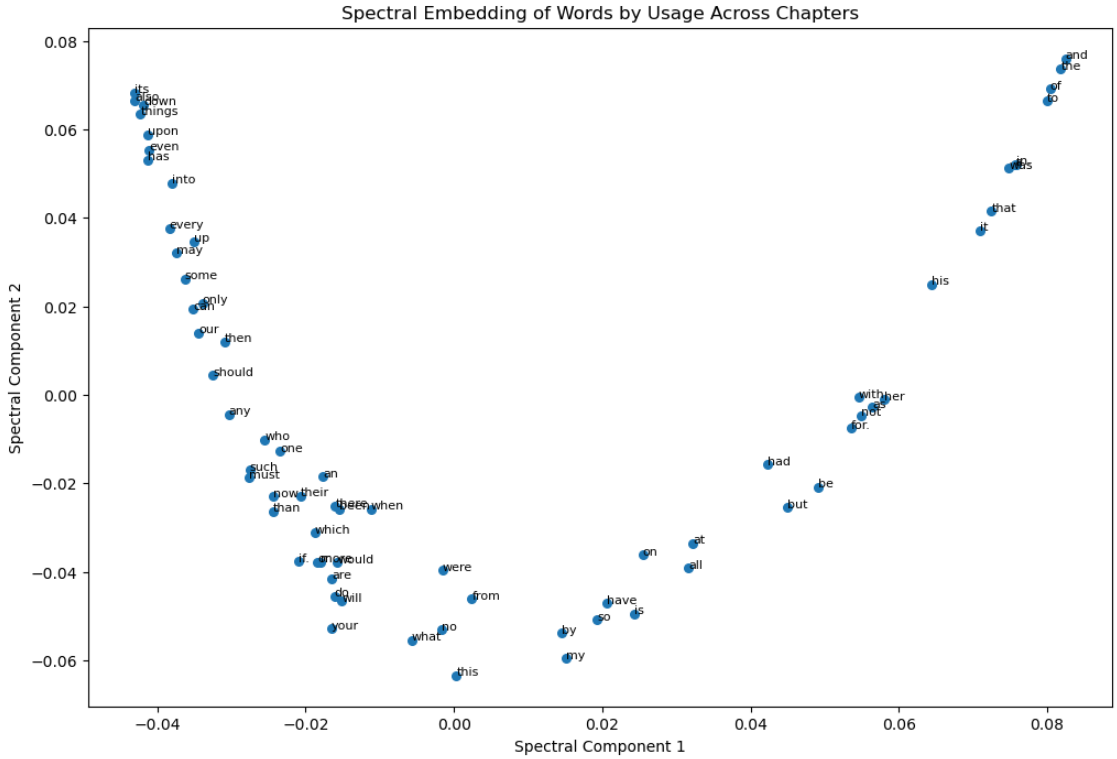
	Chapter7	Chapter8	Chapter9	…	Chapter831	Chapter832	Chapter833	\
a	38	34	54	…	46	48	39	
all	6	12	8	…	4	2	5	
also	1	0	0	…	0	0	0	
an	2	5	6	…	3	9	10	
and	67	50	44	…	43	45	38	
…	…	…	…	…	…	…	…	
who	6	1	3	…	1	0	2	
will	7	2	5	…	7	10	8	
with	15	13	15	…	18	11	26	
would	3	12	6	…	2	6	2	
your	5	5	2	…	3	11	16	

	Chapter834	Chapter835	Chapter836	Chapter837	Chapter838	Chapter839	\
a	22	28	32	16	22	25	
all	13	7	4	5	15	4	
also	0	0	0	0	0	0	
an	5	7	6	5	3	8	
and	47	45	33	49	48	59	
…	…	…	…	…	…	…	
who	4	2	3	0	0	2	
will	9	7	11	11	12	22	
with	12	8	17	20	15	23	
would	6	3	5	2	1	4	
your	7	7	10	7	10	5	

	Chapter840
a	26
all	4
also	0
an	2
and	62
…	…
who	3

```
will           11
with           19
would           0
your            3

[69 rows x 841 columns]
```

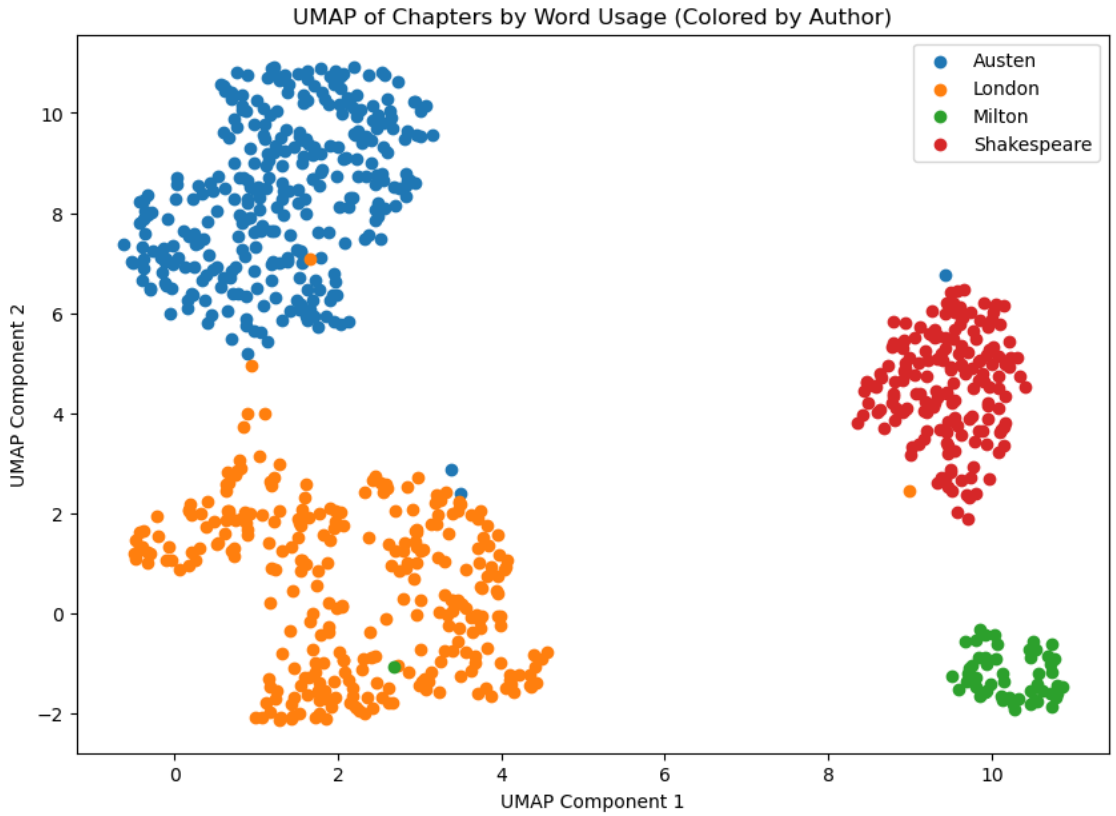
[71]:
```
X_words = X_transpose.to_numpy()
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=0)
X_words_spec = spectral.fit_transform(X_words)

plt.figure(figsize=(12, 8))
plt.scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
plt.title("Spectral Embedding of Words by Usage Across Chapters")
plt.xlabel("Spectral Component 1")
plt.ylabel("Spectral Component 2")
plt.savefig('Media/viz/01/01_spectral_feature_viz')
plt.show()
```



2.2 UMAP

2.2.1 Observations

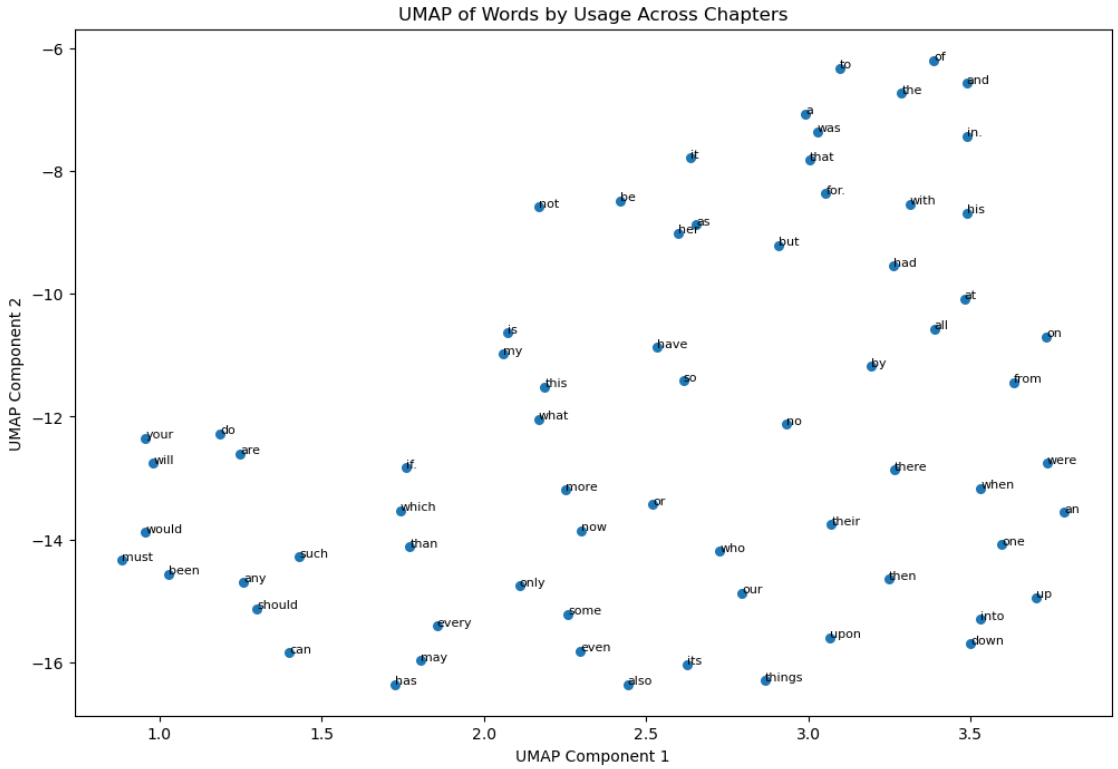
[84]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_umap = umap_model.fit_transform(X.to_numpy())

plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_umap_obs_viz')
plt.show()

plt.figure(figsize=(10, 7))
plt.scatter(X_umap[:,0],X_umap[:,1], color = 'purple')
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP")
plt.savefig('Media/viz/01/01_umap_viz_unlabeled');
plt.close()
```



UMAP does a very good job at creating seperable clusters! This visualization appears to be the best so far when observing observations (chapters)!

2.2.2 Features

[73]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_words_umap = umap_model.fit_transform(X_transpose.to_numpy())

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
plt.title("UMAP of Words by Usage Across Chapters")
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.savefig('Media/viz/01/01_umap_feature_viz')
plt.show()
```

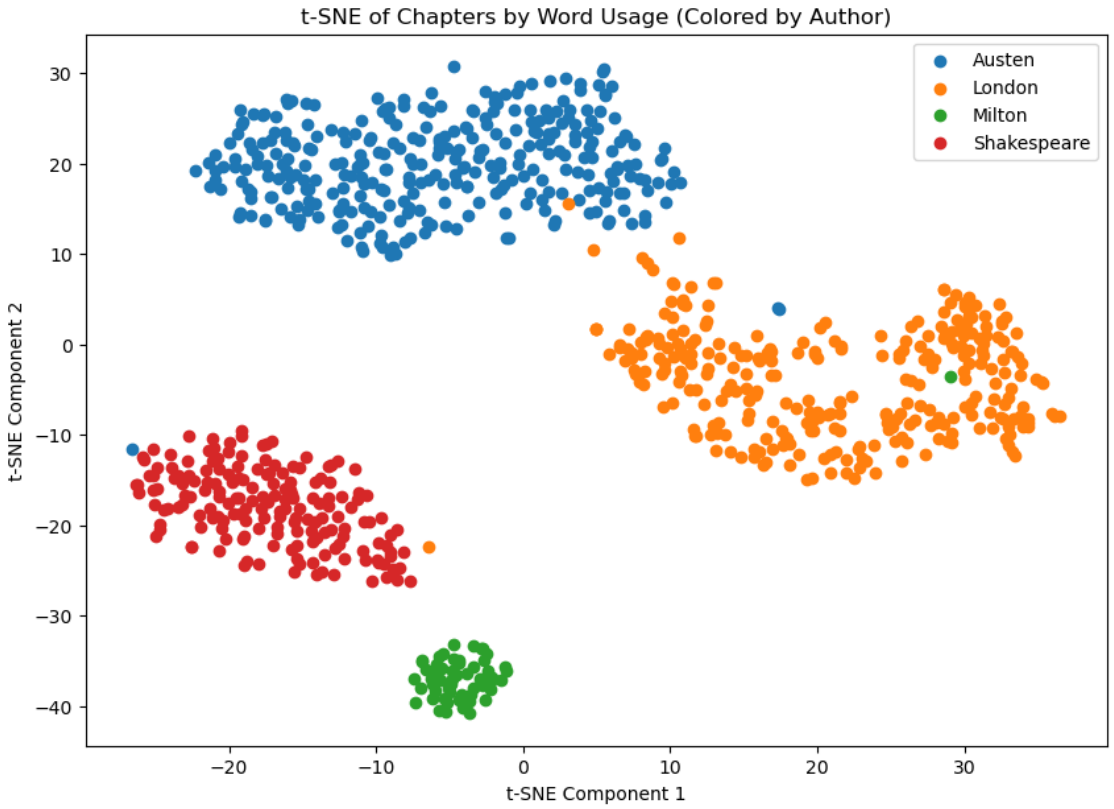


Not very interpretable/helpful.

2.3 tSNE

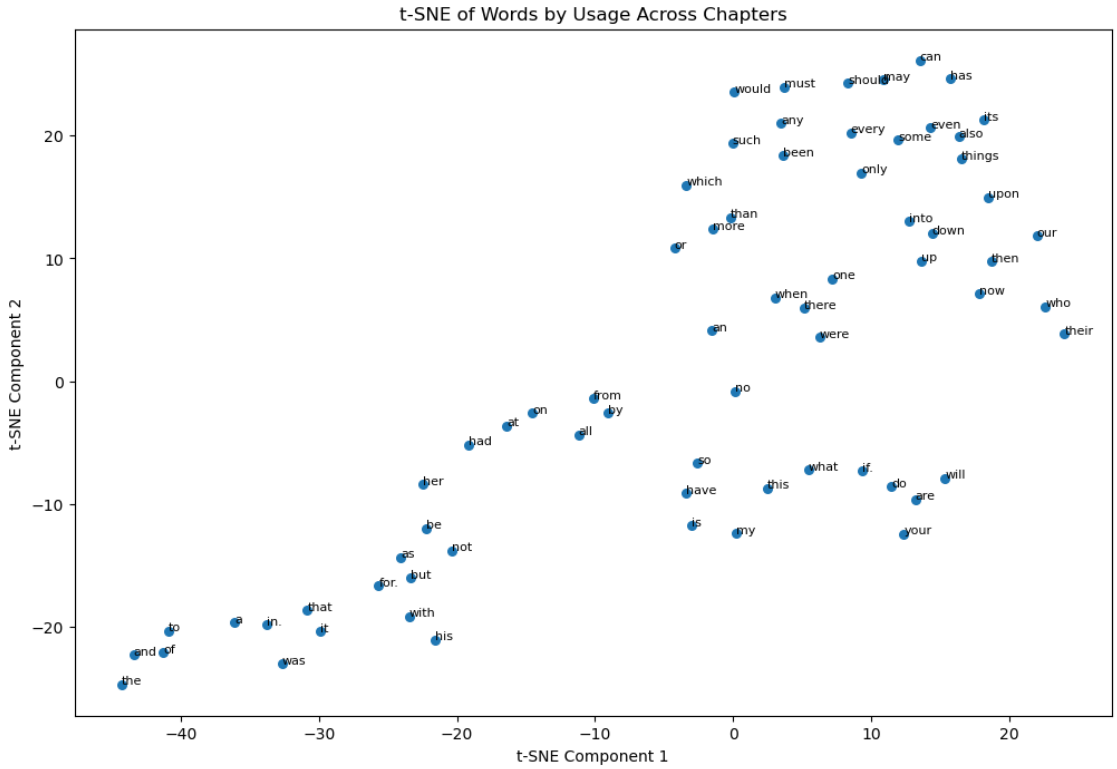
2.3.1 Observations

[74]:
```
tsne = TSNE(n_components=2, perplexity=30, learning_rate='auto') # Fit t-SNE
X_tsne = tsne.fit_transform(X.to_numpy())

# Plot chapters with author labels
plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)

plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.title("t-SNE of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_tsne_obs_viz')
plt.show()
```



2.3.2 Features

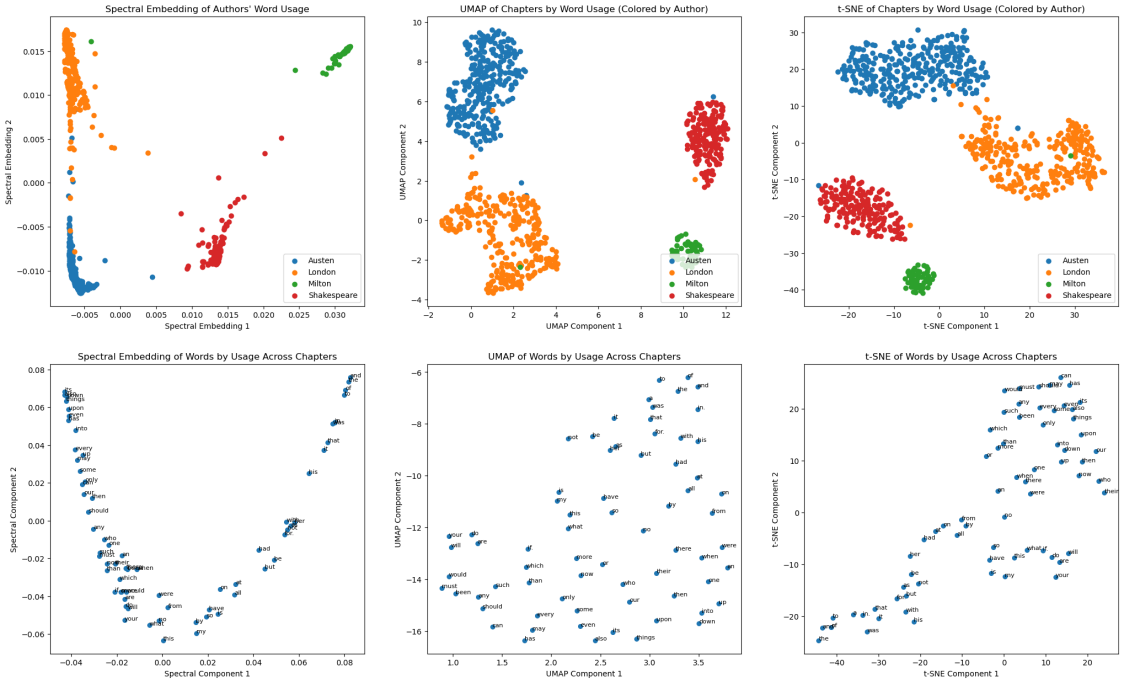
[75]:
```
tsne = TSNE(n_components=2, perplexity=5, learning_rate='auto') # Fit t-SNE
X_words_tsne = tsne.fit_transform(X_words)

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)

plt.title("t-SNE of Words by Usage Across Chapters")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.savefig('Media/viz/01/01_tsne_feature_viz')
plt.show()
```



3 Combined Visualizations

[76]:
```
fig, ax = plt.subplots(2, 3, figsize=(25, 15), sharey=False)  # 1 row, 3 columns

# Spectral Embedding
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,0].scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
ax[0,0].set_xlabel("Spectral Embedding 1")
ax[0,0].set_ylabel("Spectral Embedding 2")
ax[0,0].set_title("Spectral Embedding of Authors' Word Usage")
ax[0,0].legend(loc="lower right")

ax[1,0].scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,0].text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
ax[1,0].set_title("Spectral Embedding of Words by Usage Across Chapters")
ax[1,0].set_xlabel("Spectral Component 1")
ax[1,0].set_ylabel("Spectral Component 2")

# UMAP
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
```

```
    mask = (authors == author)
    ax[0,1].scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
ax[0,1].set_xlabel("UMAP Component 1")
ax[0,1].set_ylabel("UMAP Component 2")
ax[0,1].set_title("UMAP of Chapters by Word Usage (Colored by Author)")
ax[0,1].legend(loc="lower right")

ax[1,1].scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,1].text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
ax[1,1].set_title("UMAP of Words by Usage Across Chapters")
ax[1,1].set_xlabel("UMAP Component 1")
ax[1,1].set_ylabel("UMAP Component 2")

# tSNE
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,2].scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)
ax[0,2].set_xlabel("t-SNE Component 1")
ax[0,2].set_ylabel("t-SNE Component 2")
ax[0,2].set_title("t-SNE of Chapters by Word Usage (Colored by Author)")
ax[0,2].legend(loc="lower right")

ax[1,2].scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,2].text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)
ax[1,2].set_title("t-SNE of Words by Usage Across Chapters")
ax[1,2].set_xlabel("t-SNE Component 1")
ax[1,2].set_ylabel("t-SNE Component 2")

plt.savefig('Media/viz/01/01_nonlinear_viz')
plt.show()
```



UMAP offers the best balance between global and local structure, preserves neighborhood quality, and gives interpretable groupings without as much distortion as t-SNE.

Clusters of words in the UMAP embedding reflect similar usage patterns across chapters. Since word usage is shaped by topic and syntax, these local neighborhoods can be interpreted as reflecting semantic or grammatical similarity. While UMAP does not preserve global distances, local groupings are meaningful.

02_visualizations

April 17, 2025

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE, MDS
from sklearn.decomposition import PCA, NMF, FastICA, TruncatedSVD
from sklearn.metrics import pairwise_distances

## Msc
from adjustText import adjust_text
from itertools import combinations
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df["Author"].values

X_words = X.T.values
feature_names = X.columns
```

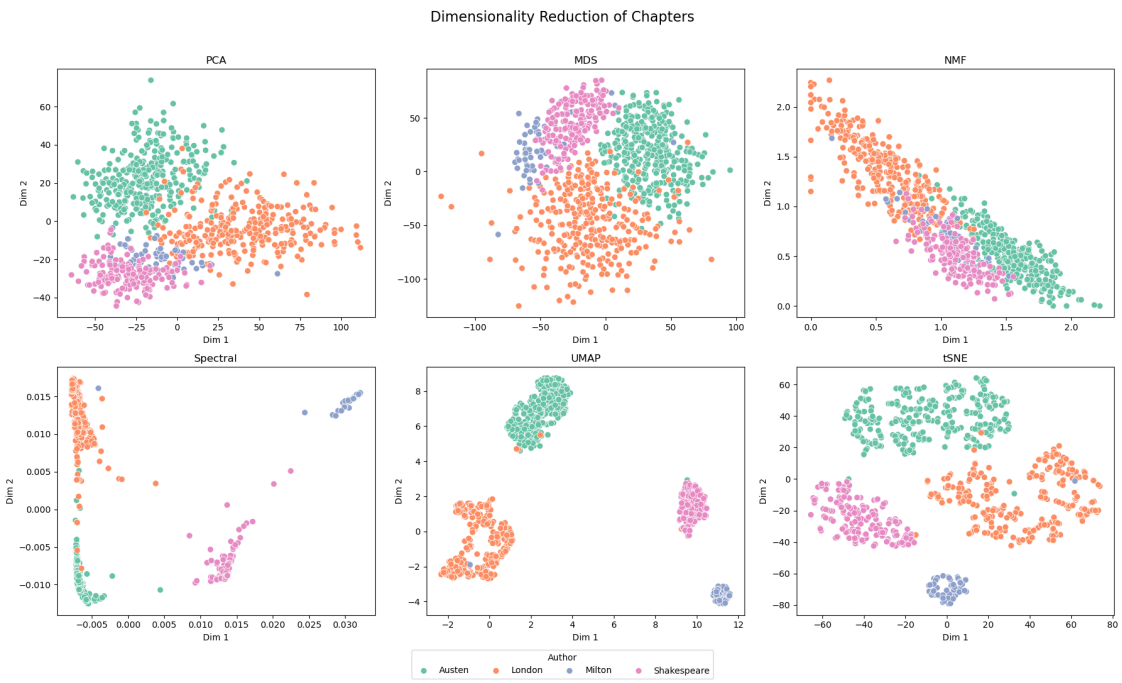
1 Comparison

1.1 Observations

[9]:
```
methods = {
    "PCA": PCA(n_components=2),
    "MDS": MDS(n_components=2, random_state=42),
```

```
    "NMF": NMF(n_components=2, random_state=42, max_iter=1000),
    "Spectral": SpectralEmbedding(n_components=2, affinity="nearest_neighbors",␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

palette = { "Austen": "#66c2a5", "London": "#fc8d62", "Milton": "#8da0cb",␣
 ↪"Shakespeare": "#e78ac3"} # Set consistent colors for each label

embeddings = {name: model.fit_transform(X) for name, model in methods.items()}

fig, axs = plt.subplots(2, int(len(embeddings)/2), figsize=(len(methods)*3, 10))

for ax, (name, embed) in zip(axs.flatten(), embeddings.items()):
    sns.scatterplot(x=embed[:, 0], y=embed[:, 1], hue=authors, palette=palette,␣
 ↪s=50, ax=ax)
    ax.set_title(name)
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")
    ax.legend().remove()

handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, title="Author", loc='lower center', ncol=4,␣
 ↪bbox_to_anchor=(0.5, -0.05))
fig.suptitle("Dimensionality Reduction of Chapters", fontsize=16, y=1.02)

plt.tight_layout()
plt.savefig("Media/viz/02/02_across_methods_obs_viz.png", bbox_inches="tight",␣
 ↪dpi=300)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```

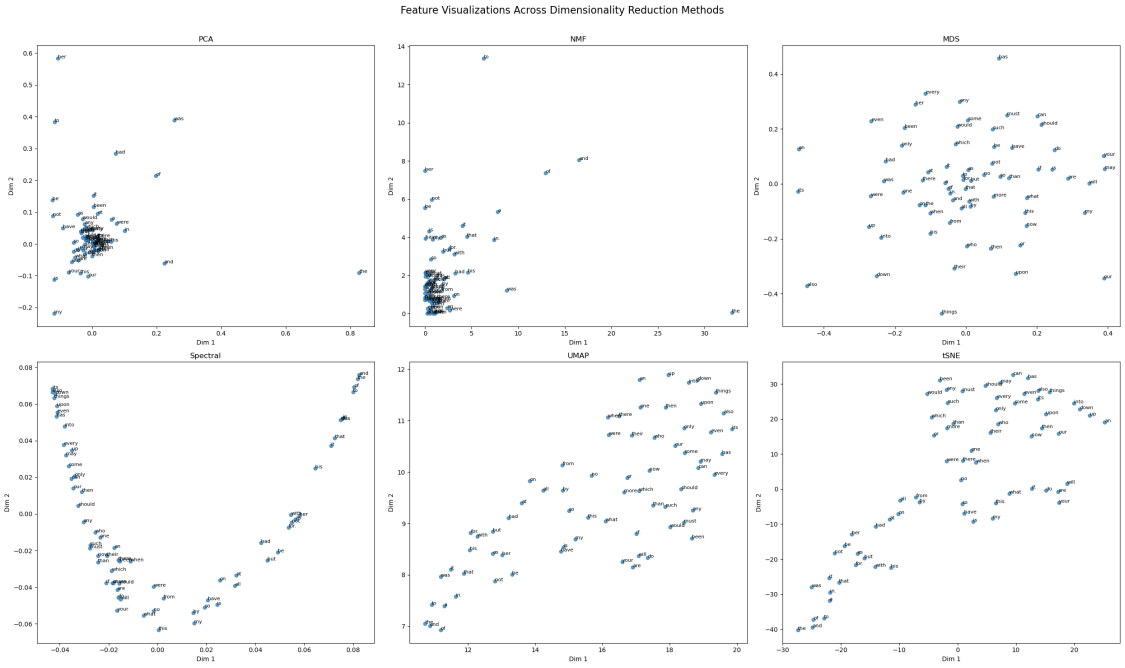


1.2 Features

[5]:
```
methods = {
    "PCA": PCA(n_components=2).fit(X),
    "NMF": NMF(n_components=2, init='random', random_state=42, max_iter=1000).
 ↪fit(X),
    "MDS": MDS(n_components=2, dissimilarity='precomputed', random_state=42),
    "Spectral": SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, n_neighbors=10, min_dist=0.3,␣
 ↪random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

feature_embeddings = {
    "PCA": methods["PCA"].components_.T,
    "NMF": methods["NMF"].components_.T,
    "MDS": methods["MDS"].fit_transform(pairwise_distances(X_words,␣
 ↪metric="cosine")),
    "Spectral": methods["Spectral"].fit_transform(X_words),
    "UMAP": methods["UMAP"].fit_transform(X_words),
    "tSNE": methods["tSNE"].fit_transform(X_words)
}
```

```
fig, axs = plt.subplots(2, 3, figsize=(25, 15))
axs = axs.flatten()

for ax, (name, coords) in zip(axs, feature_embeddings.items()):
    ax.scatter(coords[:, 0], coords[:, 1], s=30, alpha=0.7)
    for i, word in enumerate(feature_names):
        ax.text(coords[i, 0], coords[i, 1], word, fontsize=8)
    ax.set_title(f"{name}")
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")

plt.suptitle("Feature Visualizations Across Dimensionality Reduction Methods",␣
 ↪fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.savefig("Media/viz/02/02_feature_comparison_all_methods")
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```



03_biclustering_viz

April 17, 2025

Install these if necessary.

[1]:
```
# %pip install fastcluster --quiet
```

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
from sklearn.cluster import SpectralBiclustering, SpectralCoclustering
from sklearn.feature_selection import VarianceThreshold

## Msc
from PIL import Image
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values  # n_samples-length array
```

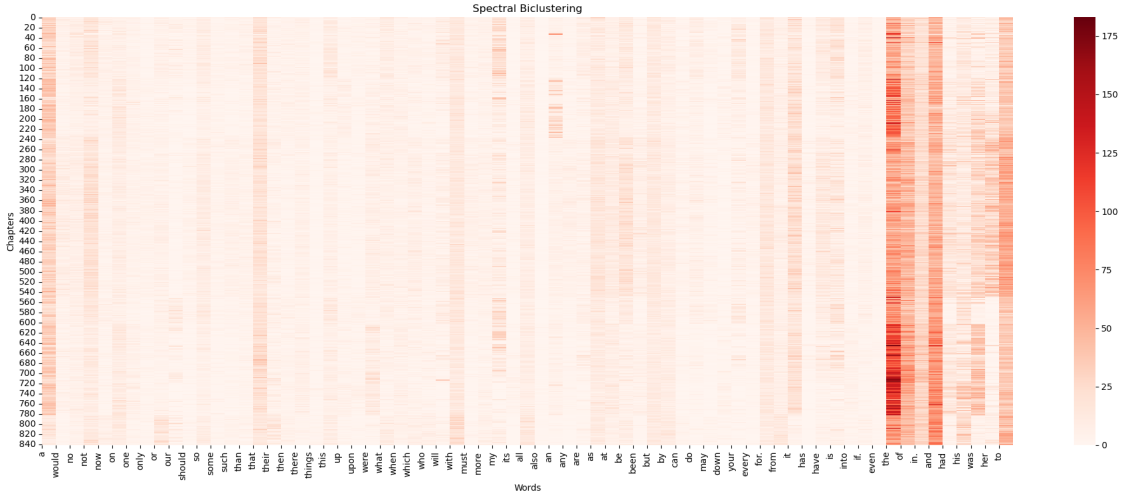
1 Biclustering

1.1 Spectral Biclustering

Let us fit and visualize both the chapters and words using Spectral Biclustering!

[4]:
```
### Fit biclustering model
model = SpectralBiclustering(n_clusters=4, method='log', random_state=0)
model.fit(X)

### Reorder the data
row_order = np.argsort(model.row_labels_)
```

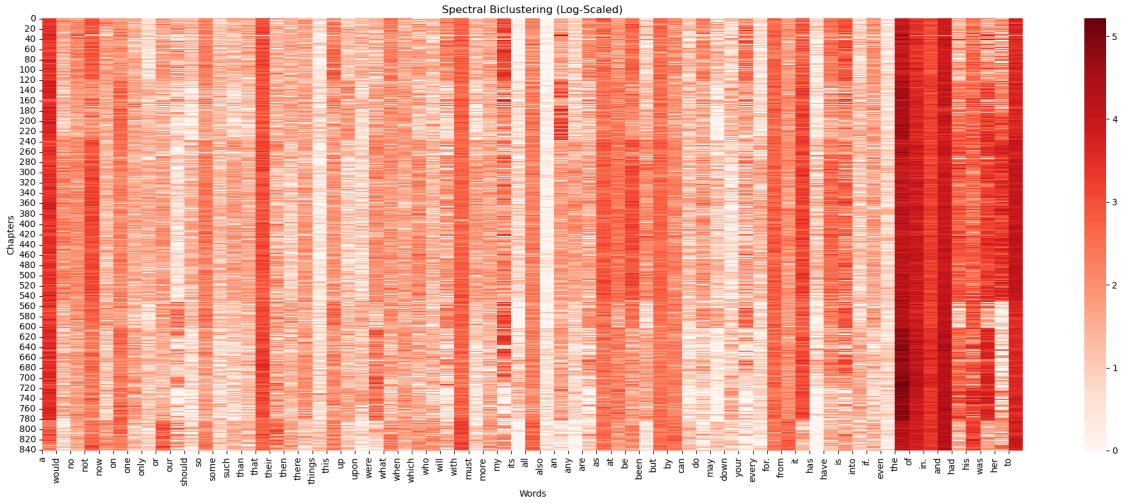
```
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Visualization
plt.figure(figsize=(20, 8))  # wider figurea
sns.heatmap(fit_data, cmap="Reds", cbar=True)
plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/spectral_biclustering_viz')
plt.show()
```



Since words like “the”, “and”, “to”, … are used frequently across chapters on a global level there is not much insights to be deduced here. We want to look for at words with specific chapters having bands. This will help us to differentiate a word that is frequently used across a cluster of chapters indicating stylistic writing by the author! This will help with classifying these chapters to their respective authors!! There is a small problem that the common words are very dark so it is harder to see a contrast in scale so we will convert the scale to log scale to see a larger contrast among the less frequently used words.

1.1.1 Log-Scale (Improve Visualization)

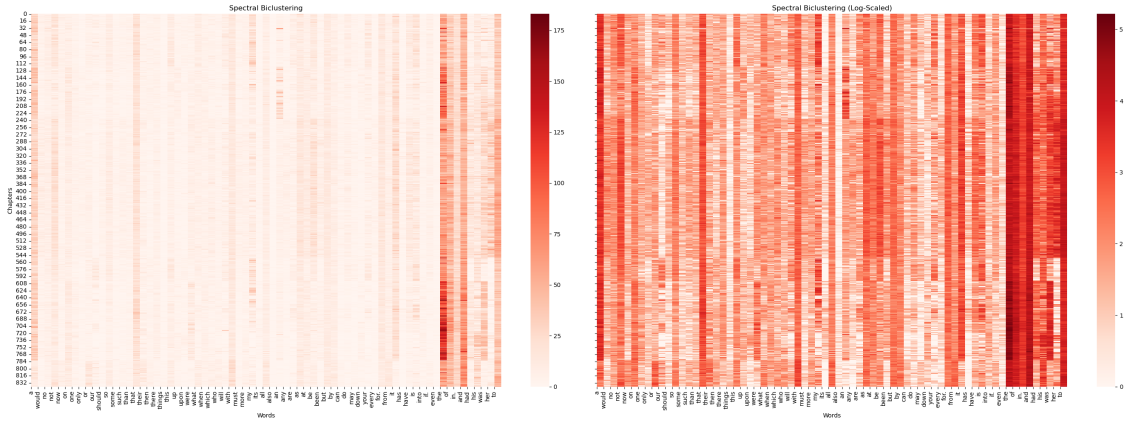
[5]:
```
### Reorder the data
row_order = np.argsort(model.row_labels_)
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Apply log scale to compress high-frequency words like "the"
log_data = np.log1p(fit_data)  # log(1 + x) to avoid log(0)

### Plot the heatmap
plt.figure(figsize=(20, 8))
sns.heatmap(log_data, cmap="Reds", cbar=True)

plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering (Log-Scaled)")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/log_scale_spectral_biclustering_viz')
plt.show()
```



1.1.2 Final Visualization

Plot side by side for visualization/comparison purposes.

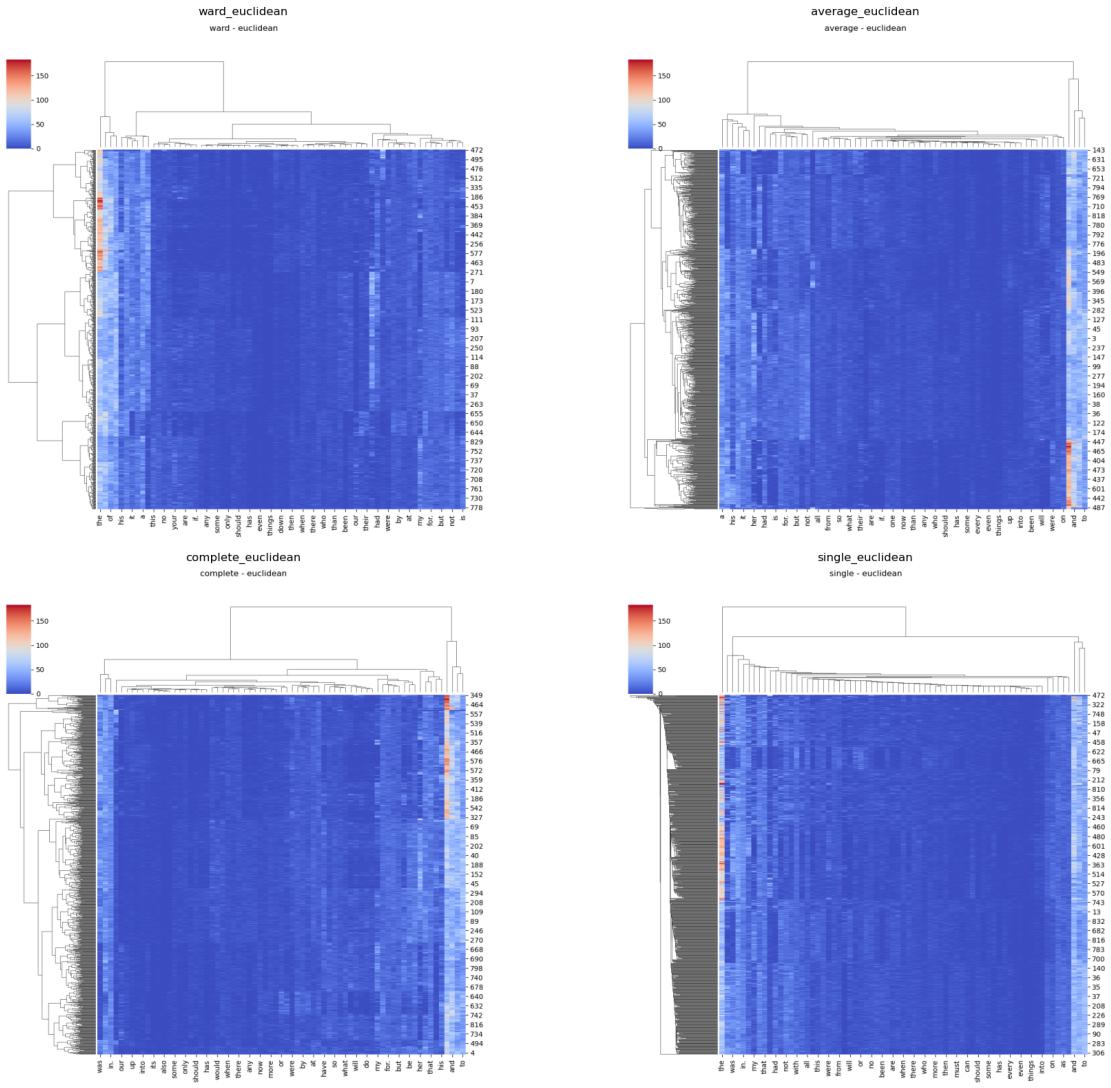
[6]:
```
# Plot side-by-side heatmaps
fig, axes = plt.subplots(1, 2, figsize=(28, 10), sharey=True)

# Original
sns.heatmap(fit_data, cmap="Reds", ax=axes[0], cbar=True)
axes[0].set_xticks(np.arange(len(word_labels)))
axes[0].set_xticklabels(word_labels, rotation=90)
axes[0].set_title("Spectral Biclustering")
axes[0].set_xlabel("Words")
axes[0].set_ylabel("Chapters")

# Log-scaled
sns.heatmap(log_data, cmap="Reds", ax=axes[1], cbar=True)
axes[1].set_xticks(np.arange(len(word_labels)))
axes[1].set_xticklabels(word_labels, rotation=90)
axes[1].set_title("Spectral Biclustering (Log-Scaled)")
axes[1].set_xlabel("Words")
axes[1].set_ylabel("")

plt.tight_layout()
plt.savefig('Media/viz/03/03_spectral_biclustering_heatmaps')
plt.show()
```



1.2 Hierarchial Biclustering

[7]:
```
# Define top 4 method-metric combinations (all Euclidean)
combinations = [
    ('ward', 'euclidean'),
    ('average', 'euclidean'),
    ('complete', 'euclidean'),
    ('single', 'euclidean'),
]
```

[8]:
```
img_paths = []
for method, metric in combinations:
    g = sns.clustermap(X, method=method, metric=metric, cmap='coolwarm')
    g.fig.suptitle(f'{method} - {metric}', y=1.05)
    img_path = f'Media/viz/03/single_plots/clustermap_{method}_{metric}.png'
    g.savefig(img_path, bbox_inches='tight')
    plt.close(g.fig)
    img_paths.append(img_path)

fig, axes = plt.subplots(2, 2, figsize=(20, 16))
for ax, path in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(path.split('/')[-1].replace('clustermap_', '').replace('.png',␣
 ↪''))
    ax.axis('off')

plt.tight_layout()
plt.savefig('Media/viz/03/03_hierarchial_biclustering_heatmaps')
plt.show()
```



This clustermap visualizes word frequency across chapters using hierarchical clustering on both rows (chapters) and columns (words). Chapters with similar word usage patterns are grouped together, as are words that tend to co-occur across the same sets of chapters. The dendrograms reveal hidden structure in the data, highlighting natural groupings based on stylistic similarity without any labels. This helps identify clusters of similar chapters and potentially stylistically meaningful groups of words.

However, we can see that the interpretation from these heatmaps is tricky so lets try and reduce the features with low variance patterns. So we will do some initial feature filtering, then perform the same heatmap visualizations!

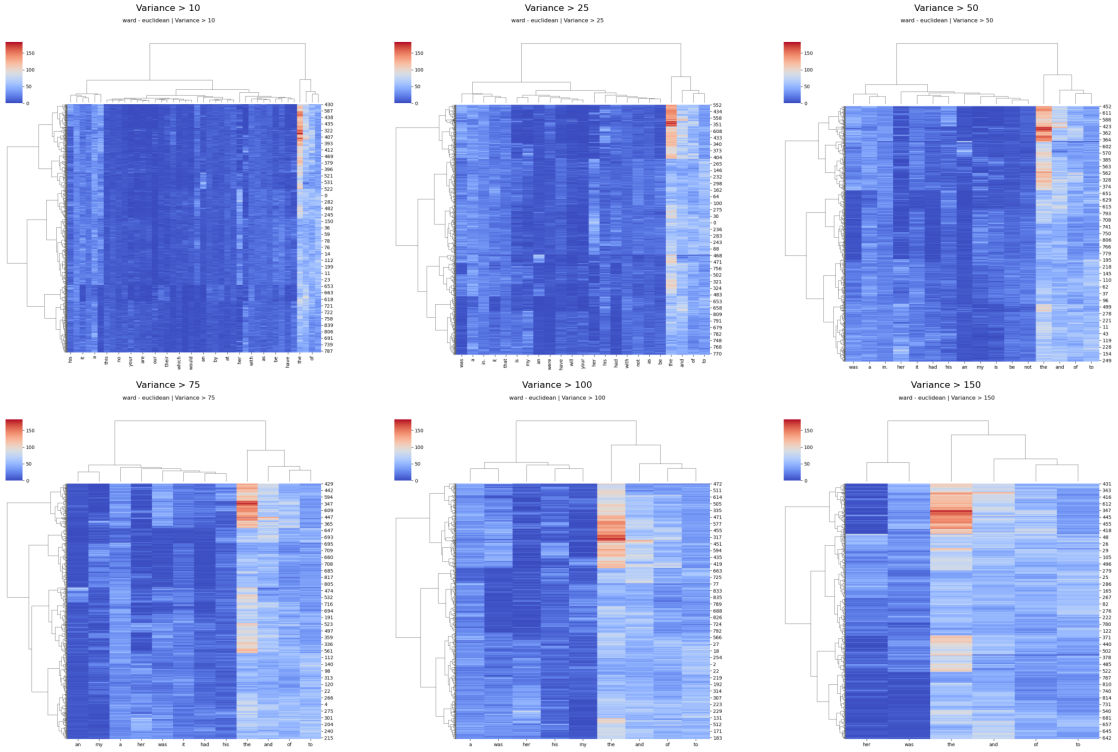
[9]:
```
X_filtered_dict = {'var_threshold':[], 'X_filtered_df':[]}
thresholds_list = [10, 25, 50, 75, 100, 150]
combinations = [('ward', 'euclidean')]
```

```
for alpha in thresholds_list: # Loop over variance filter threshold as alpha␣
 ↪increase, more features are dropped!
    selector = VarianceThreshold(threshold=alpha)
    X_reduced = selector.fit_transform(X)
    selected_columns = X.columns[selector.get_support()] # Get the column names␣
 ↪that passed this variance threshold
    X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)
    X_filtered_dict['var_threshold'].append(alpha)
    X_filtered_dict['X_filtered_df'].append(X_filtered)
```

Plotting a grid across variance thresholds as follows.

[10]:
```
img_paths = []
for alpha in thresholds_list:
    selector = VarianceThreshold(threshold=alpha)
    try:
        X_reduced = selector.fit_transform(X)
        selected_columns = X.columns[selector.get_support()]
        if len(selected_columns) == 0:
            continue
        X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)

        for method, metric in combinations:
            g = sns.clustermap(X_filtered, method=method, metric=metric,␣
 ↪cmap='coolwarm')
            g.fig.suptitle(f'{method} - {metric} | Variance > {alpha}', y=1.05)
            img_path = f'Media/viz/03/single_plots/
 ↪reduced_clustermap_{method}_{metric}_var{alpha}.png'
            g.savefig(img_path, bbox_inches='tight')
            plt.close(g.fig)
            img_paths.append((img_path, alpha))
    except ValueError:
        continue  # Skip thresholds that drop all features

n_cols = 3
n_rows = -(-len(img_paths) // n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(22, 7 * n_rows))

for ax, (path, alpha) in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(f'Variance > {alpha}', fontsize=12)
    ax.axis('off')

plt.tight_layout()
```

```
plt.savefig('Media/viz/03/03_hierarchical_biclustering_variance_grid.png')
plt.show()
```



This feature selection helps to make the patterns between the stop words (features) and the chapters a lot more distinguishable by dropping the sparse features that experience little variance across chapters. From this we can see bands that may indicate a shift in semantics across chapters!

04_clustering_comparison

April 17, 2025

Load necessary libraries.

[1]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap

### Msc
import warnings

### OOP
from ml_utils import ClusterEvaluator
```

Load dataset.

[2]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
 ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
X = X.to_numpy() # change pd.DataFrame to np.ndarray
authors = df['Author'].values  # n_samples-length array
```

We can also assign a hyperparameter $K = 4$. This represents the number of clusters (which we already know; there are 4 authors). Later on we will be hyperparameter tuning for some K based on stability!

[3]:
```
K_ = 4
```

Let us also define a dictionary to store our results.

[4]:
```
accuracy_dict = {}
```

1 Clustering Methods

1.1 Kmeans++

[5]:
```
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300) #␣
 ↪K-means++ initialization
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'London'}
The accuracy of Kmeans++ = 0.9096313912009513
```

1.2 Gaussian Mixture Models

[6]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X)
y_gmm = gmm.predict(X)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9512485136741974
```

1.3 Spectral Clustering

Spectral clustering already had a dimensional reduction by design, i.e., spectral clustering is essentially spectral embedding followed by kmeans! Therefore, we expect this 'raw' method to perform best compared to the other methods without any form of dimensionality reduction!

[7]:
```
spectral = SpectralClustering(n_clusters=K_, affinity='nearest_neighbors')
y_spectral = spectral.fit_predict(X)
```

```
accuracy, mapping = ClusterEvaluator(y_spectral,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['spectral clustering'] = accuracy
```

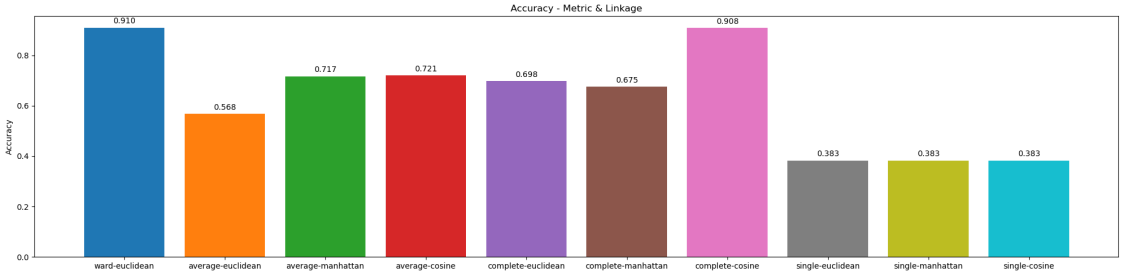
```
The mapping based on mode = {0: 'Shakespeare', 1: 'Austen', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9881093935790726
```

1.4 Hierarchial Clustering

[8]:
```
# Establish all possible combinations for linkage and metric!
methods = []
linkage_methods = ['ward', 'average', 'complete', 'single']
metrics = ['euclidean', 'manhattan', 'cosine']

for linkage in linkage_methods:
    for metric in metrics:
        if linkage == 'ward' and metric != 'euclidean':
            continue  # ward only supports euclidean
        methods.append((linkage, metric))

# Loop through all combinations and determine best accuracy!
hierarchical_accuracy_dict = {}
vals = df['Author'].values
for linkage, metric in methods:
    hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
    y_hierarchical = hierarchical.fit_predict(X)  # Fit and predict in one step
    accuracy, mapping = ClusterEvaluator(y_hierarchical,vals).accuracy # Use␣
 ↪OOP code
    # print(f'The mapping based on mode = {mapping}')
    # print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
    hierarchical_accuracy_dict[f'{linkage}-{metric}'] = accuracy
```

[9]:
```
keys = list(hierarchical_accuracy_dict.keys())
vals = list(hierarchical_accuracy_dict.values())
colors = plt.colormaps.get_cmap('tab10').colors

fig, ax = plt.subplots(figsize=(20, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
```

```
plt.savefig('Media/viz/04/04_hier_accuracy_across_metric_linkage')
plt.show()
```

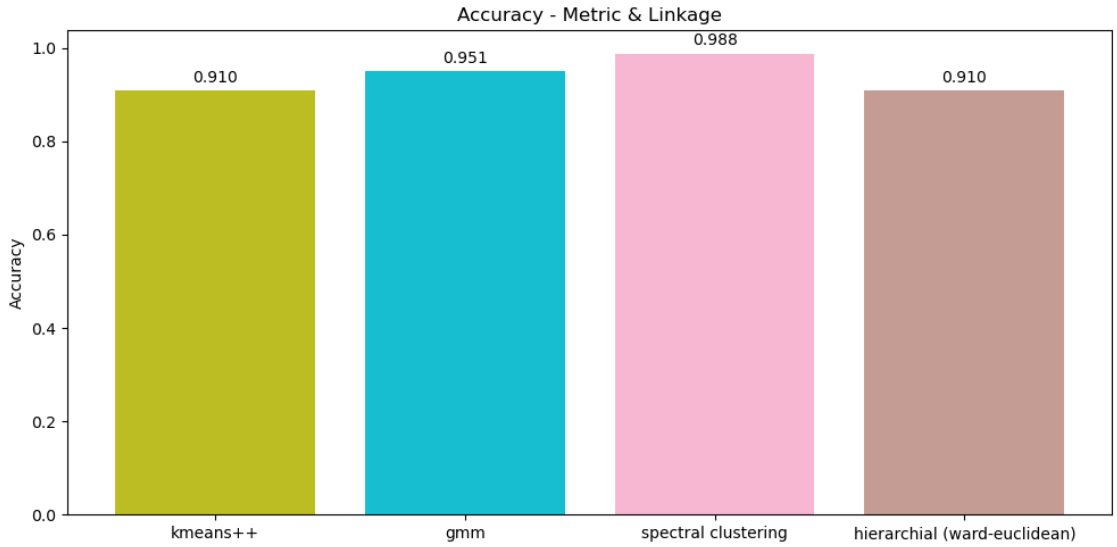


Lets store the highest accuracy params to compare across other methods - ward + euclidean.

[10]:
```
accuracy_dict['hierarchial (ward-euclidean)'] =␣
↪hierarchical_accuracy_dict['ward-euclidean']
```

1.5 Comparison

[11]:
```
keys = list(accuracy_dict.keys())
vals = list(accuracy_dict.values())

colors = ['#bcbd22', '#17becf', '#f7b6d2', '#c49c94']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods')
plt.show()
```



Clearly, spectral clustering performs the best because it has a build in dimensional reduction! Therefore, now let us apply UMAP to all these methods and recompare.

2 Dimensional Reduction + Clustering Methods

Use UMAP as dimensionality reduction for all methods (except spectral clustering as this already uses spectral embedding so we will not repeat this).

[12]:
```
warnings.filterwarnings("ignore", category=UserWarning, module="umap") #␣
↪Suppress the specific UMAP warning on parralelism
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3)
X_umap = umap_model.fit_transform(X)
```

Store results in the following dictionary to compare across methods.

[13]:
```
umap_accuracy_dict = {}
umap_accuracy_dict['spectral clustering'] = accuracy_dict['spectral clustering']
```

2.1 Kmeans++

[14]:
```
# Run K-means++ on UMAP-reduced data
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300)
kmeans.fit(X_umap)
y_kmeans = kmeans.predict(X_umap)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
```

```
umap_accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.2 Gaussian Mixture Models

[15]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X_umap)
y_gmm = gmm.predict(X_umap)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
umap_accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Shakespeare', 1: 'London', 2: 'Austen', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.3 Hierarchial Clustering

[16]:
```
linkage = 'ward'
metric = 'euclidean'
hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
y_hierarchical = hierarchical.fit_predict(X_umap)  # Fit and predict in one step
accuracy, mapping = ClusterEvaluator(y_hierarchical,df['Author'].values).
 ↪accuracy # Use OOP code
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
umap_accuracy_dict[f'hierarchial ({linkage}-{metric})'] = accuracy
```

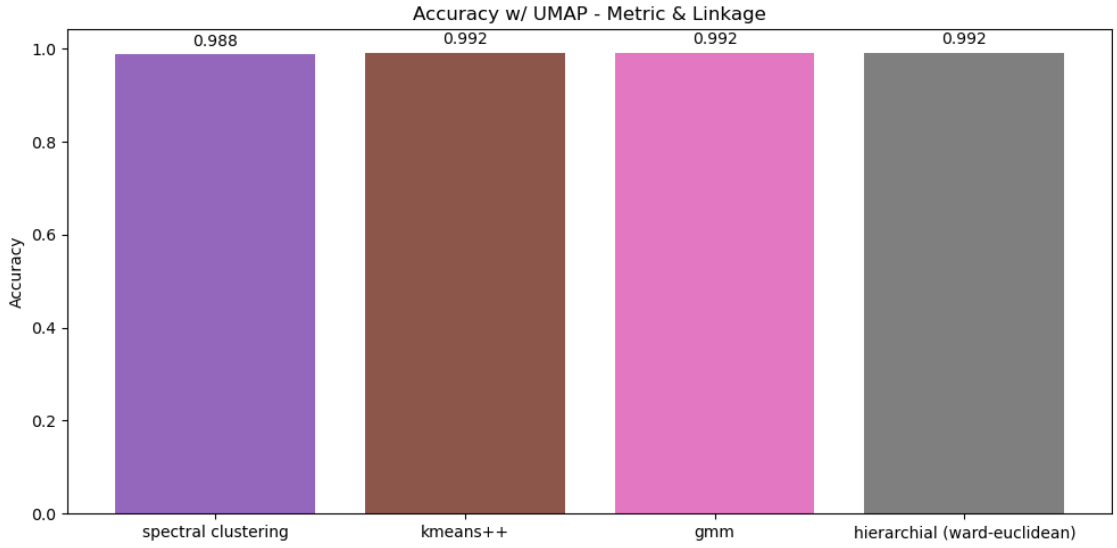
```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy hierarchial clustering with ward and euclidean = 0.9916765755053508
```

2.4 Comparison

As we can see, all these methods (with the exception of spectral clustering as this did not use UMAP) converge to the same accuracy. This is because after applying UMAP we get very clearly defined clusters so there is not much/any room for these methods to diverge in their clustering assignments! Therefore they will yield the same accuracy; this makes sense!

[17]:
```
keys = list(umap_accuracy_dict.keys())
vals = list(umap_accuracy_dict.values())
```

```
colors = ['#9467bd', '#8c564b', '#e377c2', '#7f7f7f']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy w/ UMAP - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods_with_umap')
plt.show()
```



05_generalizibility_silhouette_score

April 17, 2025

Load necessary libraries.

[9]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap
from sklearn.metrics import silhouette_samples

### Msc
import warnings
from PIL import Image

### OOP
from ml_utils import SilhouetteEvaluator, ClusterEvaluator, make_gmm,␣
 ↪make_hierarchical, make_spectral
```

Load dataset.

[10]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array

X_clean = df.drop(columns=['Author'])
X = X_clean.to_numpy()
```

```
# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = umap_model.fit_transform(X)
```

1 Validation

The method with the highest silhouette score generalizes best!

1.1 Kmeans++

Kmeans++ with and without UMAP:

1.1.1 Generalizibility

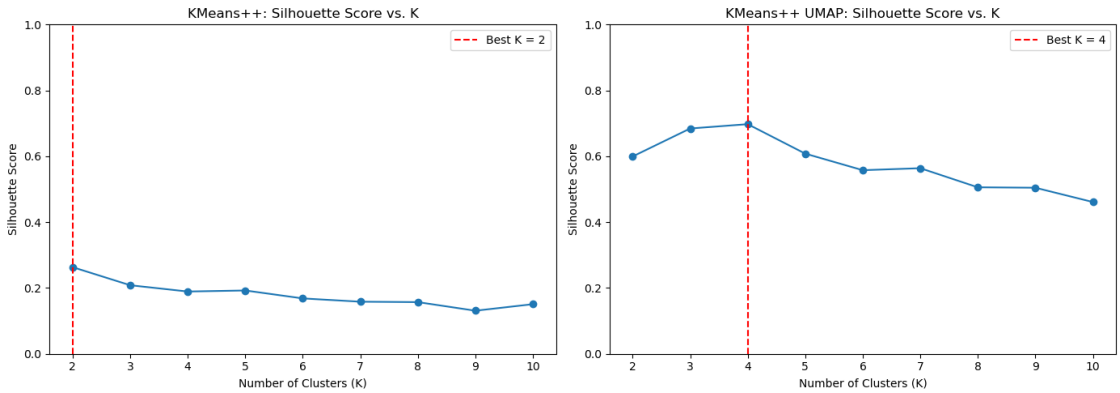
[11]:
```
# Kmeans++
print('\033[1m' + 'Kmeans++ without dimension reduction:' + '\033[0m')
evaluator_kmeans = SilhouetteEvaluator(X, KMeans, k_range=range(2, 11),␣
 ↪init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Kmeans++ with UMAP
print('\033[1m' + 'Kmeans++ with dimension reduction (UMAP):' + '\033[0m')
evaluator_kmeans_umap = SilhouetteEvaluator(X_umap, KMeans, k_range=range(2,␣
 ↪11), init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

y_min = 0
y_max = 1

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_kmeans.plot("KMeans++", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_kmeans_silhouette_score')
plt.show()
```

Kmeans++ without dimension reduction:

```
The scores across K = {2: 0.2632255060236338, 3: 0.20836751818875782, 4:
0.18927180616599434, 5: 0.1924461042273646, 6: 0.16832953504579296, 7:
0.15834519448558668, 8: 0.15708860765446275, 9: 0.13097719253825474, 10:
0.15080653215185813}
Best K: 2
```

Kmeans++ with dimension reduction (UMAP):

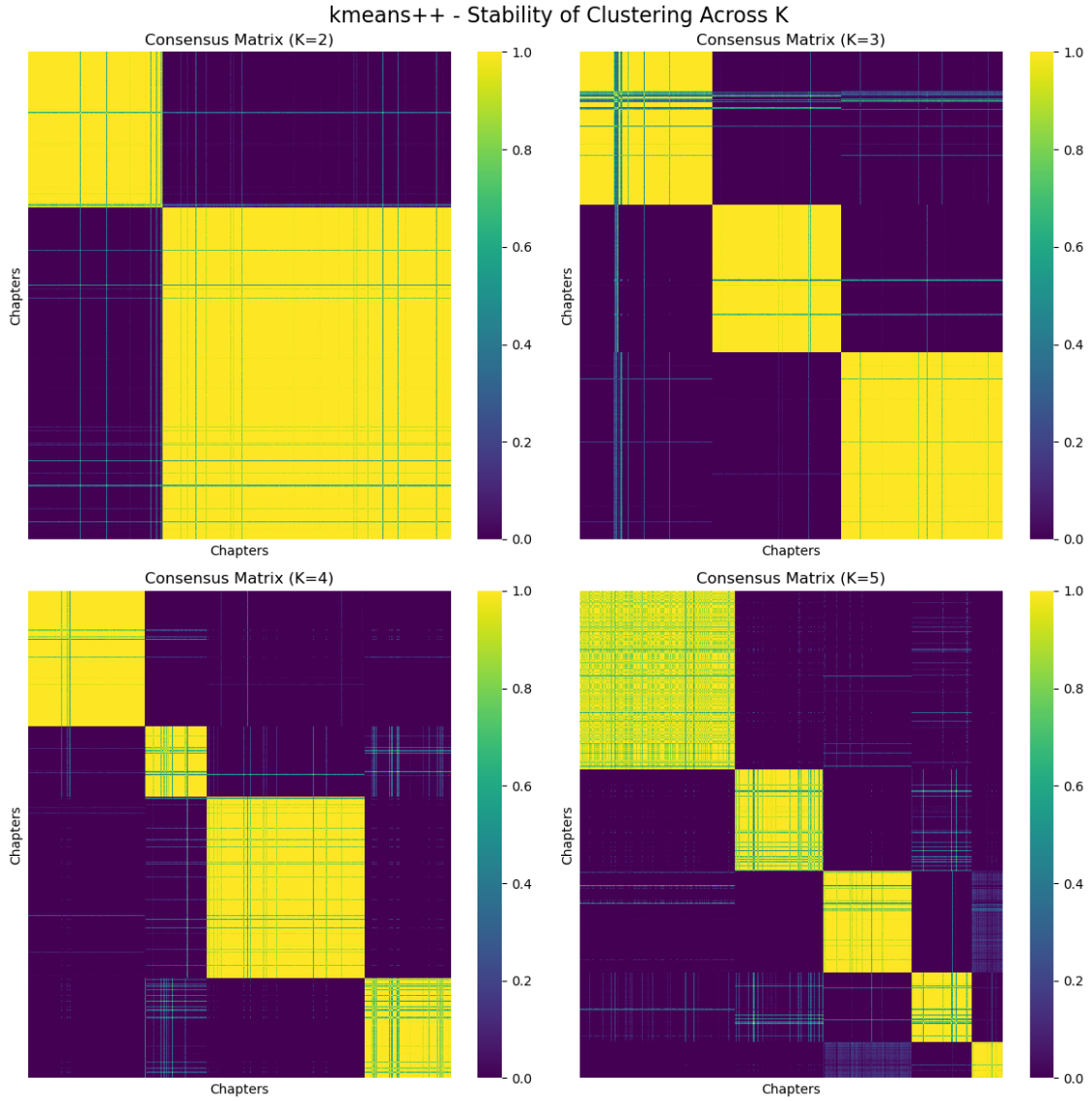
```
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60805935, 6: 0.55773854, 7: 0.5637968, 8: 0.505771, 9: 0.5044795, 10:
0.4610399}
Best K: 4
```



1.1.2 Stability

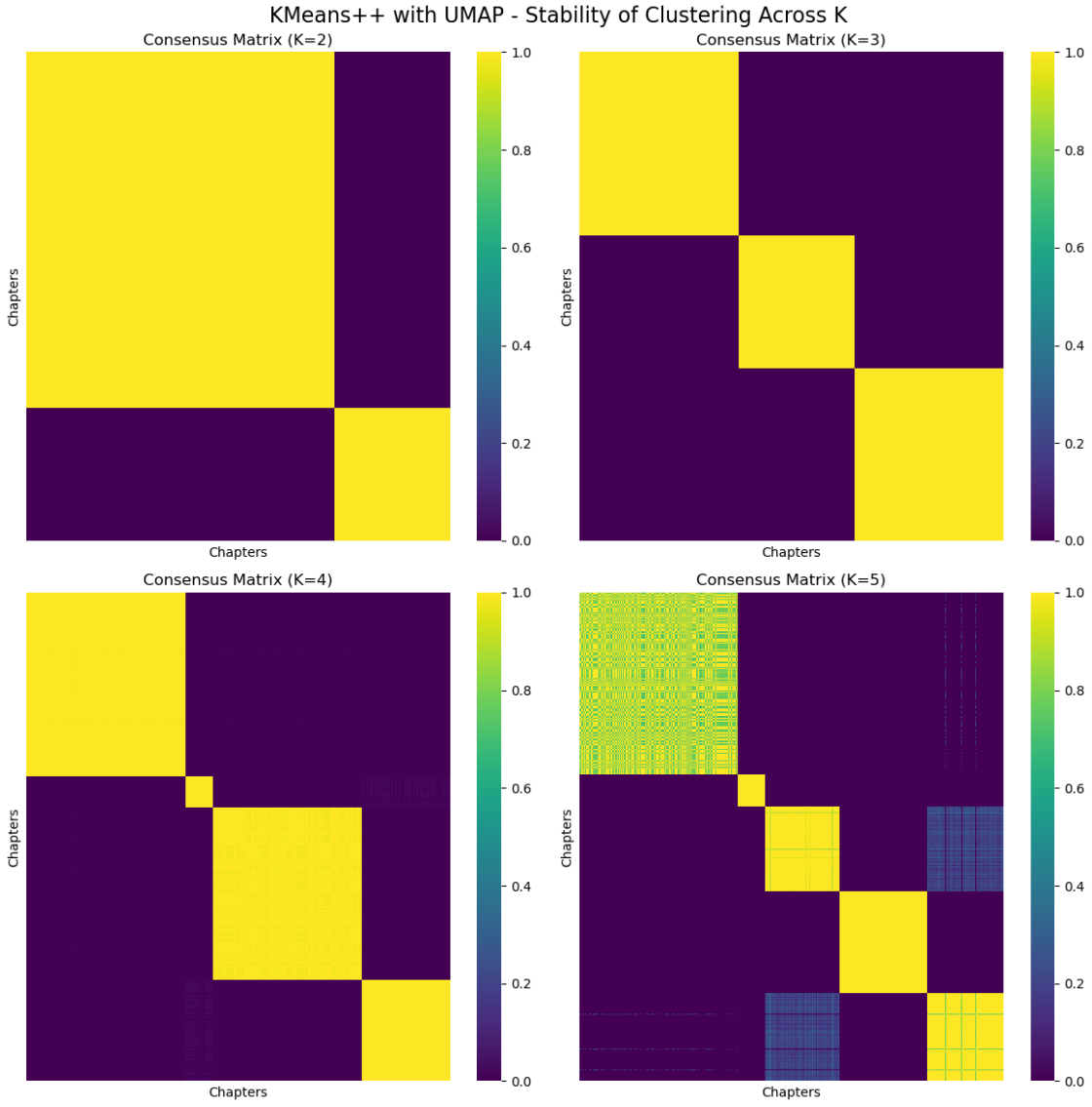
[12]:
```
method = 'kmeans++'

rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_train = X[train_idx]  # Using saved NumPy array
```

```
        km = KMeans(n_clusters=K, random_state=n, n_init=10)
        cluster_labels = km.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, random_state=42, n_init=10).
↪fit_predict(X)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans')
plt.show()
```



[13]:
```
method = 'KMeans++ with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
```

```
    sampled_matrix = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        km = KMeans(n_clusters=K, n_init=10, init='k-means++', random_state=i)
        labels = km.fit_predict(X_sample)

        sampled_matrix[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus_matrix[np.ix_(idx_sample, idx_sample)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=sampled_matrix != 0)
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, n_init=10, init='k-means++',␣
↪random_state=42).fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus_matrix[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans_umap')
plt.show()
```



1.2 Gaussian Mixture Models

1.2.1 Generalizibility

[14]:
```
# GMM
print('\033[1m' + 'GMM without dimension reduction:' + '\033[0m')
evaluator_gmm = SilhouetteEvaluator(X, make_gmm(), k_range=range(2, 11))
scores, best_k = evaluator_gmm.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# GMM with UMAP
print('\033[1m' + 'GMM with dimension reduction (UMAP):' + '\033[0m')
```

```
evaluator_gmm_umap = SilhouetteEvaluator(X_umap, make_gmm(), k_range=range(2,␣
↪11))
scores, best_k = evaluator_gmm_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_gmm.plot("GMM", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()

plt.savefig('Media/viz/05/05_gmm_silhouette_score')
plt.show()
```

GMM without dimension reduction:

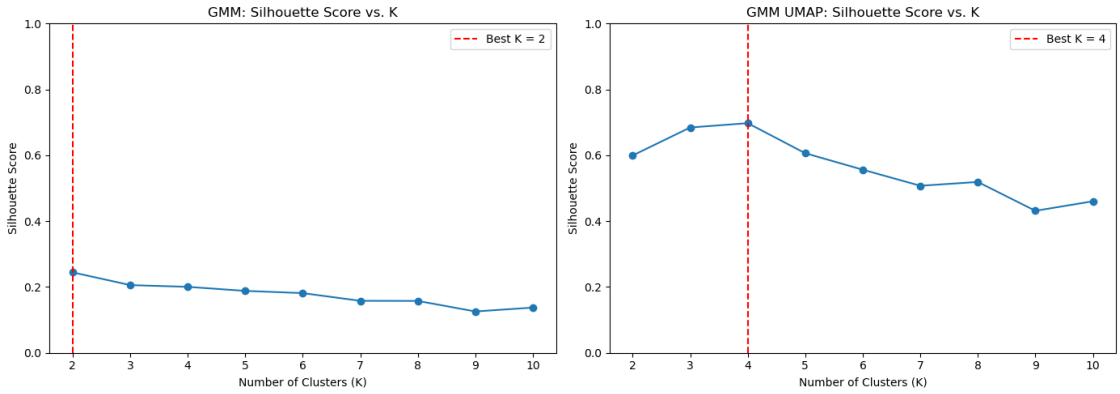
The scores across K = {2: 0.24434605026692022, 3: 0.20578434655866124, 4: 0.2004213780628659, 5: 0.18814970124375807, 6: 0.18136222277608166, 7: 0.15798484734742102, 8: 0.15765164283870803, 9: 0.1256388670716472, 10: 0.1374306014397366}

Best K: 2

GMM with dimension reduction (UMAP):

The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5: 0.60627675, 6: 0.55619246, 7: 0.5074211, 8: 0.51900756, 9: 0.43136632, 10: 0.4602903}

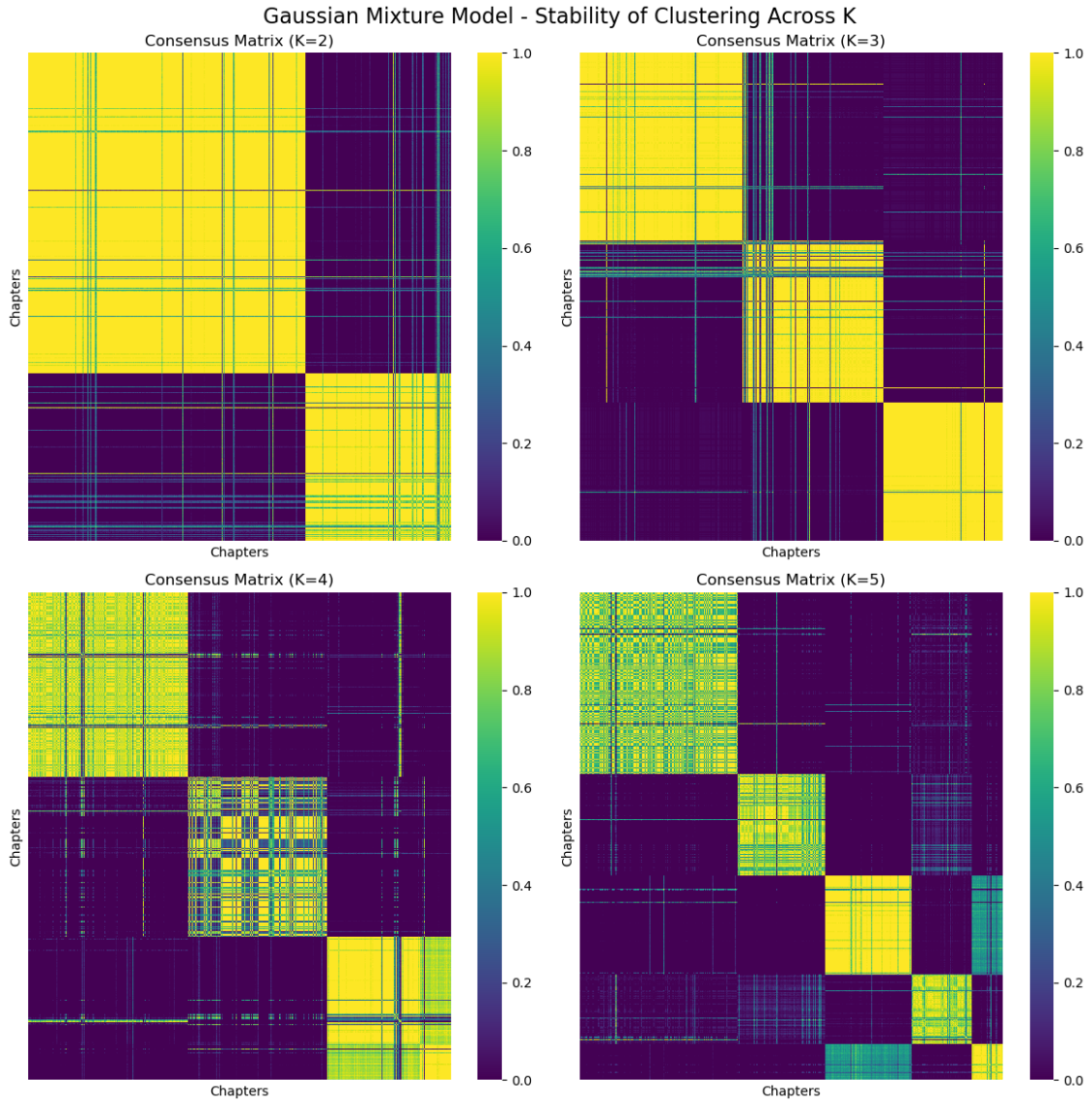
Best K: 4



1.2.2 Stability

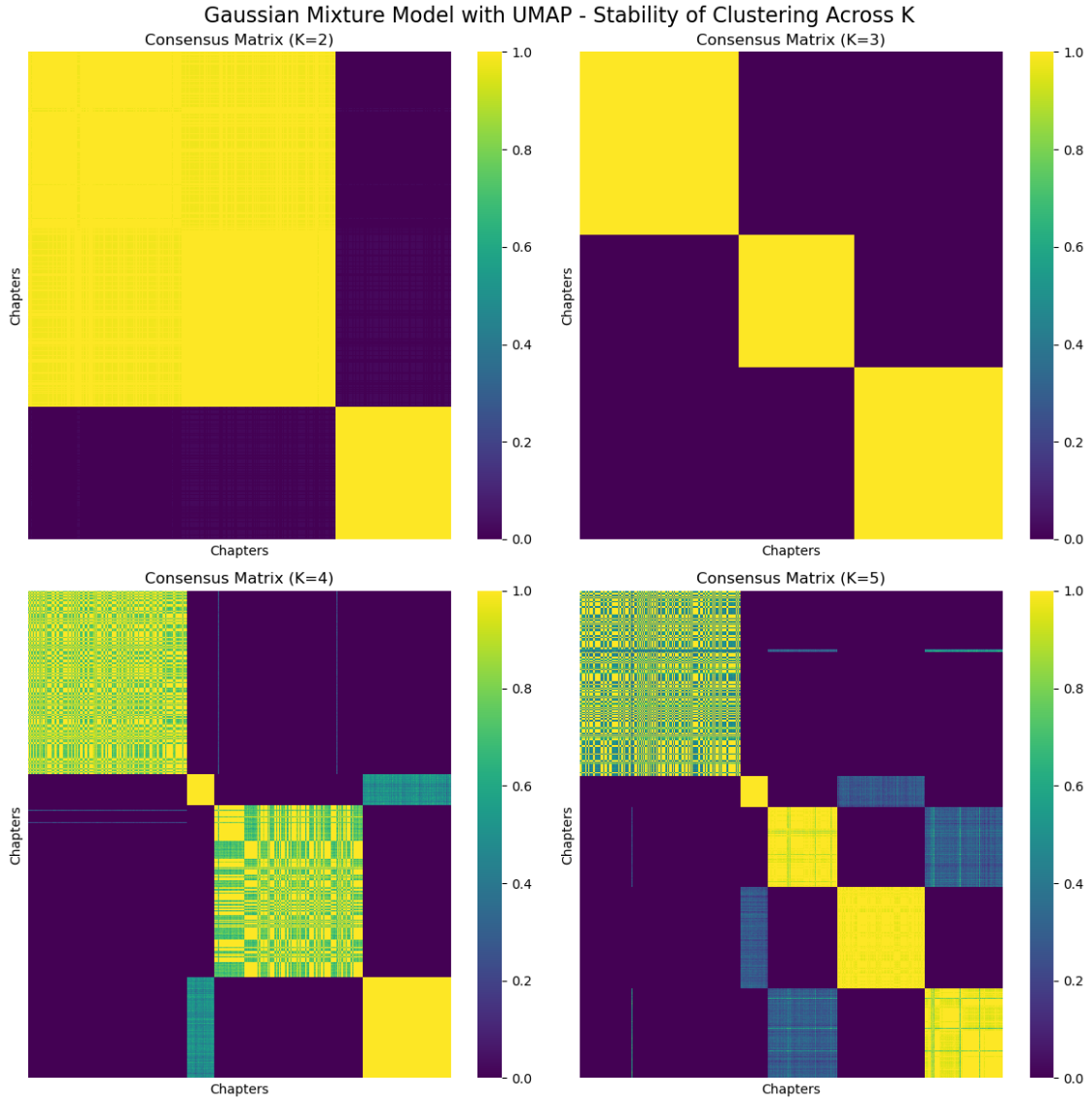
[15]:
```
method = 'Gaussian Mixture Model'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]  # NumPy version for GMM

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X).
↪predict(X)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
```

```
fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm')
plt.show()
```



[16]:
```
method = 'Gaussian Mixture Model with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()
```

```
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_sample = X_umap[idx_sample]

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X_umap).
 ↪predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm_umap')
plt.show()
```



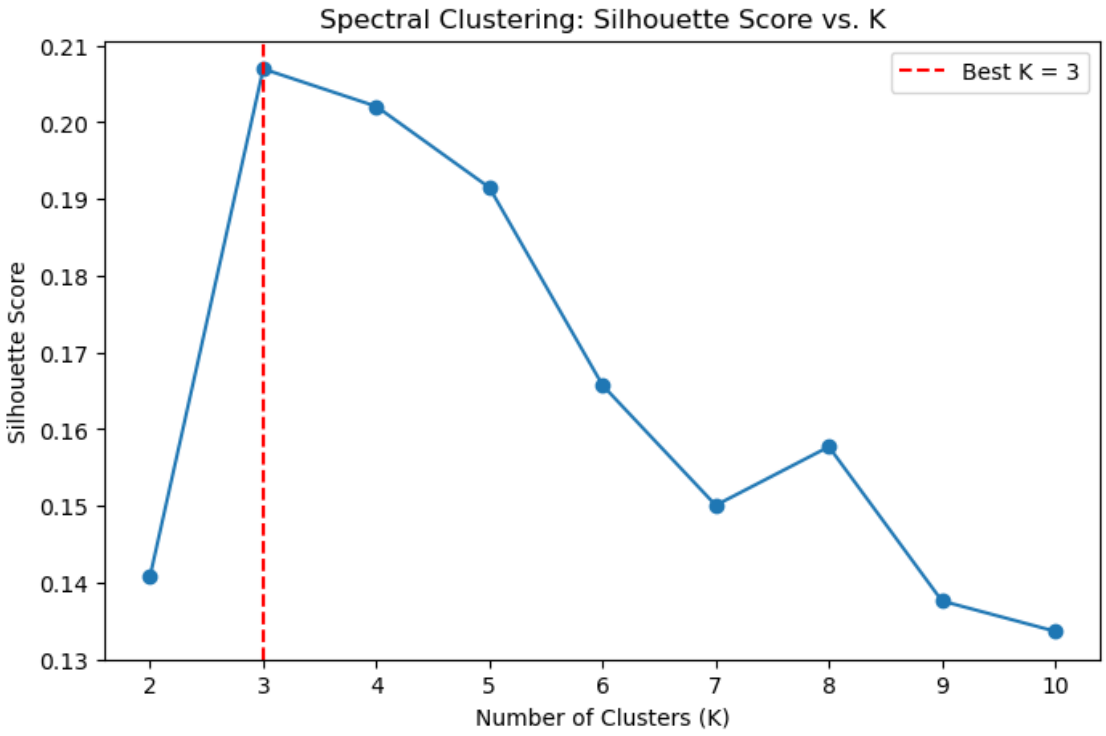
1.3 Spectral Clustering

1.3.1 Generalizibility

[17]:
```
print('\033[1m' + 'Spectral Clustering:' + '\033[0m')
evaluator_spectral = SilhouetteEvaluator(X, make_spectral(affinity =␣
 ↪'nearest_neighbors'), k_range=range(2, 11))
scores, best_k = evaluator_spectral.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

evaluator_spectral.plot('Spectral Clustering')
```

```
plt.savefig('Media/viz/05/05_spectral_silhouette_score')
```

```
Spectral Clustering:
The scores across K = {2: 0.1408300984760483, 3: 0.20696881798137212, 4:
0.2020762541851448, 5: 0.1914837165759898, 6: 0.16570630789830376, 7:
0.15006558354299535, 8: 0.1577000857169114, 9: 0.13758976585902763, 10:
0.1336400256361507}
Best K: 3
```



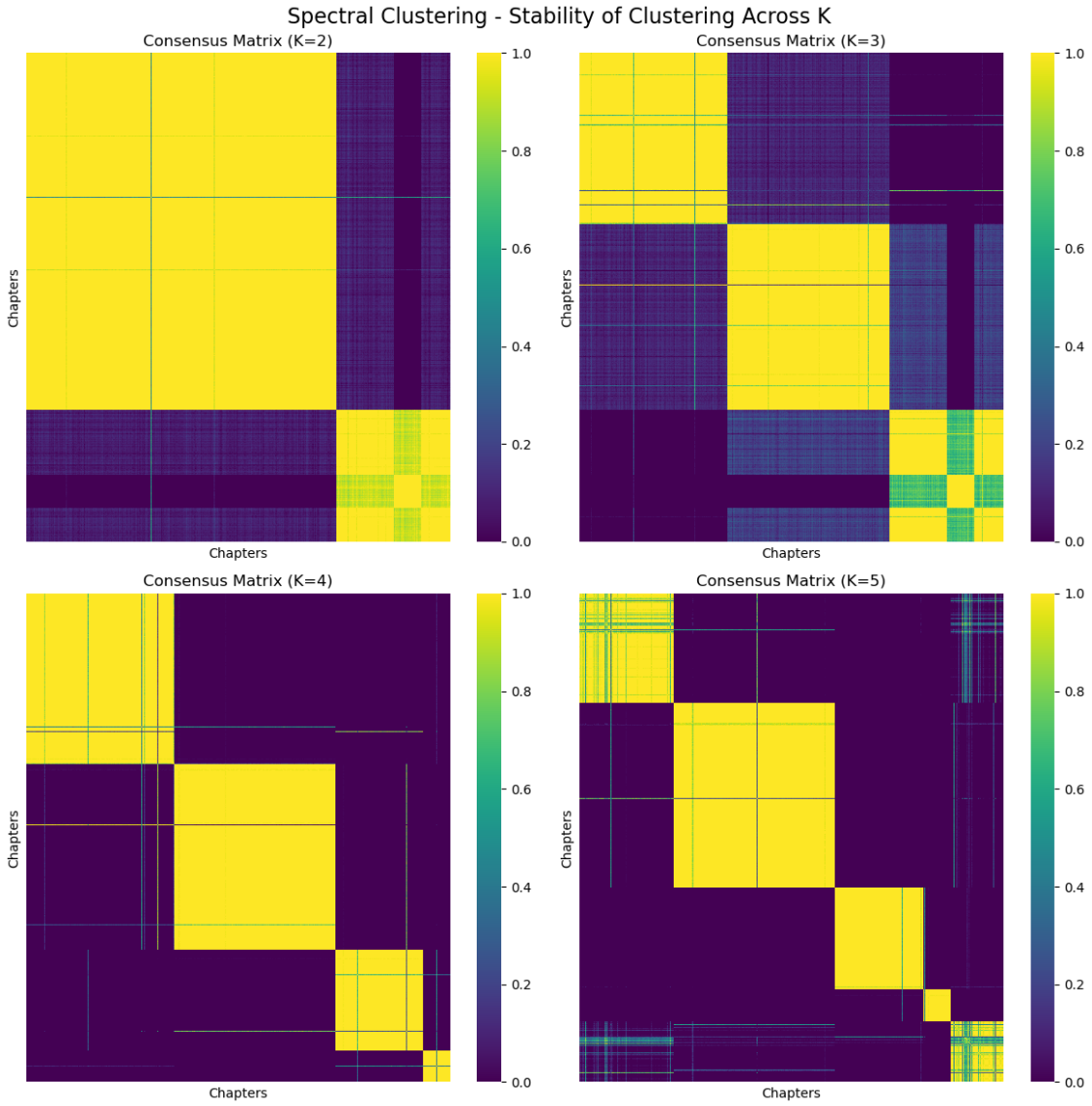
Although the silhouette score is commonly used to evaluate clustering performance, it assumes that clusters are spherical in shape, which does not align with the strengths of spectral clustering. Spectral clustering excels at uncovering non-linear and arbitrarily shaped cluster structures, which silhouette score is not well-equipped to quantify. Therefore, the low silhouette score observed here may understate the actual performance of spectral clustering. In fact, the spectral embedding reveals visually distinct and meaningful clusters (see notebook 02), suggesting that the method is effective despite the numerical metric.

1.3.2 Stability

[18]:
```
## METHOD
method = 'Spectral Clustering'

rng = np.random.default_rng(42)
iterations = 100
```

```
K_values = [2, 3, 4, 5]
n_samples = X_clean.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_train = X_clean.iloc[train_idx].to_numpy()

        sc = SpectralClustering(n_clusters=K, affinity='nearest_neighbors',␣
 ↪assign_labels='kmeans', random_state=n)
        cluster_labels = sc.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
 ↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = SpectralClustering(n_clusters=K,␣
 ↪affinity='nearest_neighbors', assign_labels='kmeans', random_state=42).
 ↪fit_predict(X_clean)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
 ↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
```

```
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_spectral')
plt.show()
```



$K = 4$ is the clear winner!

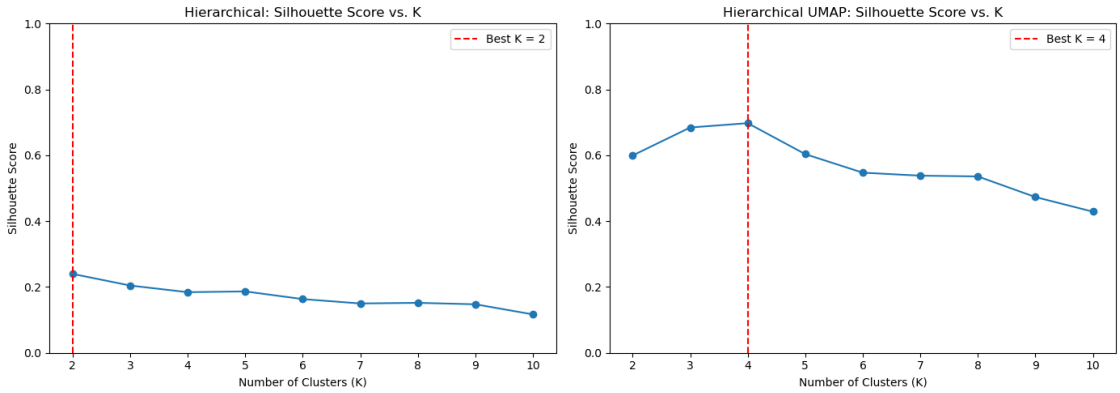
1.4 Hierarchial Clustering

Using the parameters 'ward' and 'euclidean' (because we have seen in notebook 04 that these have the highest accuracy)!

1.4.1 Generalizibility

[19]:
```
# Hierarchical Clustering
print('\033[1m' + 'Hierarchical Clustering without dimension reduction:' +␣
 ↪'\033[0m')
evaluator_hier = SilhouetteEvaluator(X, make_hierarchical(linkage='ward',␣
 ↪metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Hierarchical Clustering with UMAP
print('\033[1m' + 'Hierarchical Clustering with dimension reduction (UMAP):' +␣
 ↪'\033[0m')
evaluator_hier_umap = SilhouetteEvaluator(X_umap,␣
 ↪make_hierarchical(linkage='ward', metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_hier.plot("Hierarchical", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_silhouette_score_across_methods.png')
plt.show()
```

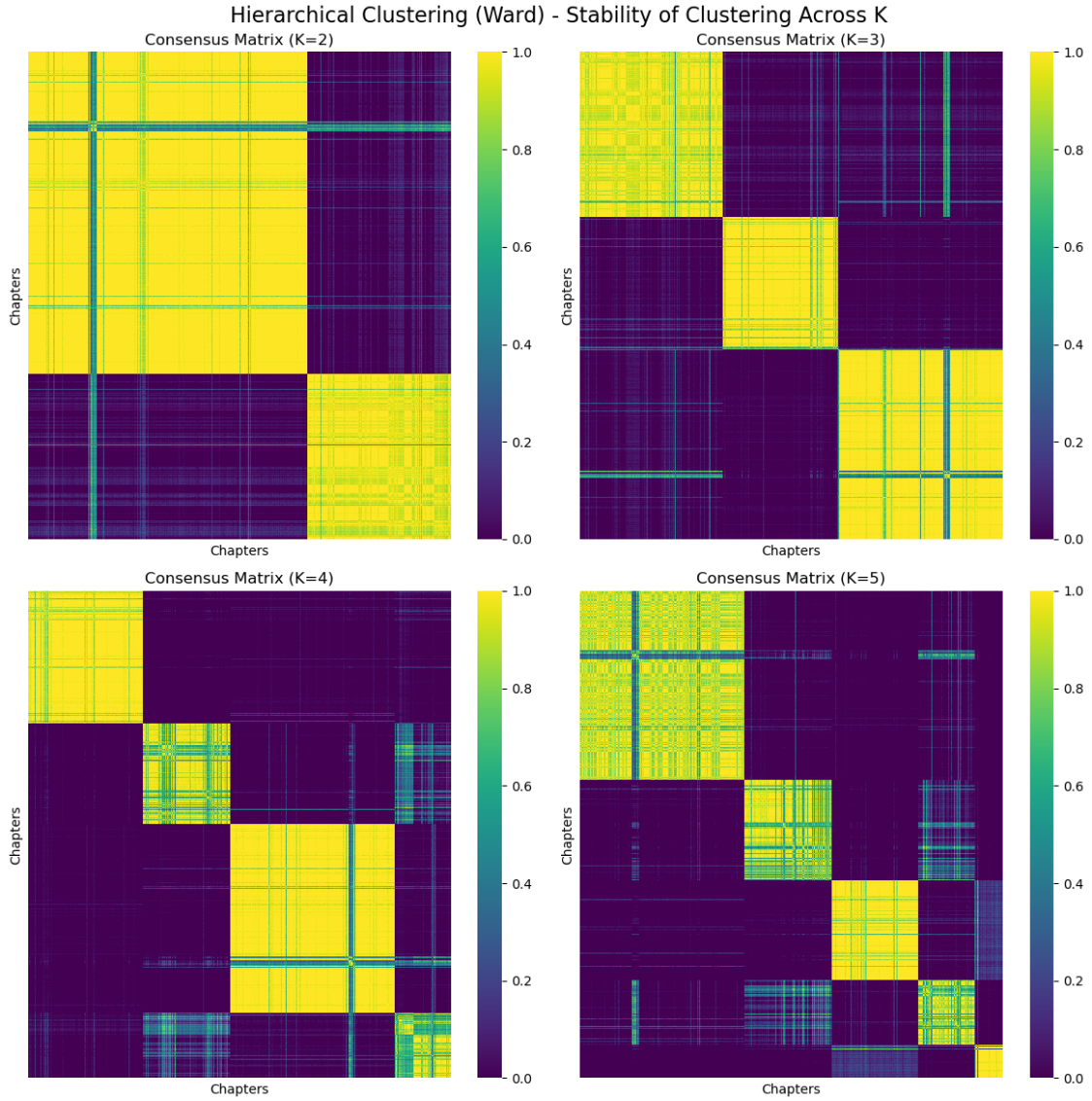
```
Hierarchical Clustering without dimension reduction:
The scores across K = {2: 0.23967574687100793, 3: 0.20437373269232859, 4:
0.18421197778513682, 5: 0.18654800991576861, 6: 0.16319550456880302, 7:
0.14995813568113975, 8: 0.1518500219855489, 9: 0.14745055233611987, 10:
0.1167264049290792}
Best K: 2
Hierarchical Clustering with dimension reduction (UMAP):
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60349095, 6: 0.5471597, 7: 0.5380937, 8: 0.53579855, 9: 0.47313103, 10:
0.42873582}
Best K: 4
```



1.4.2 Stability

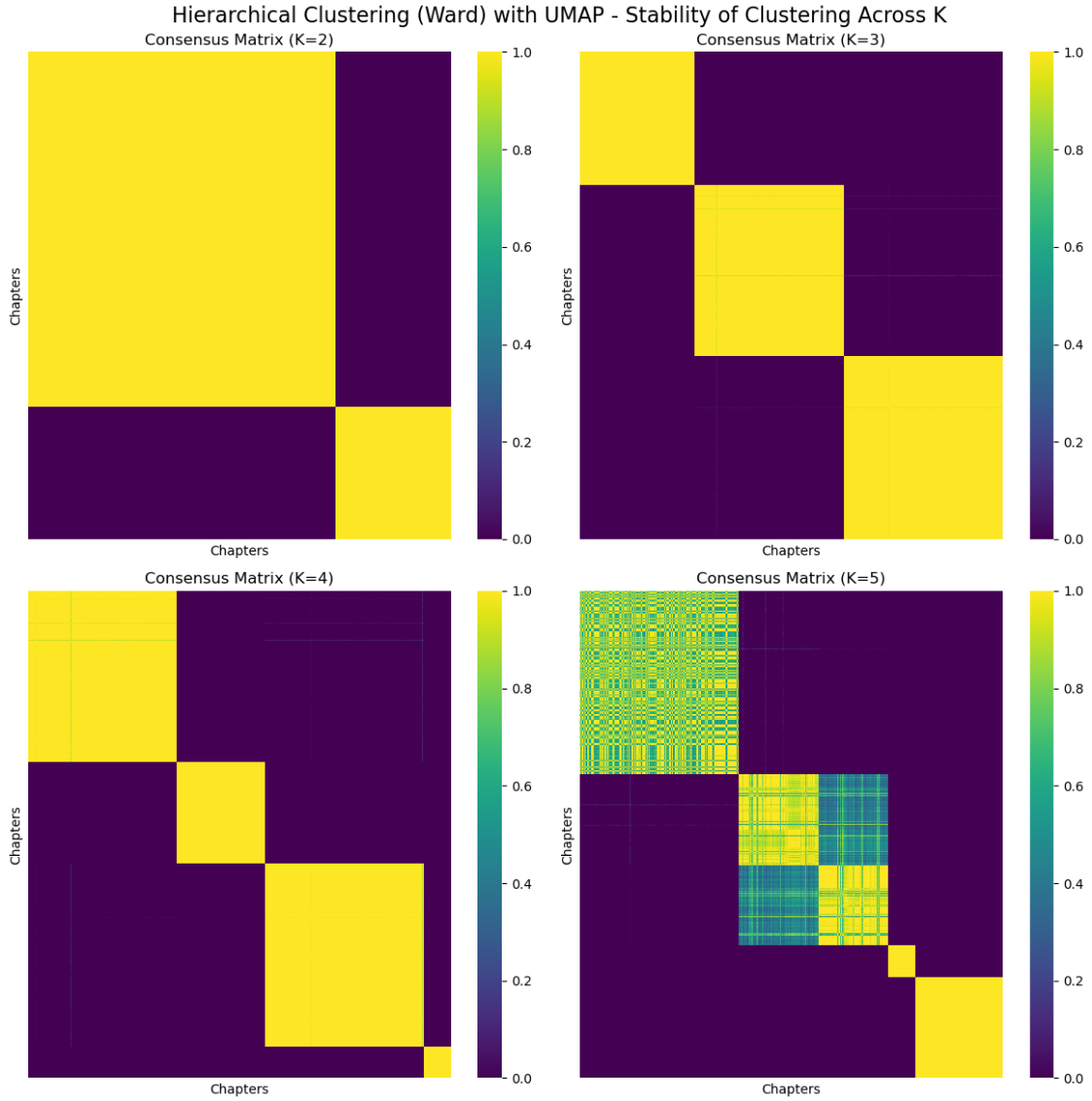
[20]:
```
method = 'Hierarchical Clustering (Ward)'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X)
```

```
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical')
plt.show()
```



[21]:
```
method = 'Hierarchical Clustering (Ward) with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
```

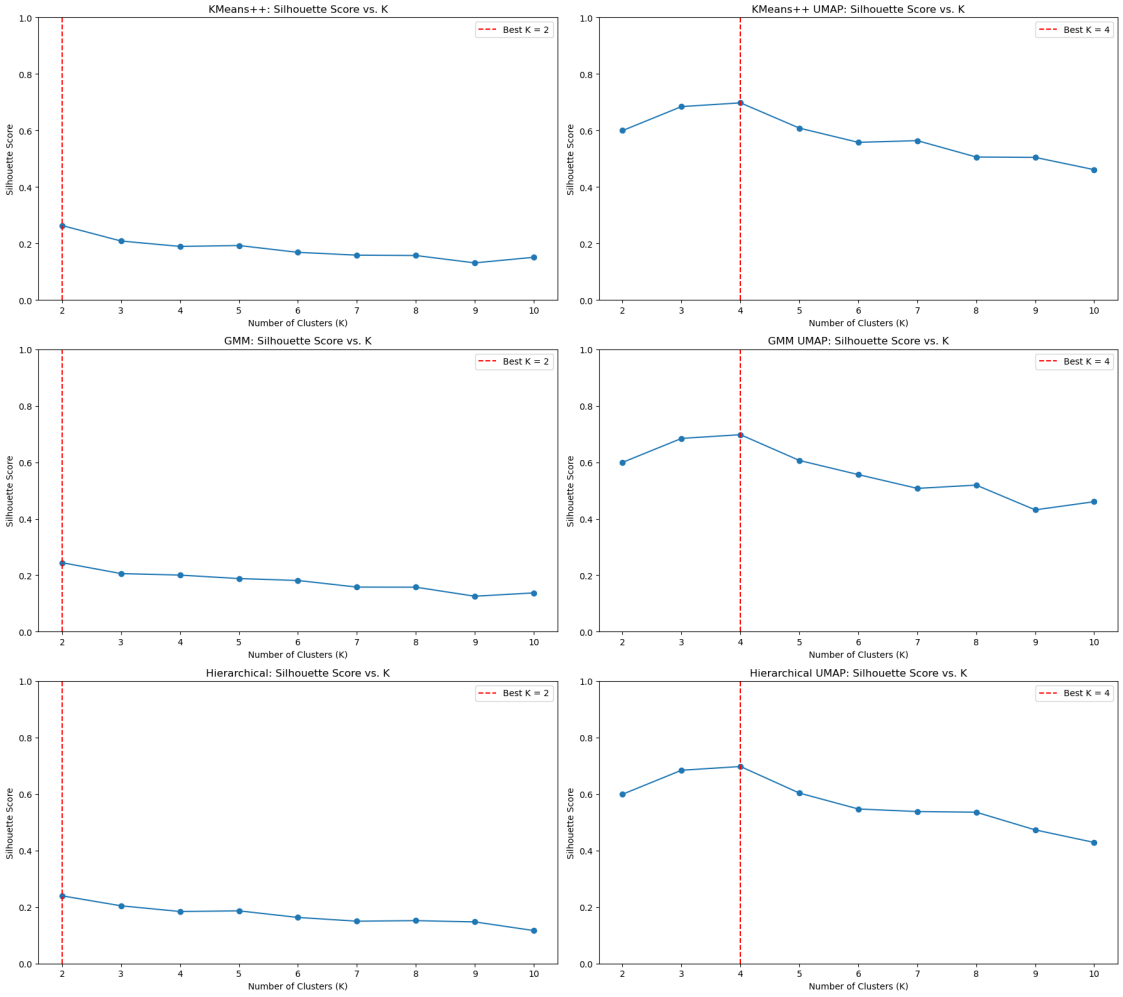
```
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical_umap')
plt.show()
```



2 Comparison Viz

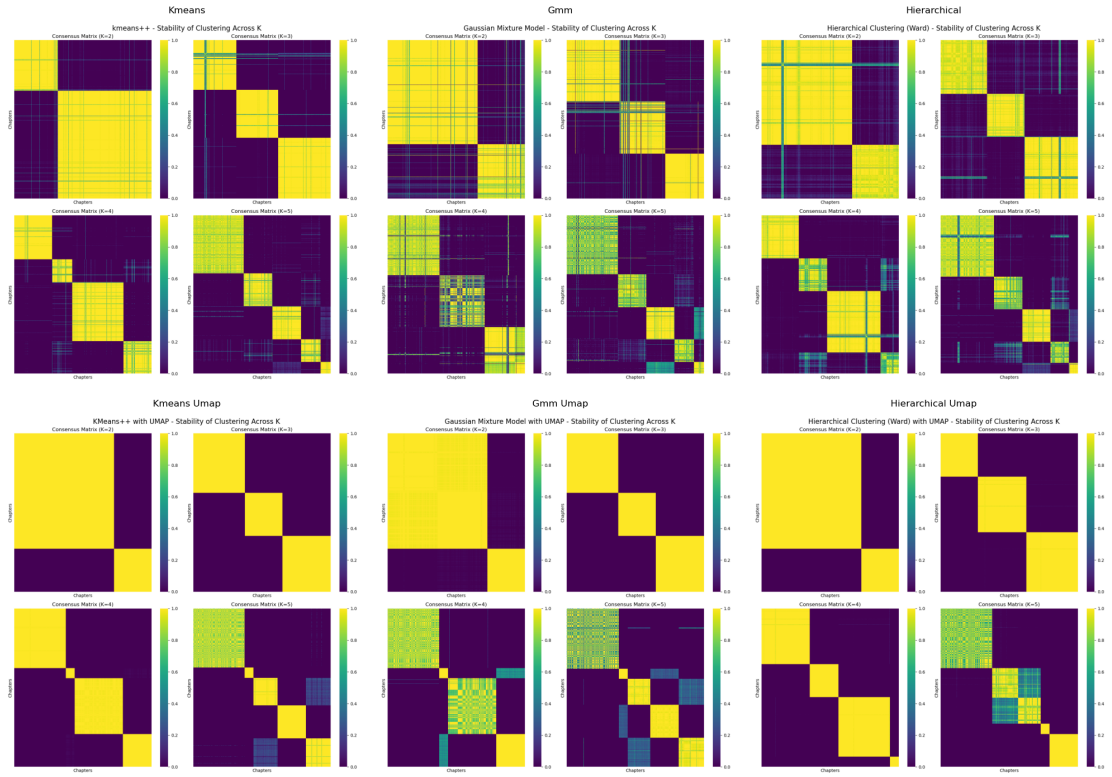
[22]:
```
fig, axs = plt.subplots(3, 2, figsize=(18, 16))

evaluator_kmeans.plot("KMeans++", ax=axs[0,0])
axs[0,0].set_ylim(y_min, y_max)
evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[0,1])
axs[0,1].set_ylim(y_min, y_max)

evaluator_gmm.plot("GMM", ax=axs[1,0])
axs[1,0].set_ylim(y_min, y_max)
```

```
evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1,1])
axs[1,1].set_ylim(y_min, y_max)

evaluator_hier.plot("Hierarchical", ax=axs[2,0])
axs[2,0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[2,1])
axs[2,1].set_ylim(y_min, y_max)

plt.savefig('Media/viz/05/05_silhouette_score_across_methods')
plt.tight_layout()
```



[26]:
```
heatmap_paths = [
    "Media/viz/05/05_consensus_heatmaps_kmeans.png",
    "Media/viz/05/05_consensus_heatmaps_gmm.png",
```

```
    "Media/viz/05/05_consensus_heatmaps_hierarchical.png",
    "Media/viz/05/05_consensus_heatmaps_kmeans_umap.png",
    "Media/viz/05/05_consensus_heatmaps_gmm_umap.png",
    "Media/viz/05/05_consensus_heatmaps_hierarchical_umap.png"
]

heatmap_images = [Image.open(path) for path in heatmap_paths]

fig, axes = plt.subplots(2, 3, figsize=(20, 14), constrained_layout=True)
axes = axes.flatten()
for ax, img, path in zip(axes, heatmap_images, heatmap_paths):
    ax.imshow(img)
    ax.set_title(
        os.path.basename(path)
          .replace("05_consensus_heatmaps_", "")
          .replace(".png", "")
          .replace("_", " ")
          .title(),
        fontsize=12
    )
    ax.axis('off')
plt.savefig("Media/viz/05/05_all_consensus_heatmaps_grid.png")
plt.show()
```



06_em_algorithm_fit_gmm

April 17, 2025

Load necessary libraries.

[2]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

### ML packages
from scipy.stats import multivariate_normal
from sklearn.mixture import GaussianMixture
import umap
from sklearn.metrics import adjusted_rand_score
from scipy.spatial.distance import cdist

### Msc
import warnings

### OOP Code
from ml_utils import GaussianMixtureEM
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array
X = df.drop(columns=['Author'])

# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = pd.DataFrame(umap_model.fit_transform(X))
```

EM Algorithm for Multivariate Gaussian Mixture Model

Let the density of $x \in \mathbb{R}^d$ be: $p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x \mid \mu_k, \Sigma_k)$

where: - π_k are the mixing proportions (with $\sum_k \pi_k = 1$), - $\mu_k \in \mathbb{R}^d$ is the mean of component k, - $\Sigma_k \in \mathbb{R}^{d \times d}$ is the covariance matrix.

E-step

$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i|\mu_j,\Sigma_j)}$ where $\mathcal{N}(x \mid \mu, \Sigma)$ is the multivariate Gaussian density: $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i$

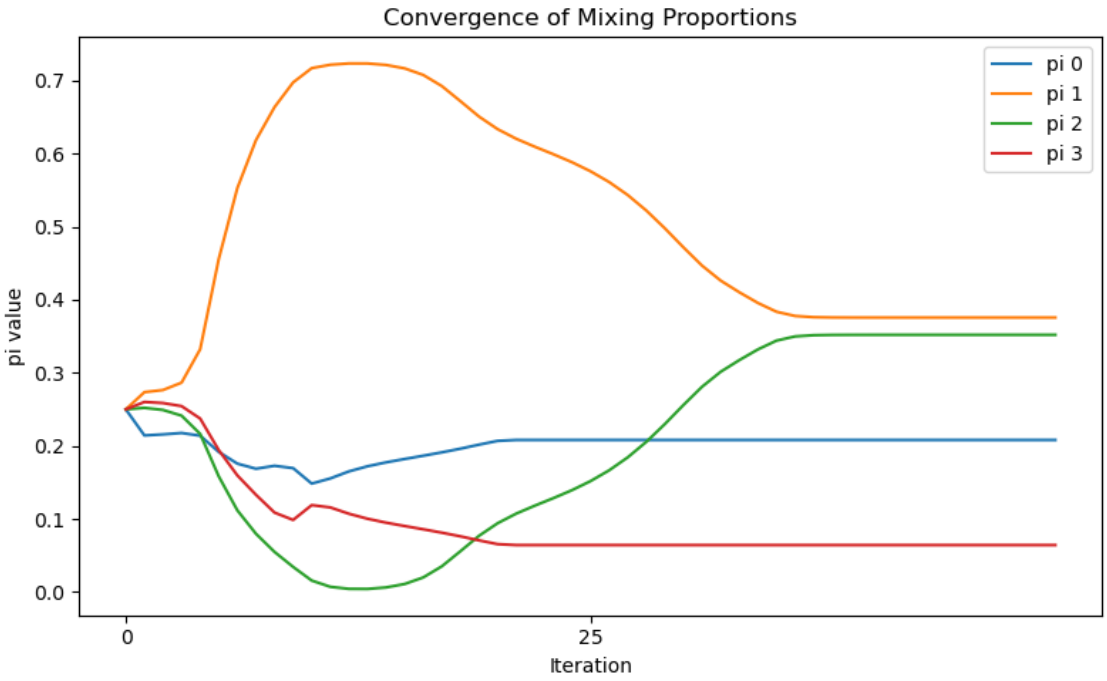
Update covariance: $\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

Update mixing weights: $\pi_k = \frac{N_k}{N}$

Repeat E-step and M-step until convergence (e.g., change in log-likelihood is below a threshold).

[4]:
```
# code in -> ml_utils.py
# code also pasted at appendix
model = GaussianMixtureEM(K=4, num_iterations=50, allow_singular=False)
results = model.fit_fast(X_umap)
```

[5]:
```
pis_dict = results['pis_dict']
iterations = [key for key in pis_dict if key.startswith('Iteration_')]
iterations_sorted = sorted(iterations, key=lambda x: int(x.split('_')[1]))

pi_matrix = np.array([pis_dict['Initial'][0]] + [pis_dict[it][0] for it in␣
 ↪iterations_sorted])

plt.figure(figsize=(8, 5))
for k in range(pi_matrix.shape[1]):
    plt.plot(range(len(pi_matrix)), pi_matrix[:, k], label=f'pi {k}')

plt.title('Convergence of Mixing Proportions')
plt.xlabel('Iteration')
plt.ylabel('pi value')

steps = 25
plt.xticks(range(0, len(iterations_sorted), steps))

plt.legend()
plt.tight_layout()
plt.savefig('Media/viz/06/06_em_convergence_plot')
plt.show()
```



1 Compare

Now to compare my manually EM function to the built in Sklearn package!

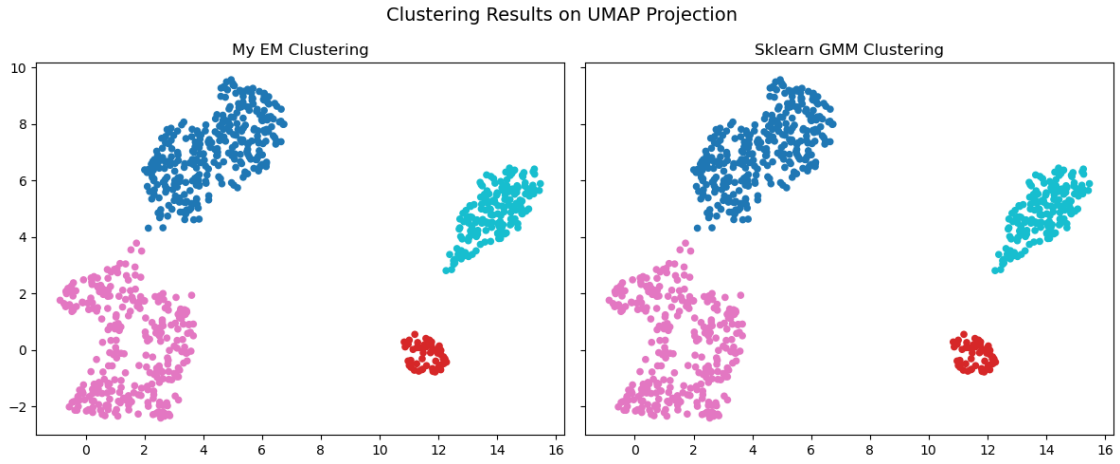
[6]:
```
sk_model = GaussianMixture(n_components=4, random_state=42)
sk_model.fit(X_umap)

# Your final results
my_pi = results['pis_dict'][f'Iteration_{model.num_iterations - 1}'][0]
my_mu = np.array(results['means'])
my_cov = np.array(results['cov'])

# Sklearn results
sk_pi = sk_model.weights_
sk_mu = sk_model.means_
sk_cov = sk_model.covariances_
```

[7]:
```
dist_matrix = cdist(my_mu, sk_mu)
matches = np.argmin(dist_matrix, axis=1)

print("Matching components based on mean proximity:")
for i, j in enumerate(matches):
    print(f"Component {i} → Sklearn {j} | Distance: {dist_matrix[i, j]:.4f}")
```

```python
print("\nMixing weights comparison:")
for i, j in enumerate(matches):
    print(f"Component {i}: mine={my_pi[i]:.4f} | sklearn={sk_pi[j]:.4f}")

print("\nMean vector L2 distances:")
for i, j in enumerate(matches):
    delta = np.linalg.norm(my_mu[i] - sk_mu[j])
    print(f"Component {i}: || _mine -  _sklearn|| = {delta:.4f}")

my_labels = np.argmax(results['gamma'], axis=1)
sk_labels = sk_model.predict(X_umap)
ari = adjusted_rand_score(my_labels, sk_labels)

print(f"\nAdjusted Rand Index: {ari:.4f}")

remapped_labels = np.zeros_like(my_labels)
for i, j in enumerate(matches):
    remapped_labels[my_labels == i] = j

X_umap_np = X_umap.to_numpy() if isinstance(X_umap, pd.DataFrame) else X_umap

fig, axs = plt.subplots(1, 2, figsize=(12, 5), sharex=True, sharey=True)

axs[0].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=remapped_labels,␣
 ↪cmap='tab10', s=20)
axs[0].set_title("My EM Clustering")

axs[1].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=sk_labels, cmap='tab10',␣
 ↪s=20)
axs[1].set_title("Sklearn GMM Clustering")

plt.suptitle("Clustering Results on UMAP Projection", fontsize=14)
plt.tight_layout()
plt.savefig('Media/viz/06/06_label_2dim_comparison_viz')
plt.show()
```

```
Matching components based on mean proximity:
Component 0 → Sklearn 3 | Distance: 0.0000
Component 1 → Sklearn 0 | Distance: 0.0000
Component 2 → Sklearn 2 | Distance: 0.0001
Component 3 → Sklearn 1 | Distance: 0.0000

Mixing weights comparison:
Component 0: mine=0.2081 | sklearn=0.2081
Component 1: mine=0.3757 | sklearn=0.3757
Component 2: mine=0.3520 | sklearn=0.3520
Component 3: mine=0.0642 | sklearn=0.0642
```

```
Mean vector L2 distances:
Component 0: || _mine -  _sklearn|| = 0.0000
Component 1: || _mine -  _sklearn|| = 0.0000
Component 2: || _mine -  _sklearn|| = 0.0001
Component 3: || _mine -  _sklearn|| = 0.0000

Adjusted Rand Index: 1.0000
```

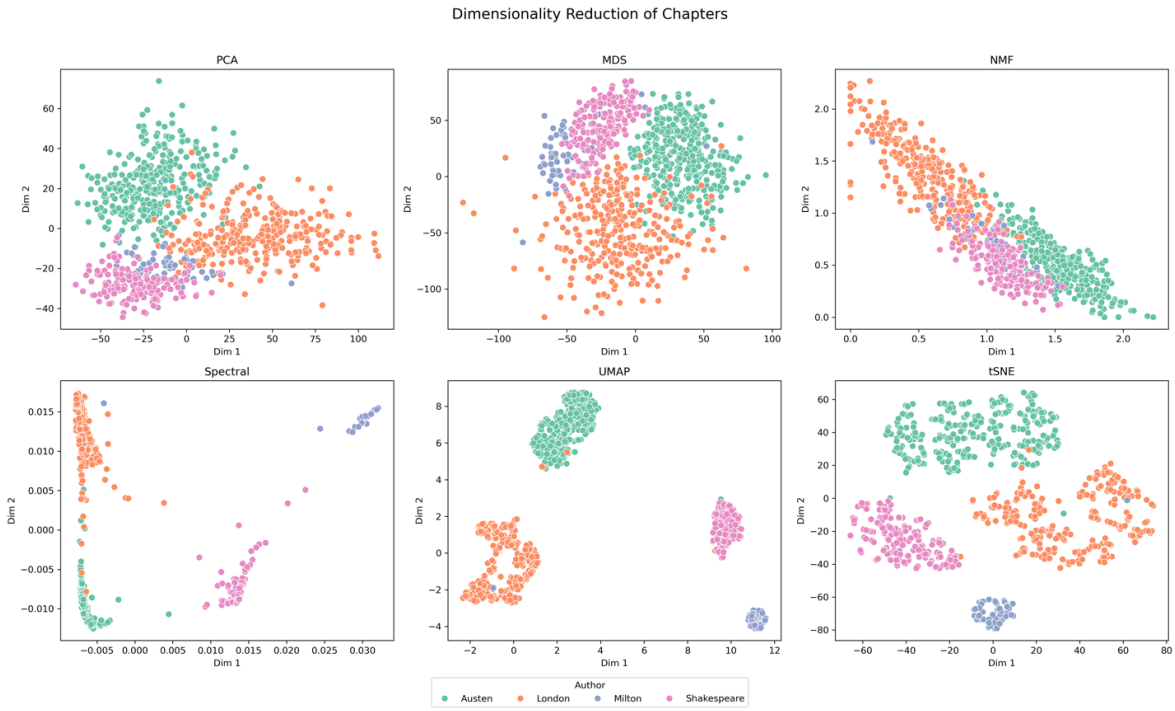


Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!

Visualizations

■ *observations (book chapters)*

Implementing PCA, MDS, NMF, Spectral Embedding, UMAP, t-SNE and validating with author labels (observations = chapters) gives the following visualizations.

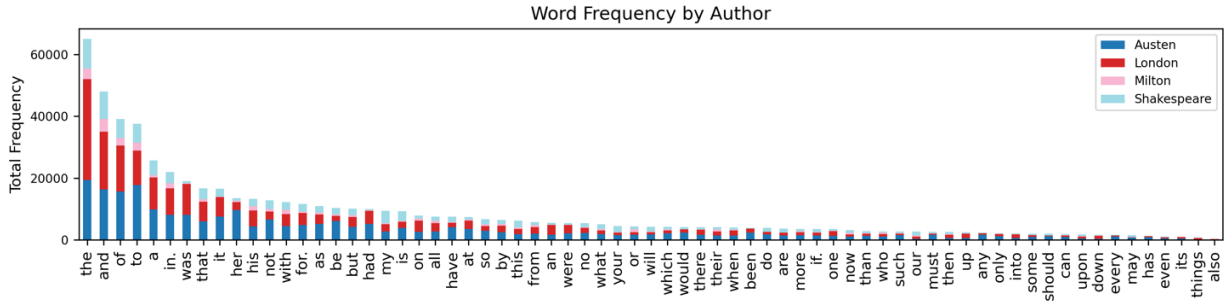


The linear methods perform worse in creating distinguishable clusters (without labels to validate PCA, MDS, and NMF would just look like "blobs" of observations) leading to more overlap and less informative projections; the structure of the data appears to be non-linear, evidenced by the superior performance of non-linear dimensionality reduction techniques. Among these, UMAP stands out—it produces well-separated, spherical clusters that significantly enhance interpretability and visual clarity. Compared to other non-linear approaches like t-SNE and Spectral Embedding, UMAP strikes a balance between global and local structure, revealing distinct author-based groupings.

In terms of interpretability, the visual above supports the claim that generally for this dataset (among the tested methods) the ***'best' interpretation = UMAP***.

■ *features (words)*

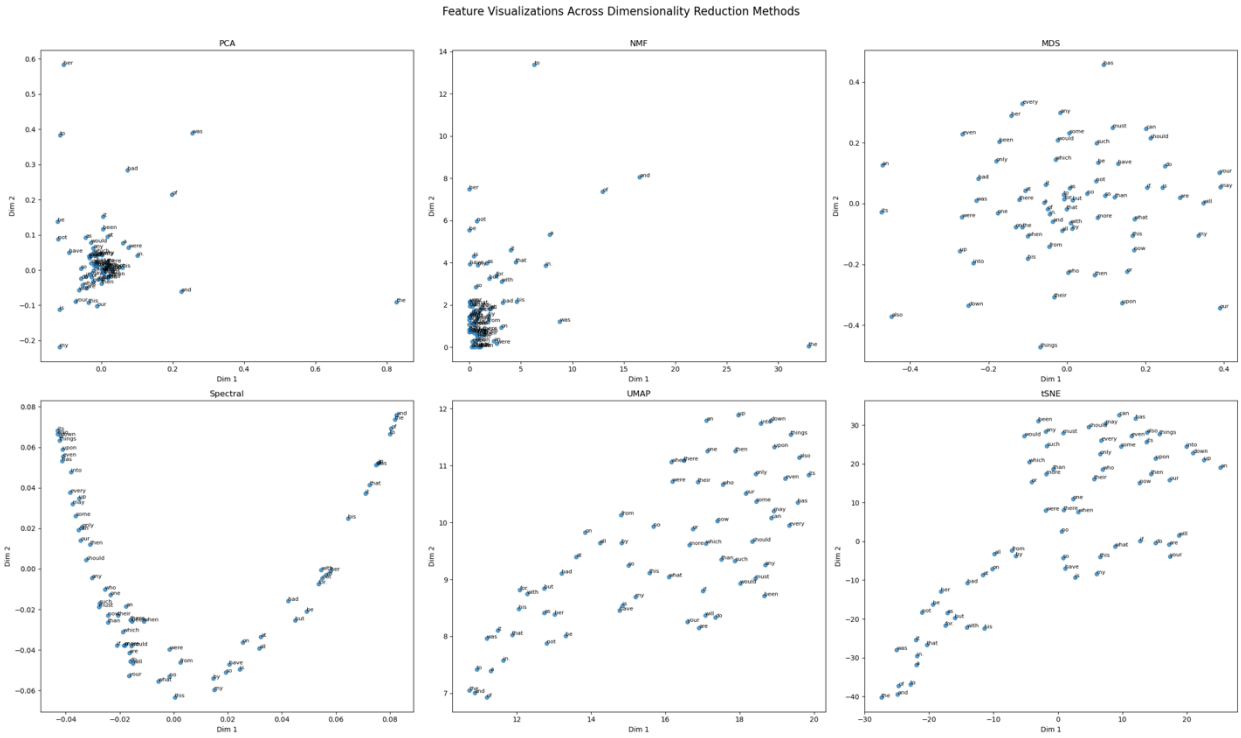
First off, the simplest visualization for the features/words is just a bar plot of word count across words!



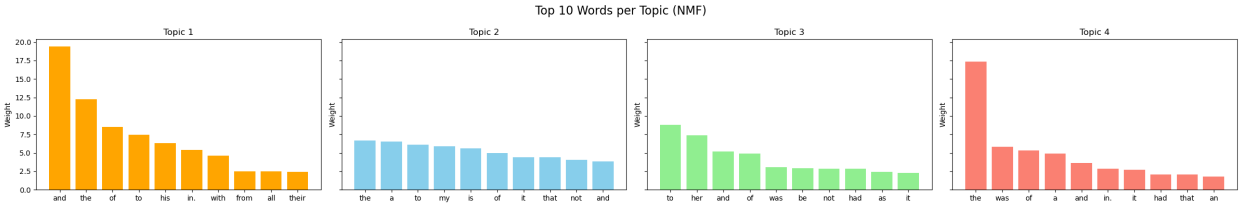
To further analyze patterns among features, I used their co-occurrences across chapters, without knowledge of authorship, to explore whether semantically or stylistically coherent word groups emerge. NMF is particularly well-suited for analyzing features (words) in an unsupervised

setting because it produces a parts-based, non-negative decomposition of the document-term matrix, allowing each topic to be represented as an additive combination of real words. This leads to intuitive and interpretable results, where the top contributing words for each topic reveal coherent semantic or stylistic themes—such as narrative tone, reflective voice, or dialogue. In contrast, methods like PCA yield components with mixed signs that are harder to interpret, and nonlinear methods like UMAP and Spectral Embedding embed words in low-dimensional space without preserving clear word-level semantics. Unlike these methods, NMF enables both dimensionality reduction and human-interpretable insights, making it the most effective choice for visualizing and understanding word-level structure in the data.

For example, visualizations on the features can be given by the following (below). Although this provides some information it is not evident and easily interpreted. So this method is **not** preferable for visual interpretation!



Going back to analyzing the NMF topics for semantic themes we can provide the following visualization of the top 10 words per topic (highest weight). This visualization in combination with the topic weights per chapter

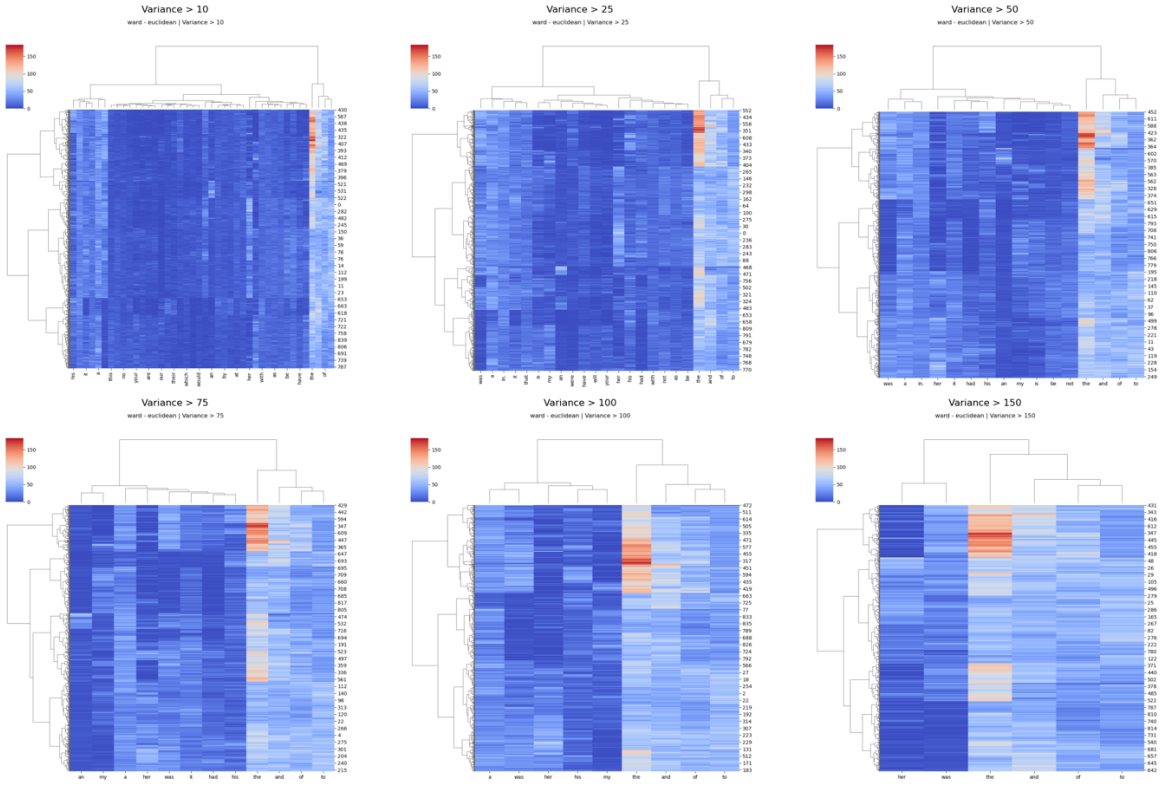


Using this it is far easier to establish semantic themes and distinguish the chapters from another according to which chapters are composed of which topics! The four topics extracted via NMF reveal distinct semantic patterns. **Topic 1** appears to reflect structural scaffolding of narrative prose, with high-weighted function words like *and*, *the*, *of*, *to*, and *with* suggesting continuous sentence construction, descriptive flow, and background exposition. **Topic 2** leans into a more introspective or active narrative stance, driven by frequent use of *my*, *is*, *to*, and *that*, pointing toward subjectivity, dialogue, or first-person accounts. **Topic 3** evokes personal and relational storytelling, emphasized by words such as *to*, *her*, *was*, *be*, and *had*, which are often found in emotionally driven or character-focused scenes. Lastly, **Topic 4** is anchored in definitive and declarative language—*the*, *was*, *and*, *had*, and *it*—potentially signaling reflective, philosophical, or climactic exposition. Together, these topic-word distributions highlight nuanced stylistic layers across authors, helping distinguish thematic patterns embedded in the text. This visual alone provides significant interpretation about the features.

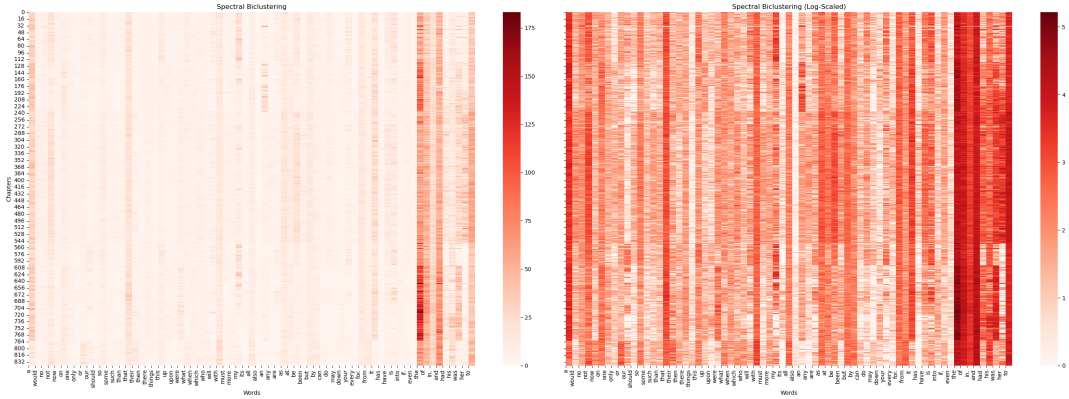
In terms of interpretability, the ***'best/great' interpretation*** is provided by ***NMF***. Now this leads to **1Ac** where we tie together these trends in words to the features which provides further value to interpretation!

■ *both - observations & features*

To visualize both observations and features I used Spectral Bi-clustering and Hierarchical Bi-clustering. To reduce noise in the heatmaps, I filtered out the highest variance features based on some threshold. I have plotted the features retained for each threshold across multiple thresholds below. The x-axis contains the words with high variance across chapters where significant chapters are plotted on the y-axis. If we look at the sparsest heatmap, for the highest threshold of variance > 150 (bottom right) we see bands emerge which provide significant interpretations. For example, if we go back to the NMF topics, for the word '*the'* we can see the red band for chapters 347 and 445 meaning it is likely that these chapters contain topic 1 or topic 4!



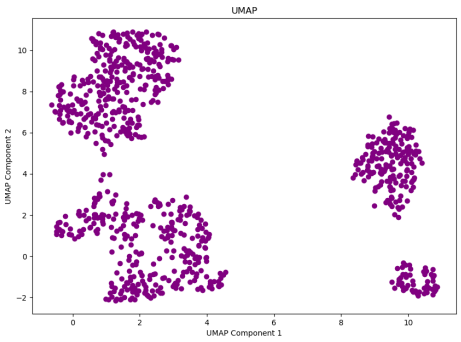
In other words, the features that have high contract among chapters allows us to interpret potential themes/topics among the chapters! I have also produced a heatmap using the Spectral Bi-clustering method which provided a decent visualization. However, clearly Hierarchical Bi-clustering outperforms this method.

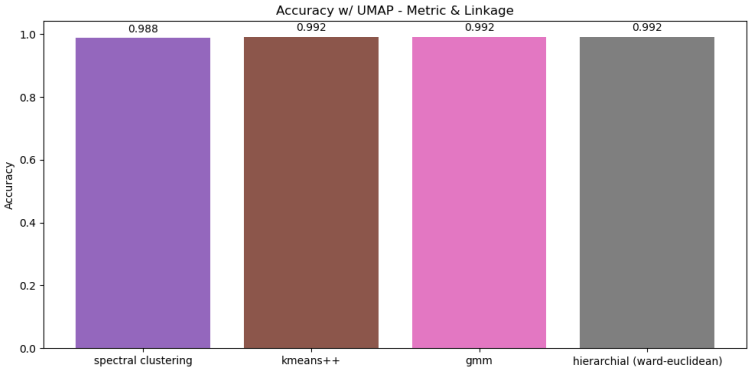


In terms of interpretability, the ***'best/great' interpretation*** is provided by ***Hierarchical Bi-clustering'***.

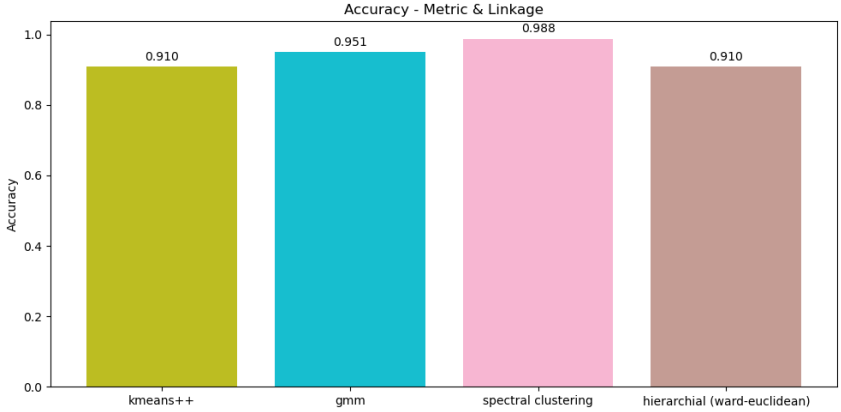
Clustering Methods

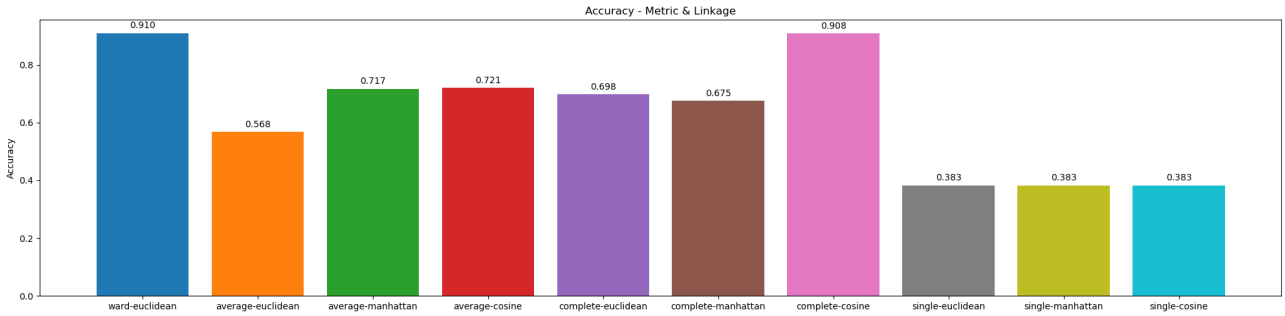
Since UMAP proved to be the superior dimension reduction technique for this dataset, I compared the clustering accuracy across methods on the UMAP transformed data (except for spectral clustering as this already has a built-in dimension reduction – spectral embedding). The results were not surprisingly the same across all methods at 0.992 because UMAP did such a good job at creating clusters that there were no points where the methods would diverge in predicting label outcomes! To make it evident, look at the UMAP data (without labels) – there are 4 clusters, and which points belongs to which clusters! Therefore, apply any methods to UMAP will yield the same labels for each point.





Now spectral clustering performs spectral embedding, so the method accuracy diverges from the methods processed on UMAP transformed data. Since UMAP dimension reduction made comparison redundant, I applied the same techniques to the raw dataset! Furthermore, I ran hierarchical clustering across many hyperparameter values for metrics and linkage.

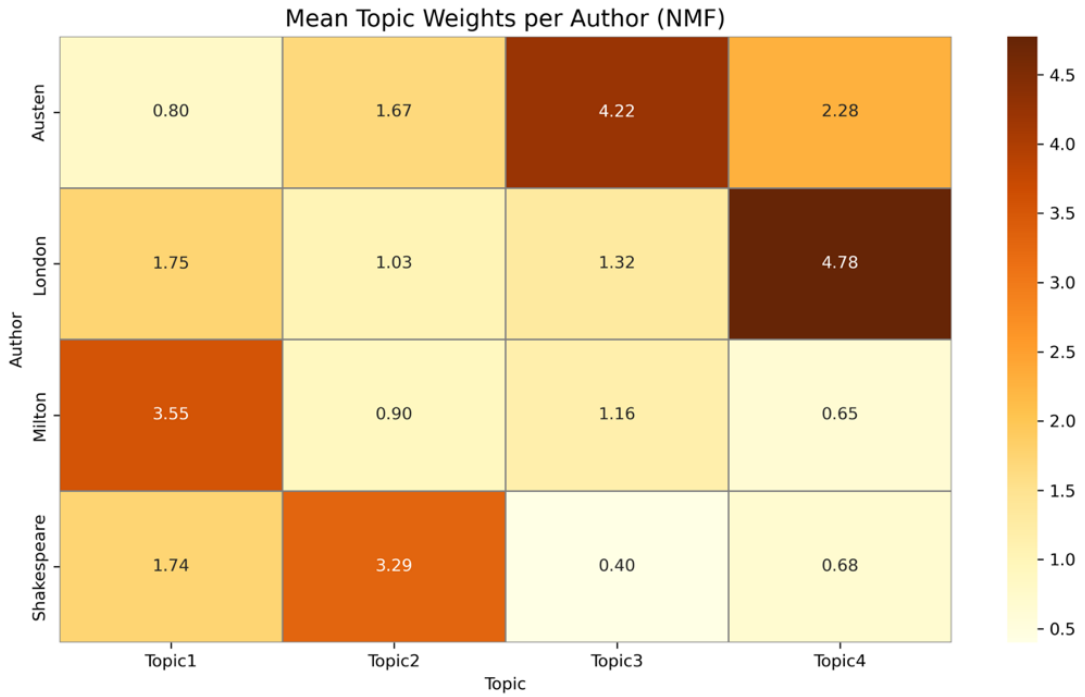




Without UMAP, performance dropped slightly yet still maintained strong results (of course spectral embedding yielded the same results). Among the metric and linkage parameters, Ward linkage with Euclidean distance yielded the highest accuracy, rivaling the performance of UMAP-based methods, while others like single-linkage performed poorly. Overall, UMAP significantly improves clustering accuracy and consistency, though the choice of metric and linkage remains critical when UMAP is not used

Pattern Recognition

Now using supervised learning to predict the author given the distribution of words per chapter! For this task I went back to the NMF topics because they revealed a lot about the dataset. I aimed to identify stylistic or feature-based patterns across the authors using an unsupervised learning approach. Recall that I applied NMF to uncover latent topics from the word frequency data, generating a document-topic matrix representing how strongly each chapter aligns with each topic. To determine which words were important for each topic, I visualized the top 10 words for each component, revealing clear semantic themes. Now using that work, I grouped by author and computed the mean topic weights and visualized them as a heatmap.



This shows that while no topic is exclusive to a single author, there are strong preferences — for instance, ***Milton* scores highest on Topic 1, *Shakespeare* on Topic 2, *Austen* on Topic 3, and *London* on Topic 4**. Since each author has a distinct #1 topic, I am inclined to predict the chapter based on this simple mapping of author to highest topic weight, i.e,

```
mapping = {'Topic1': 'Milton', 'Topic2': 'Shakespeare', 'Topic3': 'Austen', 'Topic4': 'London'}
```

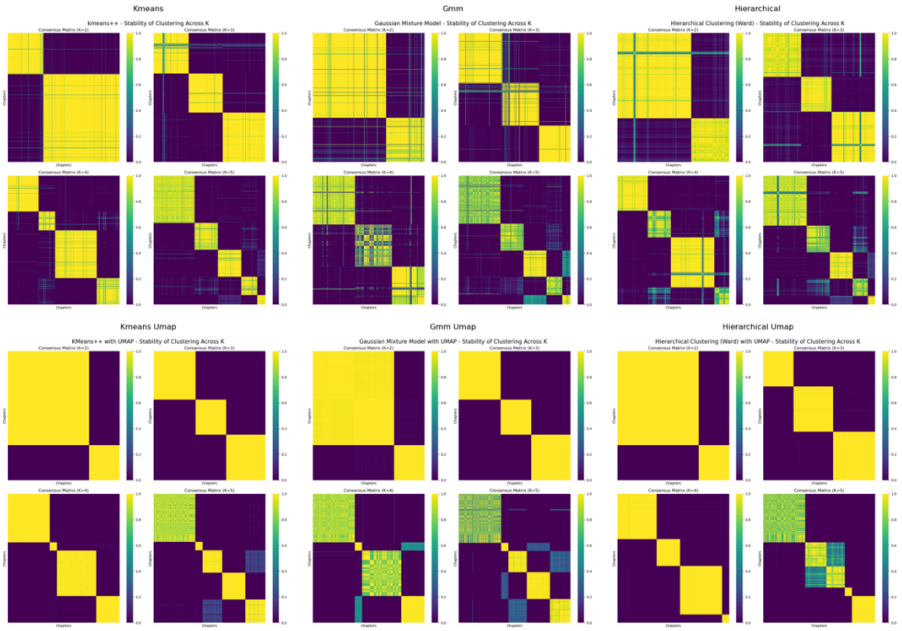
This representation suggests distinct stylistic or grammatical tendencies across authors. To evaluate how informative these patterns are, I split the data into **80% training** and **20% testing**, learned a mapping from dominant topic to author based on training data, and predicted authors on the test set using only the highest-weighted topic. This method correctly identified the author **91.7% of the time**, demonstrating that the unsupervised topic distributions captured meaningful, distinguishable patterns in writing style across authors. Now, we can claim that if the word distribution within a chapter aligns strongly with a topic, we can map that chapter to one of the known authors!

```
top_10_words_per_topic = {'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all', 'their'],
                          'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not', 'and'],
                          'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as', 'it'],
                          'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that', 'an']
}
```

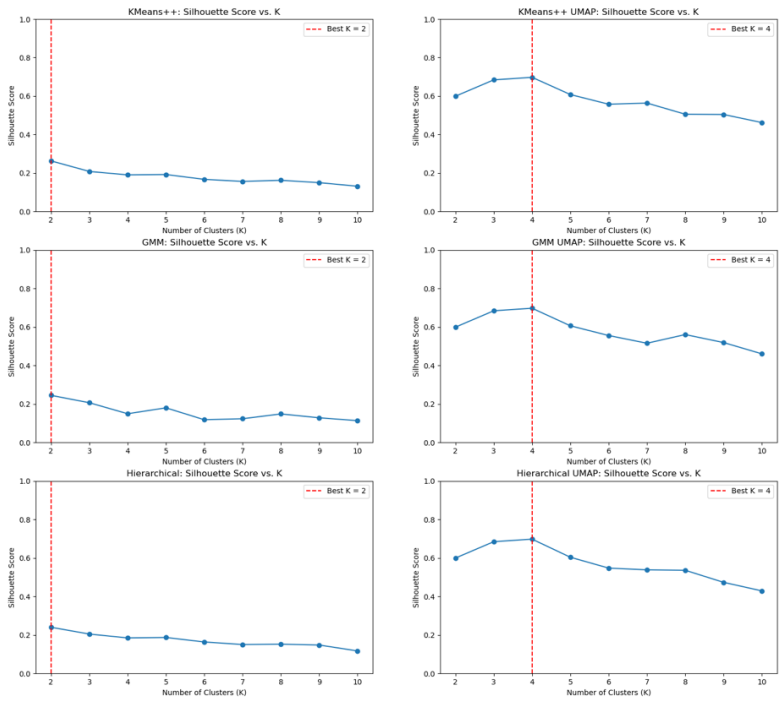
The words that are significant within topics and topics significant across authors provide predictive capabilities!

Validation

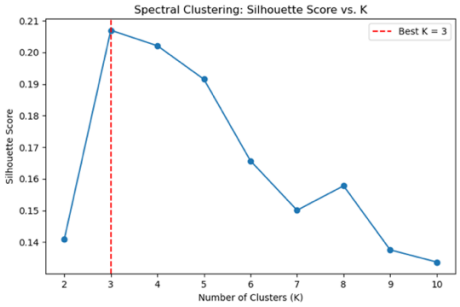
To validate our hyperparameter K without author labels, I used generalizability and stability as metrics.

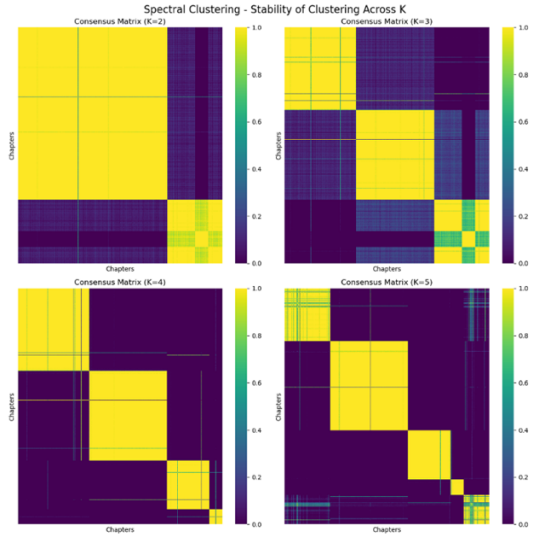


To gain insights to the generalizability, I calculated the silhouette scores across all methods (with and without the dimension reduction UMAP). From the visualization (right) you can see that before applying UMAP the silhouette score was very low across all K indicating but after applying UMAP, the silhouette score drastically increased and peaked at $K = 4$ across Kmeans++, GMM, and Hierarchial Clustering! To evaluate the stability, I used a consensus matrix, visualized above. GMM yielded a poor consensus matrix (likely due to problems in the initialization) but in combination with the silhouette scores we can confidently support the claim that $K = 4$! Furthermore, if we base our hyperparameter validation across all methods the we see a trend in stability score peaking at $K = 4$ and the consensus matrix showing strong stability with 4 clusters.



Spectral clustering can perform poorly under stability analysis because it is highly sensitive to small perturbations in the data or similarity matrix. Unlike centroid-based methods such as K-Means, which rely on explicit distance metrics and generally converge to consistent partitions, spectral clustering depends on the eigen structure of a graph Laplacian derived from data affinities. Even slight changes in the dataset can significantly alter the eigenvectors, leading to different cluster assignments across runs. As a result, traditional stability metrics may underestimate the effectiveness of spectral clustering, not due to poor clustering performance, but due to the method's inherent sensitivity to initialization and graph construction parameters. This is demonstrated below, as the silhouette scores all appeared very low and that it "peaked" at $K = 3$ but very marginally. Inspecting the $K = 3$ consensus matrix makes it clear that the stability for $K = 3$ is very poor and so it makes more sense to select the next highest silhouette score at $K = 4$ (which is essentially the same value as the stability score for $K = 3$). This deduction aligns with all the other methods.





In conclusion, Hierarchial Clustering and Kmeans++ are the most stable and generalize the best for this dataset! The results of analyzing the stability and generalizability lead to a clear hyperparameter of $K = 4$ clusters.

EM Algorithm

The EM algorithm for the Multivariate Gaussian Mixture Model

Start - Initialize

$\mu_k^{(0)} \sim$ Uniform sample from $\{x_1, \dots, x_N\}$, $\forall k \in \{1, \dots, K\}$

$\Sigma_k^{(0)} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$, $\forall k \in \{1, \dots, K\}$

$\pi_k^{(0)} = \frac{1}{K}$, $\forall k \in \{1, \dots, K\}$

E-step

$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$ where $\mathcal{N}(x_i \mid \mu_j, \Sigma_j)$ is the multivariate Gaussian density $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik} x_i$

Update covariance: $\Sigma_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

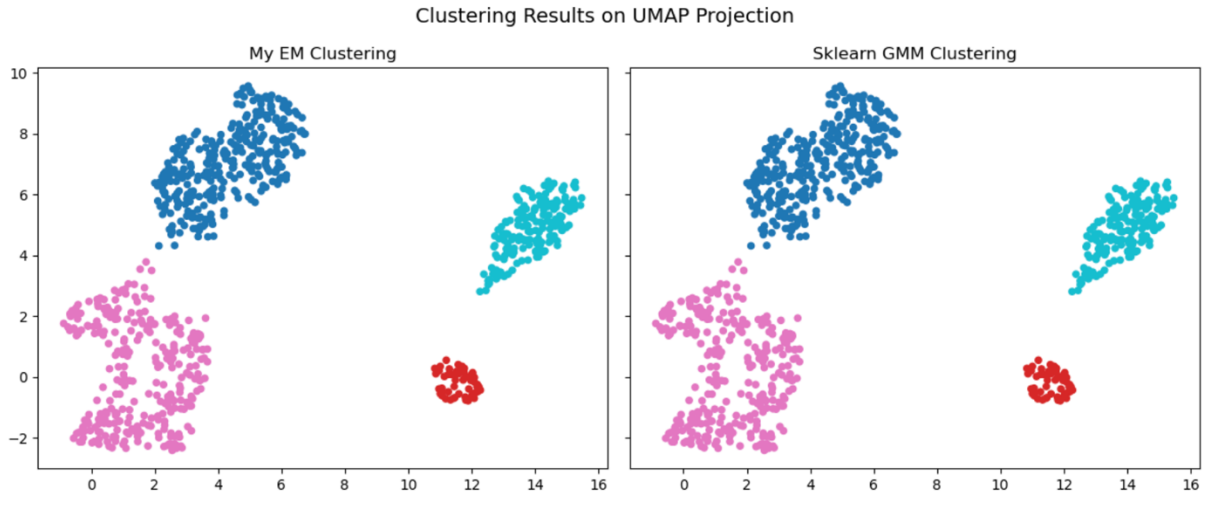
Update mixing weights: $\pi_k = \frac{N_k}{N}$

Running this on the authors algorithm on the 00_authors.csv dataset with the following code (fit_slow uses for loops and then I vectorized it using numpy for speed into fit_fast). See code here (alternatively I have attached a screenshot below).

```python
class GaussianMixtureEM:
    def __init__(self, K, num_iterations, allow_singular=True):
        self.K = K
        self.num_iterations = num_iterations
        self.allow_singular = allow_singular

    def fit_slow(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # E-step
            for i in range(n_rows):
                xi = X.iloc[i].values
                denom = 0
                for k in range(self.K):
                    numerator = pis[k] * multivariate_normal.pdf(xi, mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                    gamma[i, k] = numerator
                    denom += numerator
                gamma[i, :] /= denom

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]
            means = [np.sum([gamma[i, k] * X_array[i, :] for i in range(n_rows)],axis=0) / Nk[k] for k in range(self.K)]
            cov = [np.sum([gamma[i, k] * np.outer(X_array[i, :] - means[k], X_array[i, :] - means[k]) for i in range(n_rows)],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]
            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma
        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}

    def fit_fast(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # Vectorized E-step
            log_pdf_matrix = np.zeros((n_rows, self.K))
            for k in range(self.K):
                rv = multivariate_normal(mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                log_pdf_matrix[:, k] = np.log(pis[k] + 1e-12) + rv.logpdf(X_array)

            max_log = np.max(log_pdf_matrix, axis=1, keepdims=True)
            log_gamma = log_pdf_matrix - max_log
            gamma = np.exp(log_gamma)
            gamma /= gamma.sum(axis=1, keepdims=True)

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]

            means = [np.sum(gamma[:, k][:, np.newaxis] * X_array,axis=0) / Nk[k]for k in range(self.K)]

            cov = [np.sum(gamma[:, k][:, np.newaxis, np.newaxis] *(X_array - means[k])[:, :, np.newaxis] @ (X_array - means[k])[:, :,np.newaxis],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]

            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma

        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}
```

The results of my EM-algorithm against Sklearn EM-algorithm for GMM on the authors data set after applying UMAP yielded the same results!

```
Adjusted Rand Index: 1.0000
```



Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!

01_linear_dimension_reductionV2

April 17, 2025

Download the necessary libraries.

Link to data - https://raw.githubusercontent.com/DataSlingers/clustRviz/master/data/authors.rda

Install these if necessary.

[48]:
```
# %pip install scikit-learn --quiet
# %pip install adjustText --quiet
# %pip install umap-learn --quiet
# %pip install wordcloud
```

[49]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.decomposition import FastICA, NMF, KernelPCA, PCA, TruncatedSVD
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE
from sklearn.model_selection import train_test_split
from sklearn.manifold import MDS

## Msc
from adjustText import adjust_text
from itertools import combinations
from wordcloud import WordCloud
```

Load dataset.

[50]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values
df
```

[50]:
```
           Author   a  all  also  an  and  any  are  as  at  …  was  were  \
0          Austen  46   12     0   3   66    9    4  16  13  …   40    11
1          Austen  35   10     0   7   44    4    3  18  16  …   27    13
2          Austen  46    2     0   3   40    1   13  11   9  …   24     6
3          Austen  40    7     0   4   64    3    3  20  13  …   26    10
4          Austen  29    5     0   6   52    5   14  17   6  …   23     5
..            …  ..  …   …  ..  …   …   …  ..  ..  …  …   …
836  Shakespeare  32    4     0   6   33    0    7   8   4  …    0     1
837  Shakespeare  16    5     0   5   49    1    6  10   3  …    1     1
838  Shakespeare  22   15     0   3   48    0    9  10   2  …    4     0
839  Shakespeare  25    4     0   8   59    3    6   7   3  …    3     4
840  Shakespeare  26    4     0   2   62    0    4   7   4  …    5     0

     what  when  which  who  will  with  would  your
0       7     5      6    8     4     9      1     0
1       5     7      7    3     5    14      8     0
2      10     4      6    4     5    15      3     9
3       3     6     10    5     3    22      4     3
4       8     4     13    2     4    21     10     0
..    …   …     …  …   …   …     …   …
836    13     2      3    3    11    17      5    10
837     6     5      6    0    11    20      2     7
838    16     2      2    0    12    15      1    10
839    11     2      2    2    22    23      4     5
840    13     2      5    3    11    19      0     3

[841 rows x 70 columns]
```

[51]:
```
book_id = df['Author']
book_id.value_counts() # 4 different books ; w/ 317 - Austen, ..., 55 - Milton.
```

[51]:
```
Author
Austen         317
London         296
Shakespeare    173
Milton          55
Name: count, dtype: int64
```

- Unsupervised learning: drop columns [‘Authors’] and determine patterns with words across chapters using unsupervised learning methods.
- We will later come back to these labels we dropped to validate our results.
- Note, a row represents a book chapter with each column representing the word counts of key words in that chapter.

1 Linear Methods

1.1 PCA

PCA Theory

Given a centered data matrix $X \in \mathbb{R}^{n \times p}$ with: - n: observations (chapters), - p: features (words),

PCA uses the SVD: $X = U\Sigma V^\top$

- $U\Sigma \rightarrow$ principal component scores (embedding of chapters),
- V (or `pca.components_`) $\rightarrow$ principal axes (directions for words).

No need to transpose the data to get word embeddings.
Using `fit_transform(X)` for chapters, and `components_.T` for words — where both come from the same PCA fit.

Also note that when fitting a PCA in this dataset: **WE SHOULD NOT SCALE DATA SINCE IT IS WORD COUNT!!!**

1.1.1 Observations

[52]:
```
pca = PCA()
X_pca = pca.fit_transform(X)
word_loadings = pca.components_.T

pca_df = pd.DataFrame(X_pca)
cols = [f'PC{j+1}' for j in range(pca_df.shape[1])]
pca_df = pca_df.rename(columns = {i:cols[i] for i in range(pca_df.shape[1])})
pca_df
```

[52]:

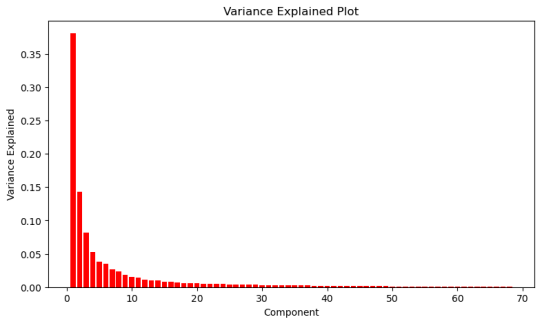
	PC1	PC2	PC3	PC4	PC5	PC6	\
0	-2.265044	43.499301	5.196950	-2.333575	23.359407	22.309224	
1	-2.604648	25.086417	-9.488717	7.748273	19.244916	1.052493	
2	-33.199533	8.667765	-14.833418	3.971572	-6.913595	4.173856	
3	8.098653	21.760546	6.962558	6.683136	15.262020	7.640936	
4	10.031814	6.801164	0.035520	22.731646	10.465248	-5.248171	
..	…	…	…	…	…	…	
836	-64.400961	-28.132705	-21.267908	0.131106	-0.337083	12.068621	
837	-58.313001	-23.417273	4.887866	2.462693	-1.275957	-2.603434	
838	-47.898865	-31.938566	-7.364020	-9.133520	10.442357	14.851150	
839	-39.844905	-29.936659	0.614003	-2.261258	5.007836	14.369278	
840	-34.807687	-38.141056	6.289341	-2.114649	1.435612	13.845645	

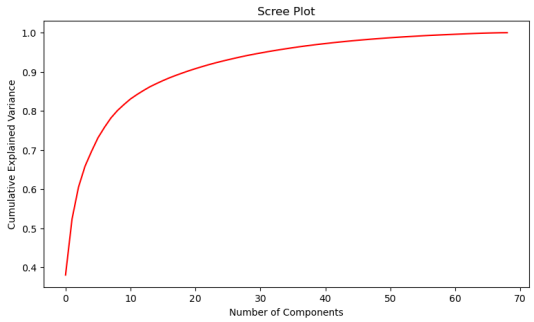
	PC7	PC8	PC9	PC10	…	PC60	PC61	\
0	13.585777	-0.891477	-2.178907	0.062718	…	-2.043904	-3.175850	
1	6.765152	3.650037	-0.746511	-1.753312	…	-2.269775	0.991903	
2	7.094662	-11.728066	1.002140	-4.270366	…	-2.055041	0.864406	
3	11.974431	-5.748289	-1.011280	-4.502115	…	0.590874	0.830834	
4	7.489147	-6.171808	-9.885234	-14.616535	…	1.616164	2.706626	
..	…	…	…	…	…	…	…	

```
836  -5.848328   3.920184 -7.233151   0.239586  … -0.089364 -0.674634
837 -11.452006   4.745220  7.397146  -1.948260  … -0.264725 -1.237238
838 -10.872174  10.228351 -5.141056   3.876664  …  0.938558  1.771321
839  -3.165040   3.620264  1.479972   2.361646  … -0.011889  1.010582
840   3.620534   3.312984 -2.660270   0.521845  …  1.273284  4.486729

         PC62      PC63      PC64      PC65      PC66      PC67      PC68  \
0    1.578725 -2.575981  1.463553 -1.107755 -0.341962  2.639943  3.214014
1    0.013898  1.037391 -1.077602  0.044514 -1.312650  0.800142  0.183083
2   -2.179206  2.564626 -3.672920  0.798756 -0.664101 -1.319190 -0.493611
3   -1.969229  0.937902 -3.603491 -1.244561  1.283435 -0.912942 -0.271062
4   -0.708168 -1.272716 -0.278721 -1.525001  1.746847 -1.004067 -1.272581
..        …         …         …         …         …         …         …
836 -1.165301 -0.470178 -1.348415 -0.240363  0.167233  0.514576 -0.123753
837 -0.170168  1.335993  0.455795  0.942986 -0.266152 -2.256417 -0.182997
838  1.542836 -1.795202  1.242780  2.648138 -0.017259 -2.002290  0.787364
839  1.514307 -0.011713  1.135751  0.803847 -0.028491 -0.803976 -1.307802
840  3.058903  0.497595  0.257034 -0.099739 -0.201420 -0.168980 -0.437193

         PC69
0   -0.594109
1   -0.194564
2    0.165242
3   -0.743841
4   -0.229321
..        …
836 -0.036760
837 -0.021365
838  0.071339
839 -0.026705
840 -0.209892

[841 rows x 69 columns]
```

[53]:
```
fig, ax = plt.subplots(1,2, figsize=(20,5))

ax[0].bar(np.arange(1, pca_df.shape[1]),pca.explained_variance_ratio_[0:pca_df.
 ↪shape[1]-1], color = 'red')
ax[0].set_xlabel('Component')
ax[0].set_ylabel('Variance Explained')
ax[0].set_title('Variance Explained Plot')

ax[1].plot(np.cumsum(pca.explained_variance_ratio_), color = 'red')
ax[1].set_xlabel('Number of Components')
ax[1].set_ylabel('Cumulative Explained Variance')
ax[1].set_title('Scree Plot')
```

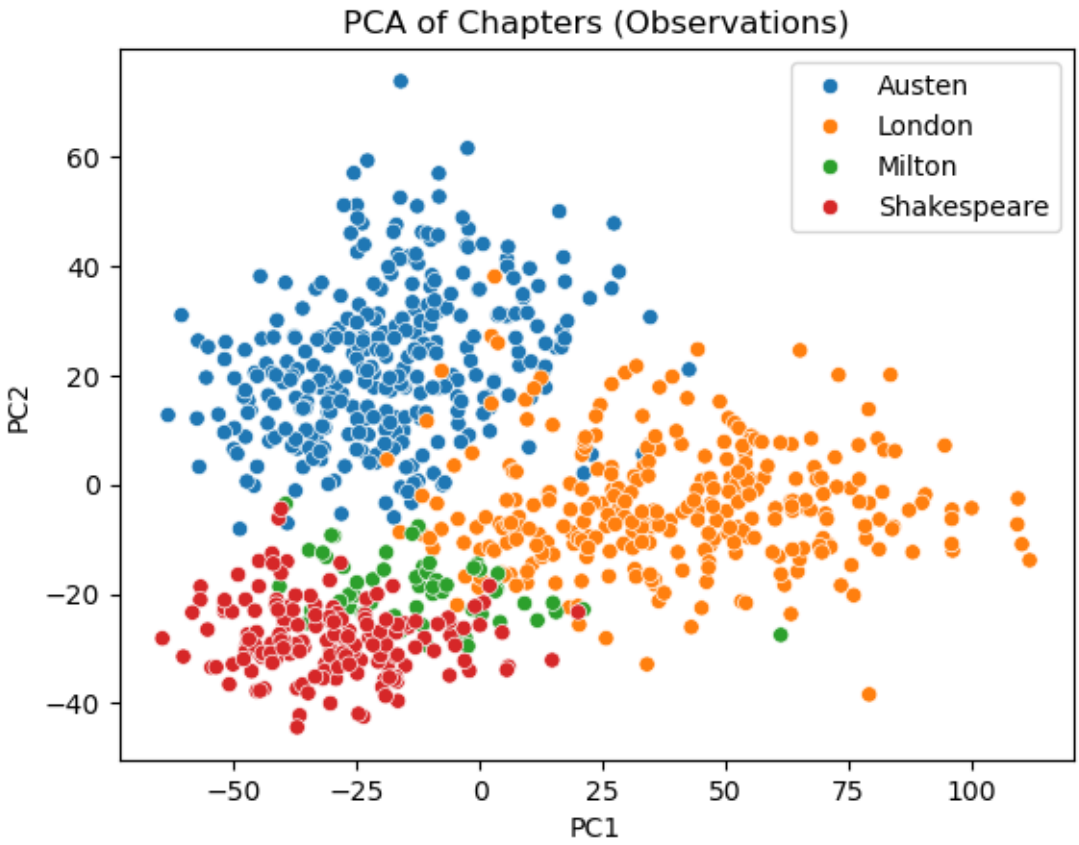
```
plt.savefig('Media/viz/01/01_pca_var_explained_and_screeplot')
plt.show()

print(f'PC1 and PC2 explain {np.sum(pca.explained_variance_ratio_[0:2])*100:.
  ↪3f}% of the variance.')
```






```
PC1 and PC2 explain 52.396% of the variance.
```

[54]:
```
pca_chapters_df = pca_df.loc[:,['PC1','PC2']] # Nested and ordered; extract␣
  ↪first 2 Principal Components
pca_chapters_df['Author'] = df['Author']

sns.scatterplot(data=pca_chapters_df, x='PC1', y='PC2', hue='Author',␣
  ↪palette='tab10')
plt.title('PCA of Chapters (Observations)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.savefig('Media/viz/01/01_pca_chapters')
plt.show()
```



1.1.2 Features

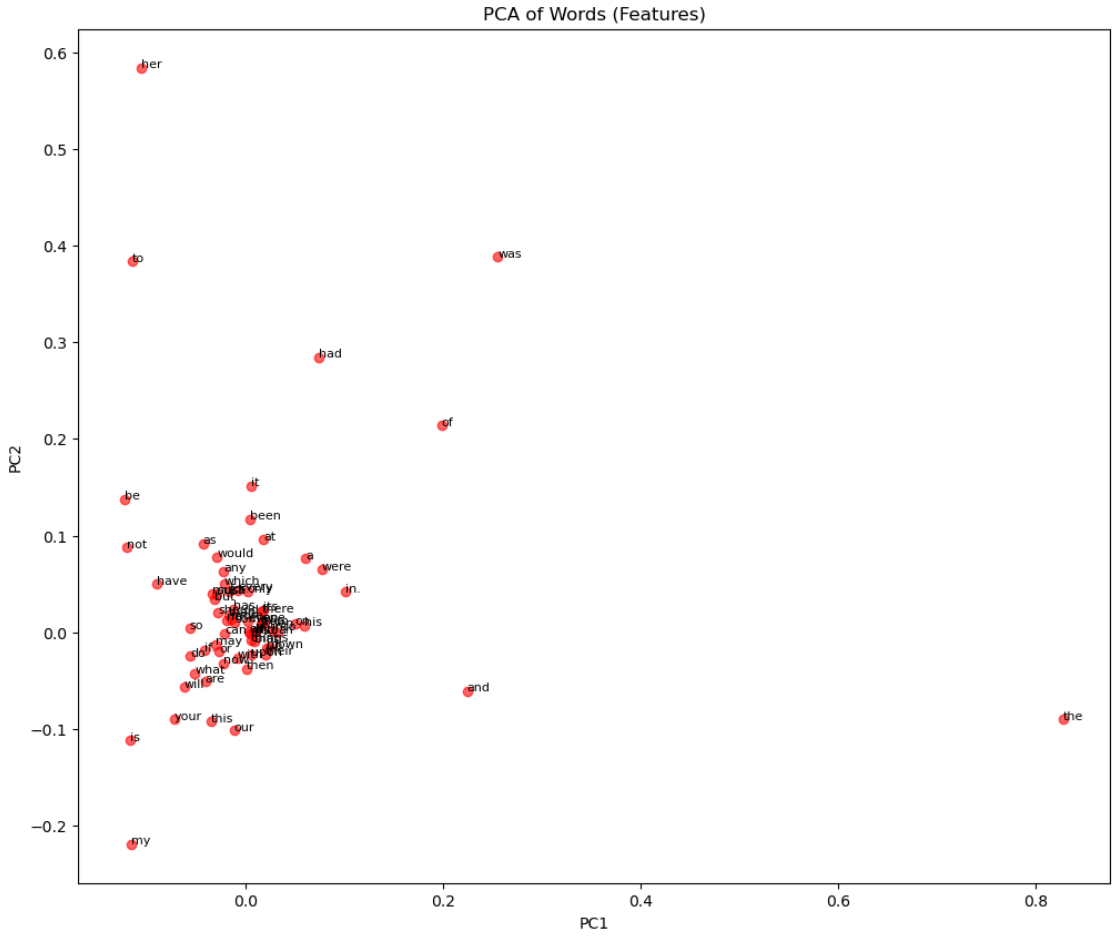
[55]:
```
pca_words_df = pd.DataFrame(word_loadings)
cols = [f'PC{j+1}' for j in range(pca_words_df.shape[1])]
pca_words_df = pca_words_df.rename(columns = {i:cols[i] for i in␣
 ↪range(pca_words_df.shape[1])})
pca_words_df.index = X.columns
pca_words_df
```

[55]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	\
a	0.060676	0.076522	-0.315737	-0.055225	-0.078926	0.263739	0.484181	
all	0.003003	0.000487	0.053015	-0.032618	0.066769	-0.029175	-0.009633	
also	0.007791	-0.000515	0.007859	0.001355	-0.008722	0.000714	0.000660	
an	0.032094	-0.000843	-0.326638	-0.223061	0.188812	-0.163141	0.137205	
and	0.223970	-0.060692	0.667811	-0.081825	-0.015979	0.543313	-0.035569	
…	…	…	…	…	…	…	…	
who	0.003630	-0.001030	0.052789	0.002280	0.002378	-0.013840	-0.007167	
will	-0.062244	-0.056680	-0.001381	0.124291	0.004152	0.024512	-0.127001	
with	-0.008282	-0.026201	0.128355	-0.102281	0.099969	-0.044243	0.033596	
would	-0.029127	0.078012	-0.025317	0.047541	-0.050614	0.034905	-0.062669	

```
your  -0.072387 -0.090066 -0.053952  0.107079 -0.015571  0.009243 -0.028564

             PC8       PC9      PC10  …      PC60      PC61      PC62  \
a      -0.525683  0.264041  0.219213  … -0.005026  0.002127 -0.020590
all     0.042661  0.058567 -0.200501  …  0.007117 -0.043006  0.021078
also    0.007541  0.003267 -0.006968  … -0.002700  0.018310 -0.007202
an      0.060527  0.235075 -0.152501  … -0.012955 -0.026053 -0.005756
and    -0.145092  0.219987 -0.003553  … -0.002065  0.003563  0.004656
…          …        …        …  …        …         …         …
who    -0.006857  0.004959 -0.064097  … -0.030093 -0.166349  0.078814
will   -0.070483 -0.042921  0.024465  …  0.029968  0.013908  0.020761
with   -0.015710  0.041990  0.047793  … -0.006375 -0.009951  0.001977
would  -0.037895 -0.032398 -0.054165  … -0.024179 -0.008643 -0.011250
your    0.019719  0.015591  0.159339  …  0.010928 -0.015083 -0.011472

            PC63      PC64      PC65      PC66      PC67      PC68      PC69
a      -0.020622 -0.005487 -0.001468  0.006522 -0.002964  0.020486  0.001386
all     0.017012 -0.001458  0.008575  0.012527  0.033792 -0.014598  0.001157
also   -0.008381 -0.001825  0.006400 -0.019838 -0.093516  0.056643  0.989992
an      0.020056  0.008385 -0.027075  0.000523  0.005918  0.007497  0.005771
and    -0.002794  0.001242 -0.004503  0.005302 -0.000100  0.000272 -0.005046
…          …         …         …         …         …         …         …
who     0.125654 -0.021114 -0.080313  0.096907  0.061265 -0.000161 -0.017311
will    0.013775  0.006232 -0.059988 -0.017531 -0.003613  0.022391 -0.003819
with   -0.007391  0.009002  0.006421 -0.005216  0.012884  0.008480 -0.000474
would   0.021813  0.022624 -0.031247  0.009858 -0.018687 -0.006684 -0.001789
your   -0.023488 -0.021720  0.023567  0.006809 -0.026835  0.019442 -0.000493

[69 rows x 69 columns]
```

[56]:
```python
plt.figure(figsize=(12, 10))
plt.scatter(pca_words_df['PC1'], pca_words_df['PC2'], alpha=0.6, color = 'red')

for _, row in pca_words_df.iterrows():
    plt.text(row['PC1'], row['PC2'], row.name, fontsize=8)  # ← fixed here
plt.title('PCA of Words (Features)')
plt.xlabel('PC1')
plt.ylabel('PC2');
plt.savefig('Media/viz/01/01_pca_words')
```



This method offers poor visualization and little interpretability. There are other methods that will provide more interpretability on the features such as NMF.

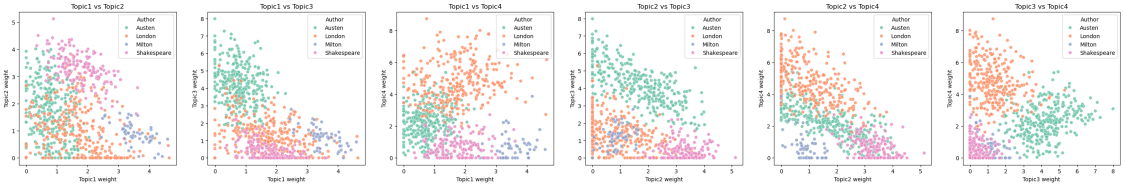
1.2 NMF

NMF may perform strongly here because our word frequencies are non-negative! Furthermore, this method will provide semantic intuition by creating topics.

[57]:
```
k = 4 # hyperparameter -> clusters
model = NMF(n_components=k, init='random', random_state=0, max_iter=500)

# dimensions -> (841 chapters, k topics)
W = model.fit_transform(X)

# dimensions -> (k topics, 69 words)
H = model.components_
```

1.2.1 Observations

[58]:
```
W_df = pd.DataFrame(W, index=X.index, columns=[f'Topic{i+1}' for i in␣
↪range(k)]) # Turn W into a DataFrame with word labels
topic_df = W_df.reset_index(drop=True)
topic_df["Author"] = df["Author"].reset_index(drop=True)
topics = W_df.columns.tolist() # List of topic names
topic_pairs = list(combinations(topics, 2)) # All pairs of topics

n_plots = len(topic_pairs)
fig, axes = plt.subplots(nrows=1, ncols=n_plots, figsize=(6 * n_plots, 5))

if n_plots == 1:
    axes = [axes]

for i, (x_topic, y_topic) in enumerate(topic_pairs):
    ax = axes[i]
    sns.scatterplot(data=topic_df, x=x_topic, y=y_topic, hue="Author",␣
↪palette="Set2", alpha=0.8, ax=ax)
    ax.set_title(f"{x_topic} vs {y_topic}")
    ax.set_xlabel(f"{x_topic} weight")
    ax.set_ylabel(f"{y_topic} weight")
    ax.legend().set_title("Author")

plt.savefig('Media/viz/01/01_nmf_observations')
```



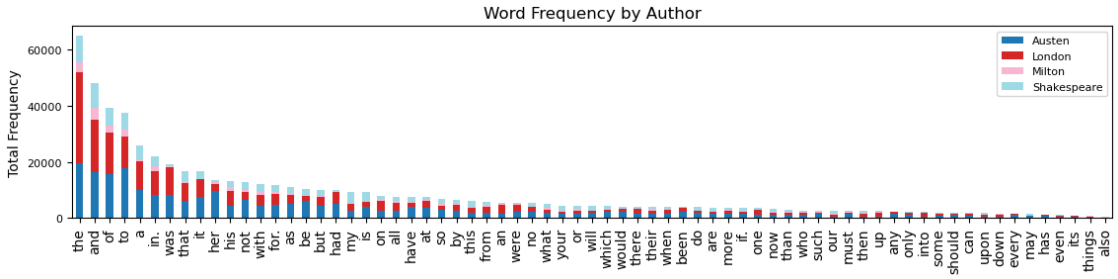
This provides poor interpretability so it is clearly not our first choice when visualizing chapters/observations.

1.2.2 Features

First of all the most simple and easy/inuitive vizualization of the features/words is just a simple total count!

[59]:
```
X_df = X.copy()
X_df['Author'] = authors
author_word_freq = X_df.groupby('Author').sum().T
author_word_freq = author_word_freq.loc[author_word_freq.sum(axis=1).
↪sort_values(ascending=False).index]
```

```
fig, ax = plt.subplots(figsize=(len(author_word_freq)/6, 3))  # adjust based on␣
 ↪# of words
author_word_freq.plot(
    kind='bar',
    stacked=True,
    ax=ax,
    colormap='tab20'
)

plt.title("Word Frequency by Author", fontsize=12)
plt.ylabel("Total Frequency", fontsize=10)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=8)
plt.legend(loc='upper right', fontsize=8)
plt.tight_layout()
plt.savefig("Media/viz/01/01_all_word_freq_by_author_stackedbar_condensed.png",␣
 ↪dpi=300)
plt.show()
```



This is where NMF will truly shine.

[60]:
```
n = 10 # Set here -> top words per topic!
```

[61]:
```
word_topic_df = pd.DataFrame(H.T, index=X.columns, columns=[f"Topic{i+1}" for i␣
 ↪in range(H.shape[0])]) # Transpose H to get words as rows, topics as columns
columns = list(word_topic_df.columns)
highest_n_weighted_words_per_topic = {}
for col in columns:
    words = list(word_topic_df[col].sort_values(ascending=False).head(n).index)
    highest_n_weighted_words_per_topic[col] = words
print(highest_n_weighted_words_per_topic)
```

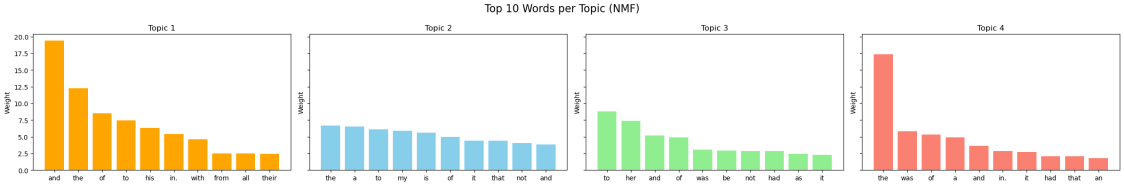
```
{'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all',
'their'], 'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not',
'and'], 'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as',
'it'], 'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that',
'an']}
```

Below are semantic interpretations of each topic based on the top contributing words:

- **Topic 1 – Structural & Possessive Language**
  Emphasizes grammatical connectors, possession, and sentence scaffolding — likely reflecting narrative structure or formal exposition.
- **Topic 2 – Determiners & Negation**
  Centers on articles, pronouns, and simple negations, suggesting basic sentence formation and assertive language.
- **Topic 3 – Past-Tense Narration**
  Highlights auxiliary verbs and past-tense forms, indicating descriptive or event-driven storytelling.
- **Topic 4 – Temporal & Descriptive Grammar**
  Focuses on narrative tense, articles, and function words often used in unfolding sequences or descriptive prose.

These topics reflect broad grammatical and stylistic features common in text, helping differentiate author styles or narrative structures.

[62]:
```
fig, axes = plt.subplots(1, H.shape[0], figsize=(6 * H.shape[0], 4),␣
 ↪sharey=True)

colors = ['orange', 'skyblue', 'lightgreen', 'salmon']  # or however many␣
 ↪topics you have
for topic_idx, ax in enumerate(axes):
    topic_weights = H[topic_idx]
    top_word_indices = topic_weights.argsort()[::-1][:n]
    top_words = [X.columns[i] for i in top_word_indices]
    top_scores = topic_weights[top_word_indices]
    ax.bar(top_words, top_scores, color=colors[topic_idx % len(colors)])
    ax.set_title(f"Topic {topic_idx + 1}")
    ax.set_ylabel("Weight")

fig.suptitle(f"Top {n} Words per Topic (NMF)", fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/01/01_nmf_top_words_per_topic')
```



Extra Feature Vizuals (not scientific)

[63]:
```
word_freq_dict = X.sum(axis=0).to_dict()

wordcloud = WordCloud(width=1000, height=500, background_color='white').
 ↪generate_from_frequencies(word_freq_dict)

plt.figure(figsize=(15, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of All Features", fontsize=16)
plt.savefig('Media/viz/01/01_word_cloud')
plt.show()
```

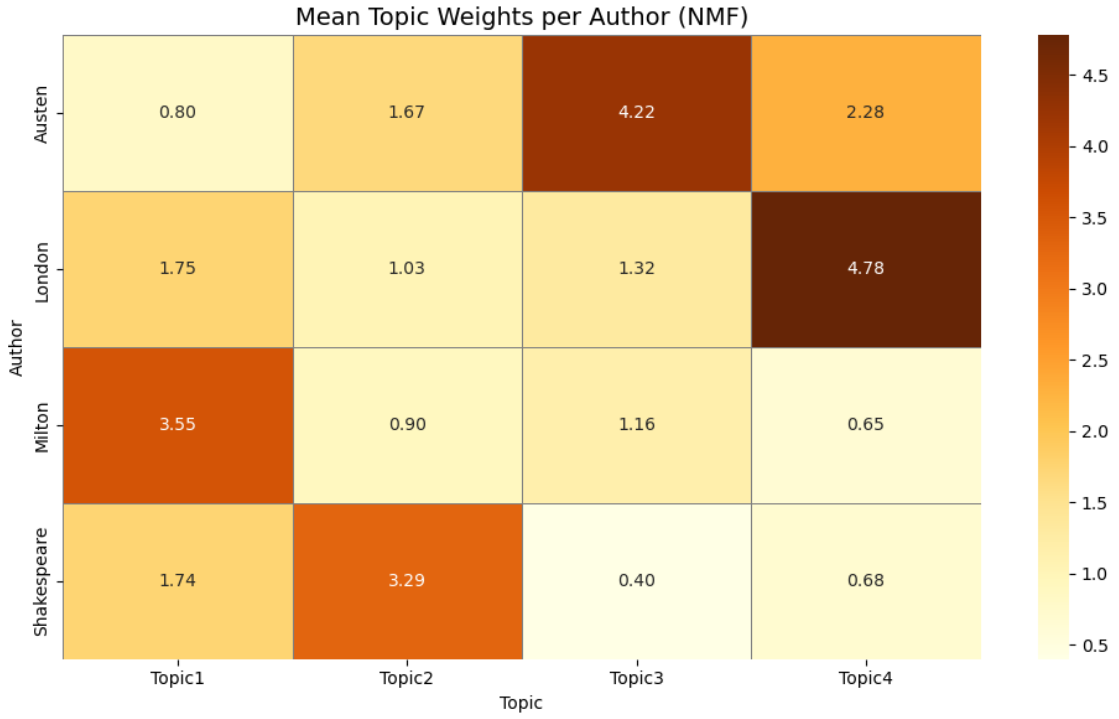
Now let us try and use this to determine which chapters correspond to which topics and predict the author based on the semantics of each topic.

Establish a mapping by grouping by using the labels, grouping by Author and then evaluating the mean topic weights per author! First we should split our data into a training and testing.

[64]:
```
W_df['Authors'] = df['Author']

# Split the data
train_df, test_df = train_test_split(W_df, test_size=0.2, random_state=42,␣
 ↪stratify=W_df['Authors'])

print(f"Training samples: {len(train_df)}")
print(f"Testing samples: {len(test_df)}")
```

```
Training samples: 672
Testing samples: 169
```

[65]:
```
author_topic_means = train_df.groupby('Authors').mean() # Compute mean topic␣
 ↪weights grouped by author
plt.figure(figsize=(10, 6))
sns.heatmap(author_topic_means, annot=True, fmt=".2f", cmap="YlOrBr",␣
 ↪linewidths=0.5, linecolor='gray')
plt.title("Mean Topic Weights per Author (NMF)", fontsize=14)
plt.xlabel("Topic")
plt.ylabel("Author")
plt.tight_layout()
plt.savefig("Media/viz/01/01_nmf_author_topic_heatmap", dpi=300)
```



Mean Topic Weights per Author (NMF)

The heatmap above shows the average topic weights across all chapters written by each author. It is computed by grouping the NMF document-topic matrix (`W`) by author and taking the mean.

Interpretation: - Each cell reflects how strongly a given author tends to express a specific topic. - Higher values indicate that the author frequently uses patterns or structures associated with that topic. - While no topic is exclusive to a single author, we observe distinct preferences: - **Milton** leans heavily on Topic 1 - **Shakespeare** shows strong usage of Topic 2 - **Austen** favors Topic 3 - **London** stands out on Topic 4

This unsupervised representation helps reveal stylistic or grammatical tendencies across authors.

[66]:
```
mapping = {'Topic3': 'Austen', 'Topic4': 'London', 'Topic1': 'Milton', 'Topic2':
 ↪ 'Shakespeare'}
```

Now lets evaluate this mapping on the test set!

[67]:
```
test_df['Highest Weighted Topic'] = test_df.iloc[:, :4].idxmax(axis=1).values
test_df['Likely Author'] = test_df['Highest Weighted Topic'].apply(lambda x:␣
 ↪mapping[x])
correct_proportion = (test_df['Likely Author'] == test_df['Authors']).mean()
print(f'This method was correct {correct_proportion*100:.4f}% of the time')
```

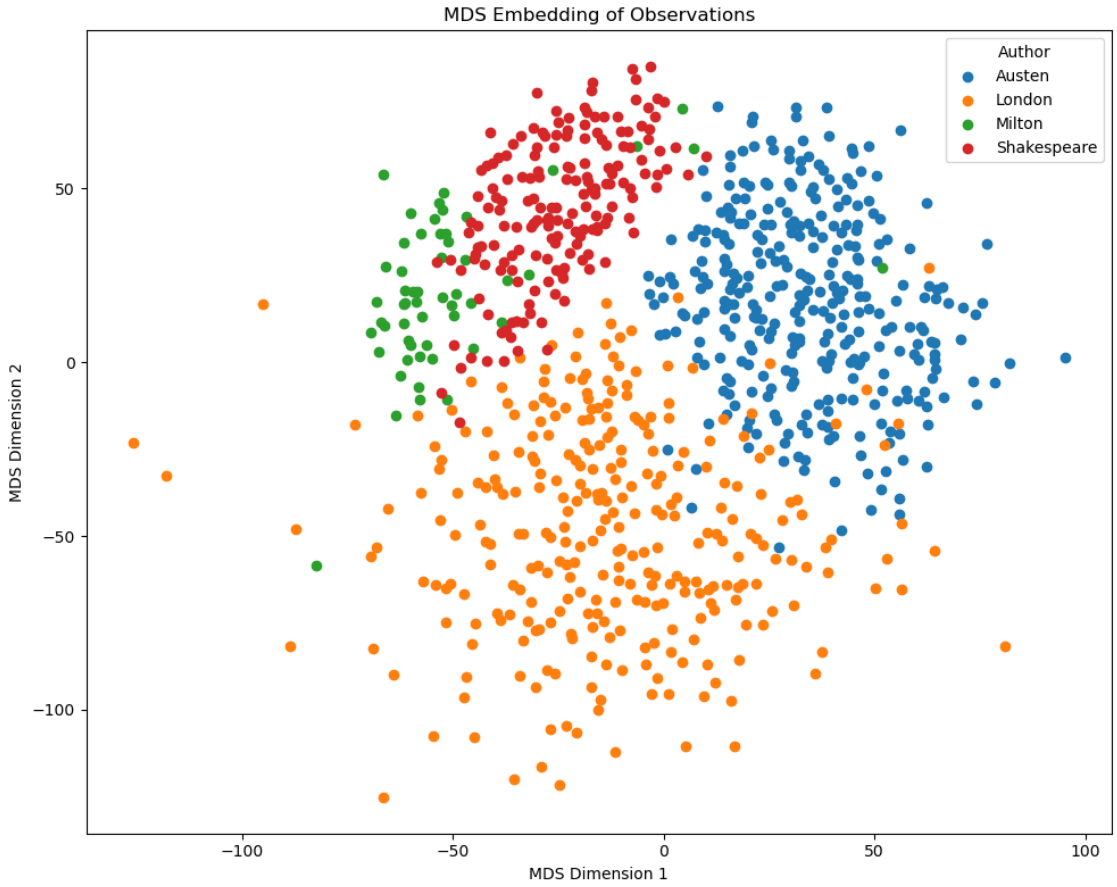
```
This method was correct 91.7160% of the time
```

This yields pretty good results!!

1.3 MDS

1.3.1 Observations

[68]:
```
model = MDS(n_components=2, random_state=42)
X_mds = model.fit_transform(X)

unique_authors = sorted(set(authors))

# Plot
plt.figure(figsize=(10, 8))
for i, author in enumerate(unique_authors):
    mask = authors == author
    plt.scatter(X_mds[mask, 0], X_mds[mask, 1], label=author)

plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.title("MDS Embedding of Observations")
plt.legend(title="Author", loc="best")
plt.tight_layout()
plt.savefig("Media/viz/01/01_mds_observation_viz")
plt.show()
```



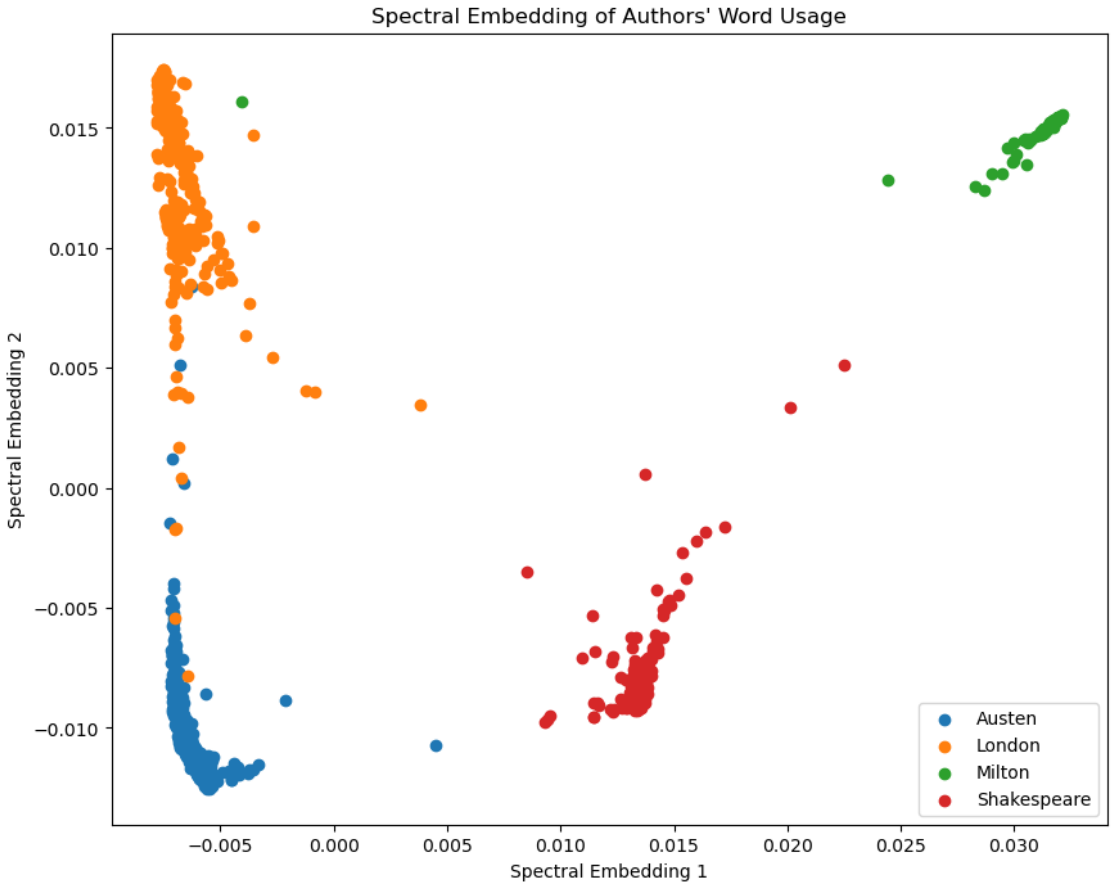
2 Non-Linear Methods

2.1 Spectral Embeddings

2.1.1 Observations

[69]:
```
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10)
X_spec = spectral.fit_transform(X)  # shape: (n_samples, 2)
spectral_df = pd.DataFrame(X_spec, columns=['Dim1', 'Dim2'])

plt.figure(figsize=(10, 8))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
plt.xlabel("Spectral Embedding 1")
plt.ylabel("Spectral Embedding 2")
plt.title("Spectral Embedding of Authors' Word Usage")
plt.legend(loc="lower right")
```

```
plt.savefig('Media/viz/01/01_spectal_obs_viz')
plt.show()
```



2.1.2 Features

Unlike PCA and NMF there are no components we can use to visualize the features in Spectral Embedding. However, we can apply a transpose to our data and then reapply Spectral Embedding.

This procedure will yield results where:

- Two words will be close in the embedding if they tend to appear in the same chapters.
- This gives you a co-occurrence-like structure, driven by chapter usage.
- This is conceptually similar to Latent Semantic Analysis, just via graph-based distance instead of SVD.

[70]:
```
X_transpose = X.T
X_transpose = X_transpose.rename(columns = {i:f'Chapter{i}' for i in range(df.
  ↪shape[0])})
X_transpose
```

[70]:

	Chapter0	Chapter1	Chapter2	Chapter3	Chapter4	Chapter5	Chapter6	\
a	46	35	46	40	29	27	34	
all	12	10	2	7	5	8	8	
also	0	0	0	0	0	0	0	
an	3	7	3	4	6	3	15	
and	66	44	40	64	52	42	44	
…	…	…	…	…	…	…	…	
who	8	3	4	5	2	6	4	
will	4	5	5	3	4	3	9	
with	9	14	15	22	21	18	11	
would	1	8	3	4	10	4	6	
your	0	0	9	3	0	5	4	

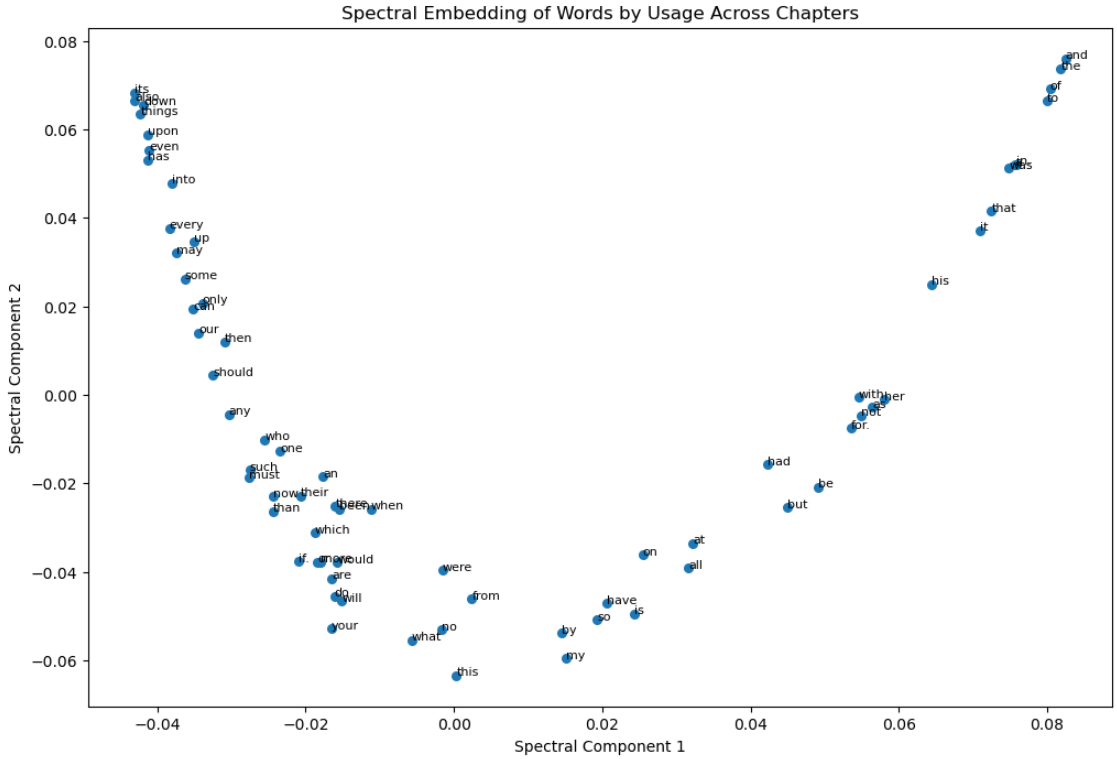
	Chapter7	Chapter8	Chapter9	…	Chapter831	Chapter832	Chapter833	\
a	38	34	54	…	46	48	39	
all	6	12	8	…	4	2	5	
also	1	0	0	…	0	0	0	
an	2	5	6	…	3	9	10	
and	67	50	44	…	43	45	38	
…	…	…	…	…	…	…	…	
who	6	1	3	…	1	0	2	
will	7	2	5	…	7	10	8	
with	15	13	15	…	18	11	26	
would	3	12	6	…	2	6	2	
your	5	5	2	…	3	11	16	

	Chapter834	Chapter835	Chapter836	Chapter837	Chapter838	Chapter839	\
a	22	28	32	16	22	25	
all	13	7	4	5	15	4	
also	0	0	0	0	0	0	
an	5	7	6	5	3	8	
and	47	45	33	49	48	59	
…	…	…	…	…	…	…	
who	4	2	3	0	0	2	
will	9	7	11	11	12	22	
with	12	8	17	20	15	23	
would	6	3	5	2	1	4	
your	7	7	10	7	10	5	

	Chapter840
a	26
all	4
also	0
an	2
and	62
…	…
who	3

```
will           11
with           19
would           0
your            3

[69 rows x 841 columns]
```

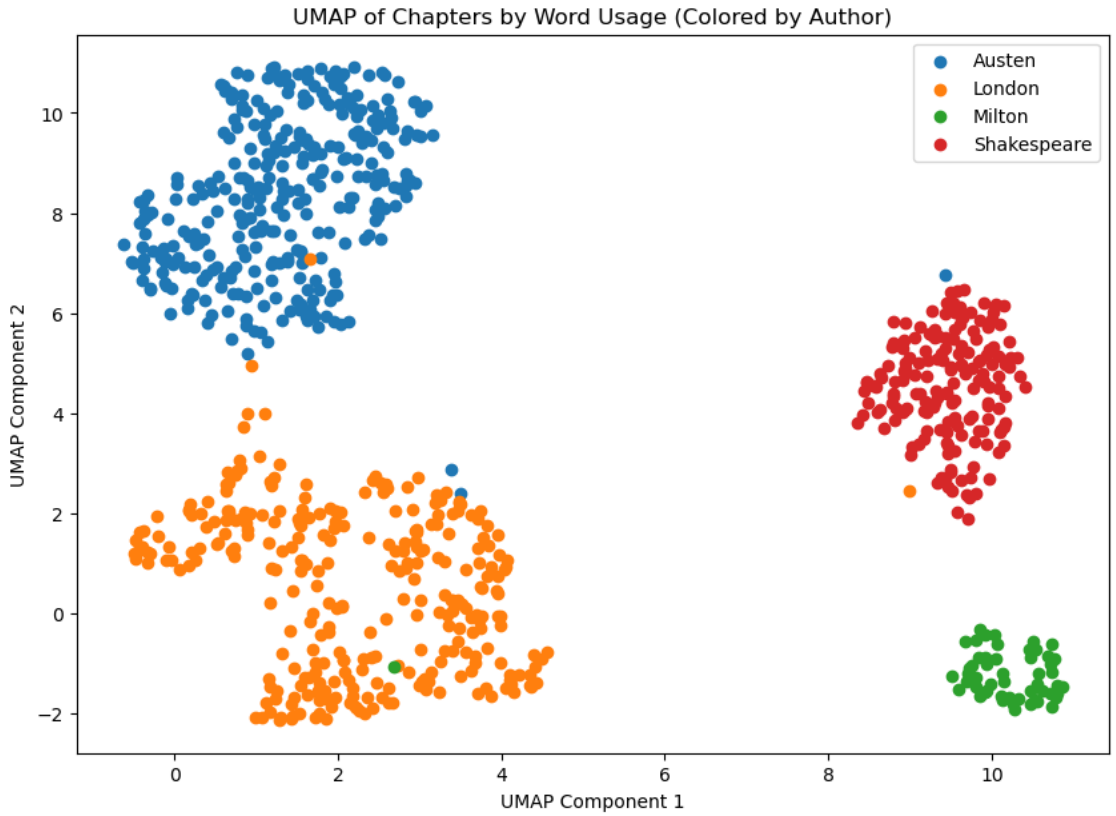
[71]:
```
X_words = X_transpose.to_numpy()
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=0)
X_words_spec = spectral.fit_transform(X_words)

plt.figure(figsize=(12, 8))
plt.scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
plt.title("Spectral Embedding of Words by Usage Across Chapters")
plt.xlabel("Spectral Component 1")
plt.ylabel("Spectral Component 2")
plt.savefig('Media/viz/01/01_spectral_feature_viz')
plt.show()
```



2.2 UMAP

2.2.1 Observations

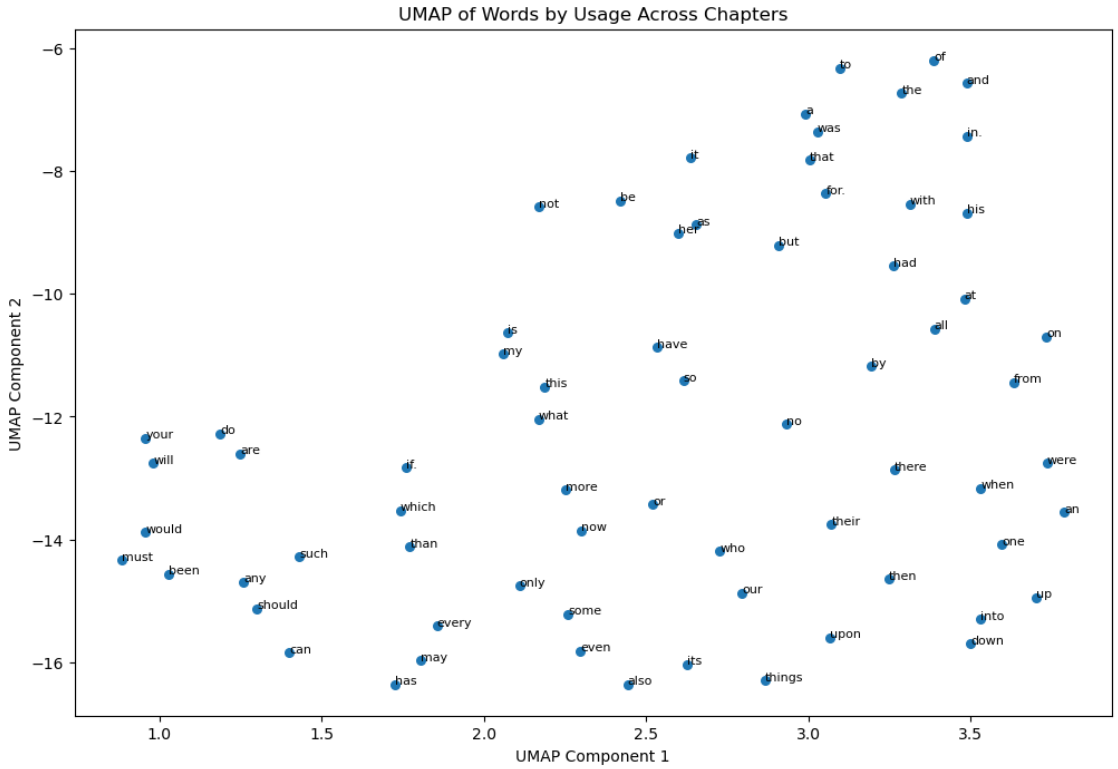
[84]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_umap = umap_model.fit_transform(X.to_numpy())

plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_umap_obs_viz')
plt.show()

plt.figure(figsize=(10, 7))
plt.scatter(X_umap[:,0],X_umap[:,1], color = 'purple')
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP")
plt.savefig('Media/viz/01/01_umap_viz_unlabeled');
plt.close()
```



UMAP does a very good job at creating seperable clusters! This visualization appears to be the best so far when observing observations (chapters)!

2.2.2 Features

[73]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_words_umap = umap_model.fit_transform(X_transpose.to_numpy())

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
plt.title("UMAP of Words by Usage Across Chapters")
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.savefig('Media/viz/01/01_umap_feature_viz')
plt.show()
```

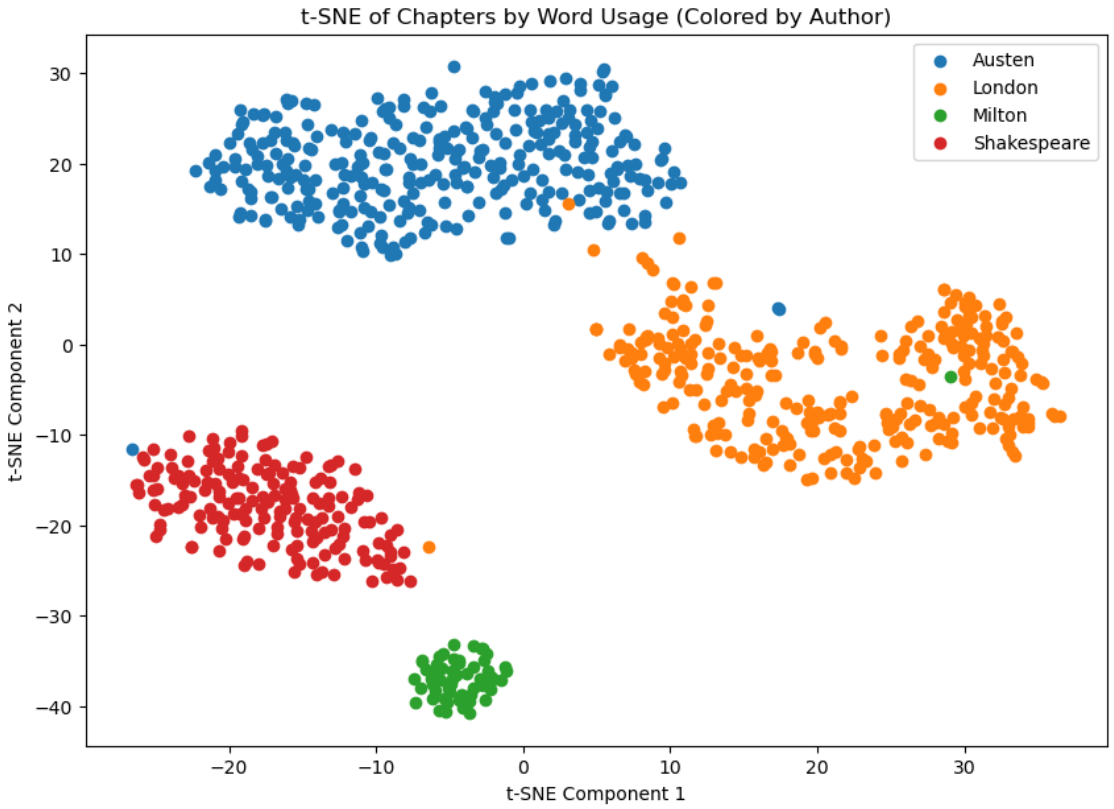


Not very interpretable/helpful.

2.3 tSNE

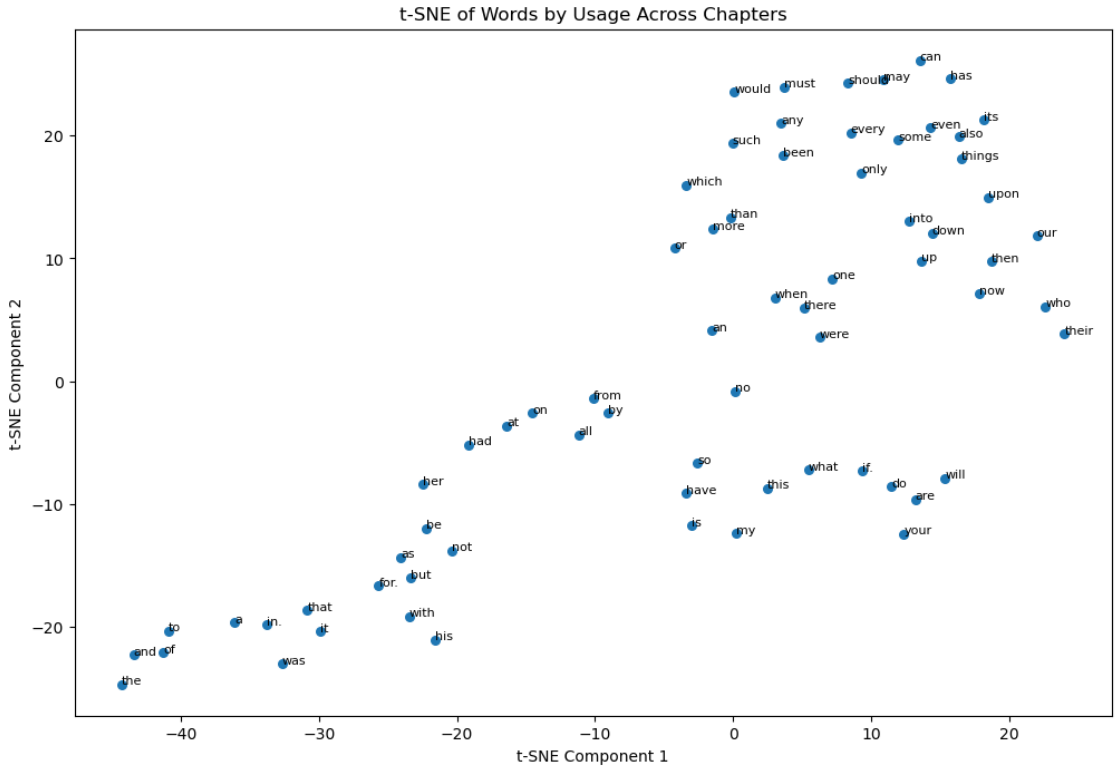
2.3.1 Observations

[74]:
```
tsne = TSNE(n_components=2, perplexity=30, learning_rate='auto') # Fit t-SNE
X_tsne = tsne.fit_transform(X.to_numpy())

# Plot chapters with author labels
plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)

plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.title("t-SNE of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_tsne_obs_viz')
plt.show()
```



2.3.2 Features

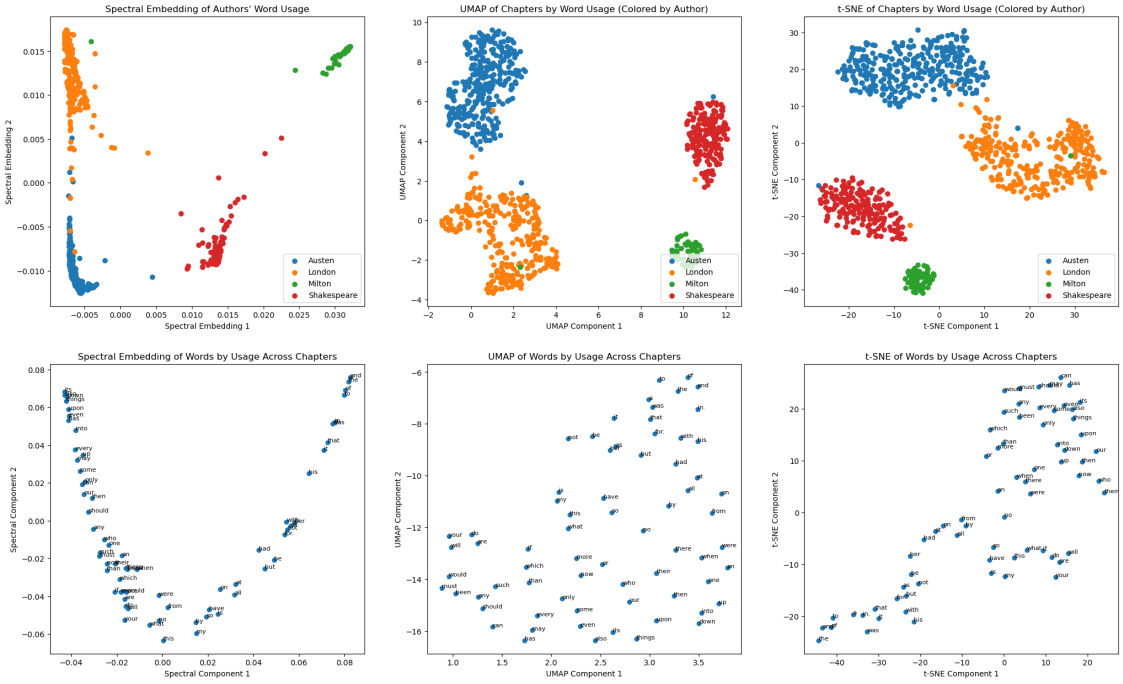
[75]:
```
tsne = TSNE(n_components=2, perplexity=5, learning_rate='auto') # Fit t-SNE
X_words_tsne = tsne.fit_transform(X_words)

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)

plt.title("t-SNE of Words by Usage Across Chapters")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.savefig('Media/viz/01/01_tsne_feature_viz')
plt.show()
```



3 Combined Visualizations

[76]:
```
fig, ax = plt.subplots(2, 3, figsize=(25, 15), sharey=False)  # 1 row, 3 columns

# Spectral Embedding
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,0].scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
ax[0,0].set_xlabel("Spectral Embedding 1")
ax[0,0].set_ylabel("Spectral Embedding 2")
ax[0,0].set_title("Spectral Embedding of Authors' Word Usage")
ax[0,0].legend(loc="lower right")

ax[1,0].scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,0].text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
ax[1,0].set_title("Spectral Embedding of Words by Usage Across Chapters")
ax[1,0].set_xlabel("Spectral Component 1")
ax[1,0].set_ylabel("Spectral Component 2")

# UMAP
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
```

```
    mask = (authors == author)
    ax[0,1].scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
ax[0,1].set_xlabel("UMAP Component 1")
ax[0,1].set_ylabel("UMAP Component 2")
ax[0,1].set_title("UMAP of Chapters by Word Usage (Colored by Author)")
ax[0,1].legend(loc="lower right")

ax[1,1].scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,1].text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
ax[1,1].set_title("UMAP of Words by Usage Across Chapters")
ax[1,1].set_xlabel("UMAP Component 1")
ax[1,1].set_ylabel("UMAP Component 2")

# tSNE
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,2].scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)
ax[0,2].set_xlabel("t-SNE Component 1")
ax[0,2].set_ylabel("t-SNE Component 2")
ax[0,2].set_title("t-SNE of Chapters by Word Usage (Colored by Author)")
ax[0,2].legend(loc="lower right")

ax[1,2].scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,2].text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)
ax[1,2].set_title("t-SNE of Words by Usage Across Chapters")
ax[1,2].set_xlabel("t-SNE Component 1")
ax[1,2].set_ylabel("t-SNE Component 2")

plt.savefig('Media/viz/01/01_nonlinear_viz')
plt.show()
```



UMAP offers the best balance between global and local structure, preserves neighborhood quality, and gives interpretable groupings without as much distortion as t-SNE.

Clusters of words in the UMAP embedding reflect similar usage patterns across chapters. Since word usage is shaped by topic and syntax, these local neighborhoods can be interpreted as reflecting semantic or grammatical similarity. While UMAP does not preserve global distances, local groupings are meaningful.

02_visualizations

April 17, 2025

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE, MDS
from sklearn.decomposition import PCA, NMF, FastICA, TruncatedSVD
from sklearn.metrics import pairwise_distances

## Msc
from adjustText import adjust_text
from itertools import combinations
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df["Author"].values

X_words = X.T.values
feature_names = X.columns
```

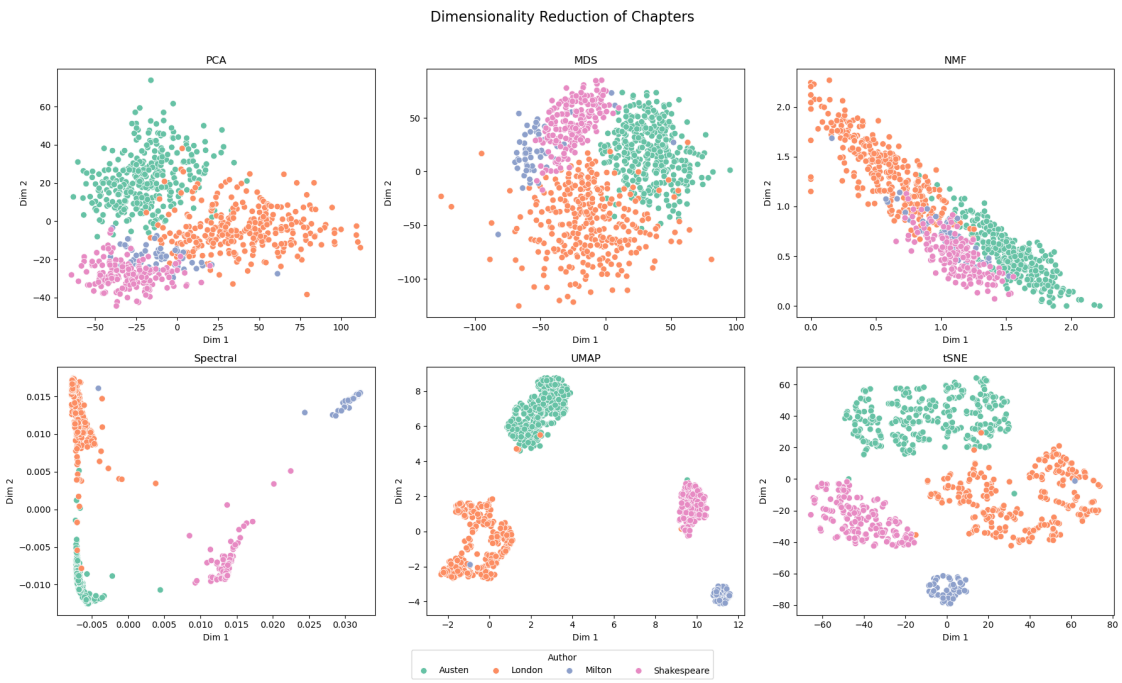
1 Comparison

1.1 Observations

[9]:
```
methods = {
    "PCA": PCA(n_components=2),
    "MDS": MDS(n_components=2, random_state=42),
```

```
    "NMF": NMF(n_components=2, random_state=42, max_iter=1000),
    "Spectral": SpectralEmbedding(n_components=2, affinity="nearest_neighbors",␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

palette = { "Austen": "#66c2a5", "London": "#fc8d62", "Milton": "#8da0cb",␣
 ↪"Shakespeare": "#e78ac3"} # Set consistent colors for each label

embeddings = {name: model.fit_transform(X) for name, model in methods.items()}

fig, axs = plt.subplots(2, int(len(embeddings)/2), figsize=(len(methods)*3, 10))

for ax, (name, embed) in zip(axs.flatten(), embeddings.items()):
    sns.scatterplot(x=embed[:, 0], y=embed[:, 1], hue=authors, palette=palette,␣
 ↪s=50, ax=ax)
    ax.set_title(name)
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")
    ax.legend().remove()

handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, title="Author", loc='lower center', ncol=4,␣
 ↪bbox_to_anchor=(0.5, -0.05))
fig.suptitle("Dimensionality Reduction of Chapters", fontsize=16, y=1.02)

plt.tight_layout()
plt.savefig("Media/viz/02/02_across_methods_obs_viz.png", bbox_inches="tight",␣
 ↪dpi=300)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```

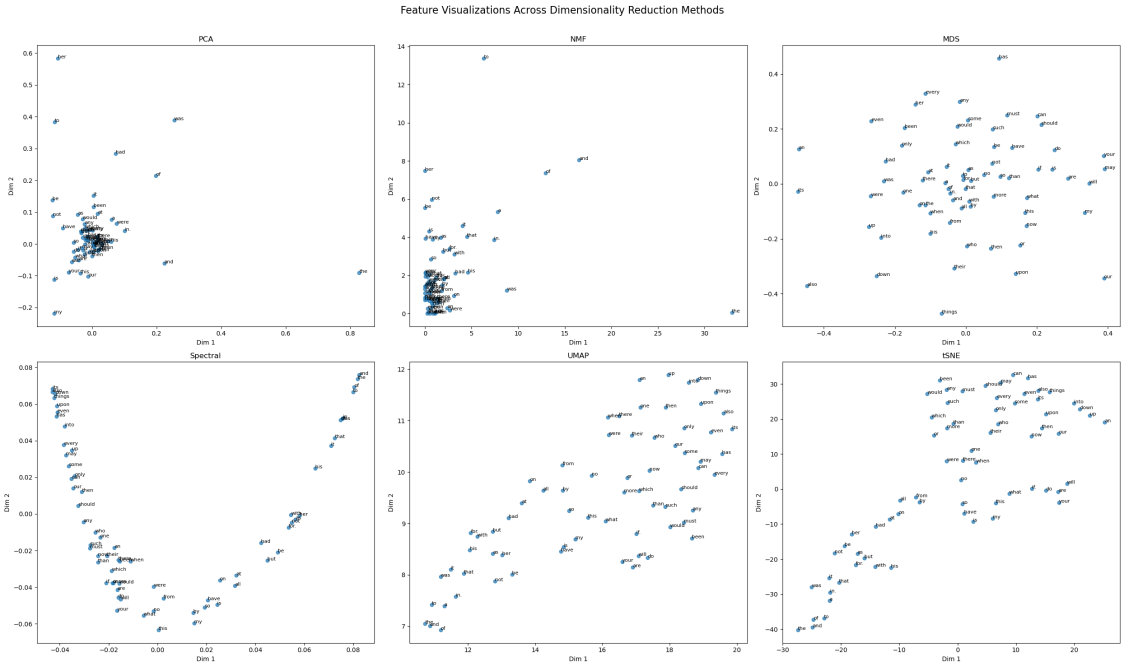


1.2 Features

[5]:
```
methods = {
    "PCA": PCA(n_components=2).fit(X),
    "NMF": NMF(n_components=2, init='random', random_state=42, max_iter=1000).
 ↪fit(X),
    "MDS": MDS(n_components=2, dissimilarity='precomputed', random_state=42),
    "Spectral": SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, n_neighbors=10, min_dist=0.3,␣
 ↪random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

feature_embeddings = {
    "PCA": methods["PCA"].components_.T,
    "NMF": methods["NMF"].components_.T,
    "MDS": methods["MDS"].fit_transform(pairwise_distances(X_words,␣
 ↪metric="cosine")),
    "Spectral": methods["Spectral"].fit_transform(X_words),
    "UMAP": methods["UMAP"].fit_transform(X_words),
    "tSNE": methods["tSNE"].fit_transform(X_words)
}
```

```
fig, axs = plt.subplots(2, 3, figsize=(25, 15))
axs = axs.flatten()

for ax, (name, coords) in zip(axs, feature_embeddings.items()):
    ax.scatter(coords[:, 0], coords[:, 1], s=30, alpha=0.7)
    for i, word in enumerate(feature_names):
        ax.text(coords[i, 0], coords[i, 1], word, fontsize=8)
    ax.set_title(f"{name}")
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")

plt.suptitle("Feature Visualizations Across Dimensionality Reduction Methods",␣
 ↪fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.savefig("Media/viz/02/02_feature_comparison_all_methods")
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```



03_biclustering_viz

April 17, 2025

Install these if necessary.

[1]:
```
# %pip install fastcluster --quiet
```

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
from sklearn.cluster import SpectralBiclustering, SpectralCoclustering
from sklearn.feature_selection import VarianceThreshold

## Msc
from PIL import Image
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values  # n_samples-length array
```

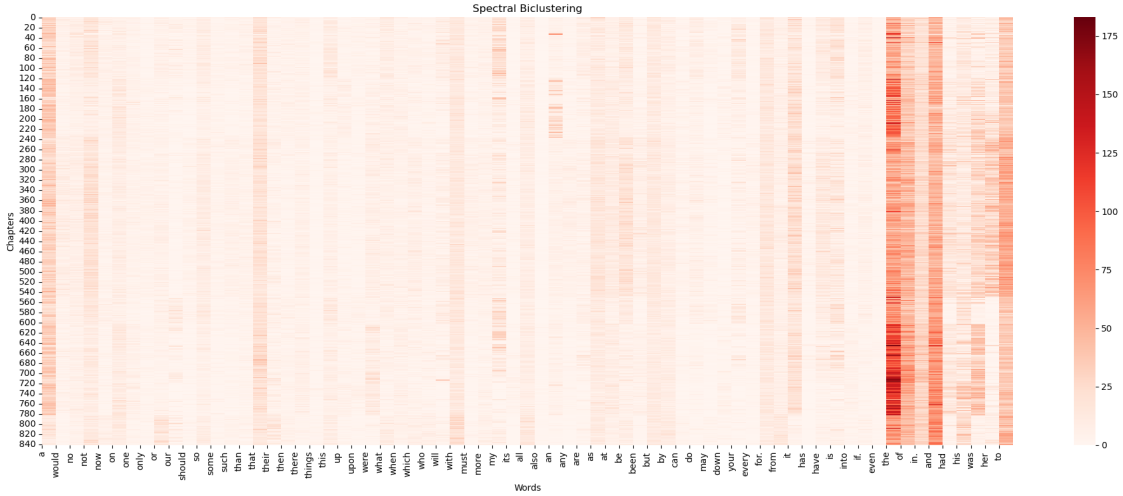
1 Biclustering

1.1 Spectral Biclustering

Let us fit and visualize both the chapters and words using Spectral Biclustering!

[4]:
```
### Fit biclustering model
model = SpectralBiclustering(n_clusters=4, method='log', random_state=0)
model.fit(X)

### Reorder the data
row_order = np.argsort(model.row_labels_)
```

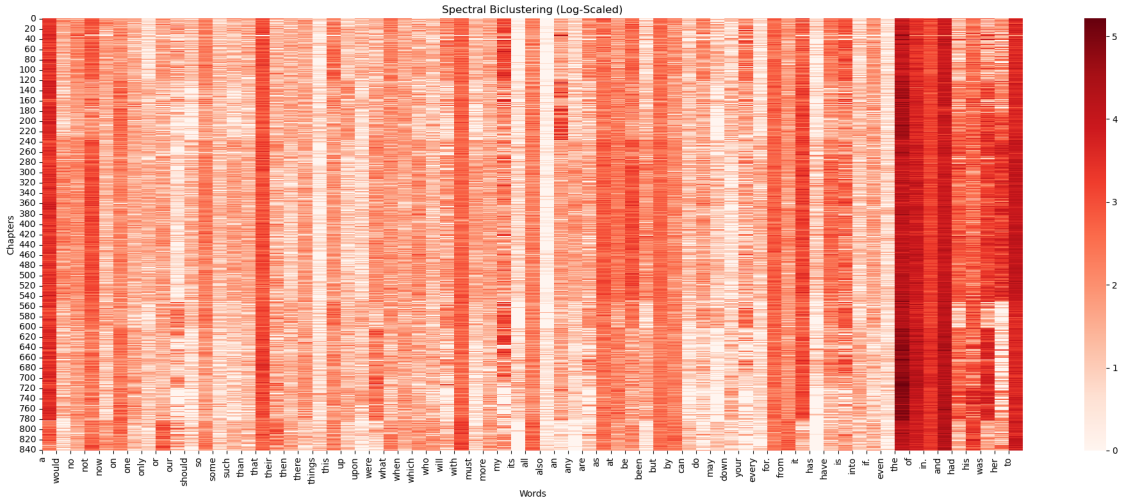
```
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Visualization
plt.figure(figsize=(20, 8))  # wider figurea
sns.heatmap(fit_data, cmap="Reds", cbar=True)
plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/spectral_biclustering_viz')
plt.show()
```



Since words like “the”, “and”, “to”, … are used frequently across chapters on a global level there is not much insights to be deduced here. We want to look for at words with specific chapters having bands. This will help us to differentiate a word that is frequently used across a cluster of chapters indicating stylistic writing by the author! This will help with classifying these chapters to their respective authors!! There is a small problem that the common words are very dark so it is harder to see a contrast in scale so we will convert the scale to log scale to see a larger contrast among the less frequently used words.

1.1.1 Log-Scale (Improve Visualization)

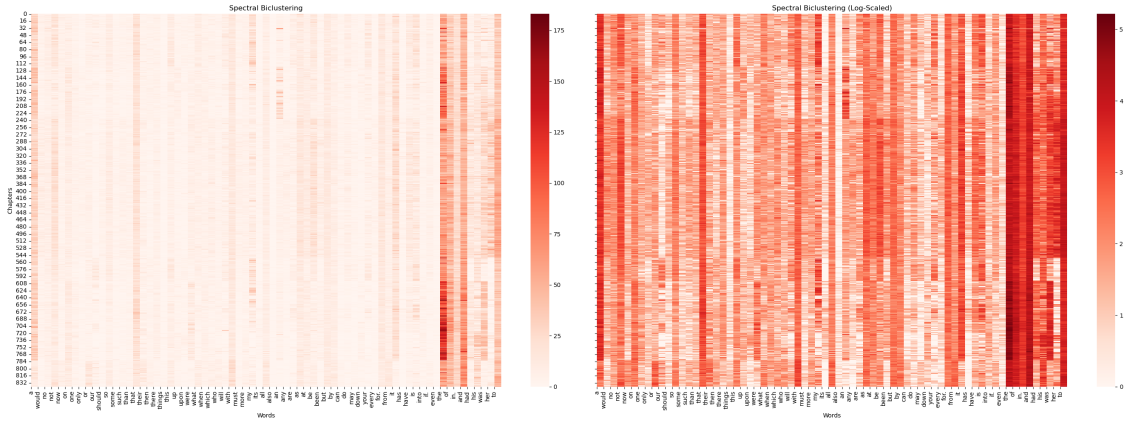
[5]:
```
### Reorder the data
row_order = np.argsort(model.row_labels_)
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Apply log scale to compress high-frequency words like "the"
log_data = np.log1p(fit_data)  # log(1 + x) to avoid log(0)

### Plot the heatmap
plt.figure(figsize=(20, 8))
sns.heatmap(log_data, cmap="Reds", cbar=True)

plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering (Log-Scaled)")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/log_scale_spectral_biclustering_viz')
plt.show()
```



1.1.2 Final Visualization

Plot side by side for visualization/comparison purposes.

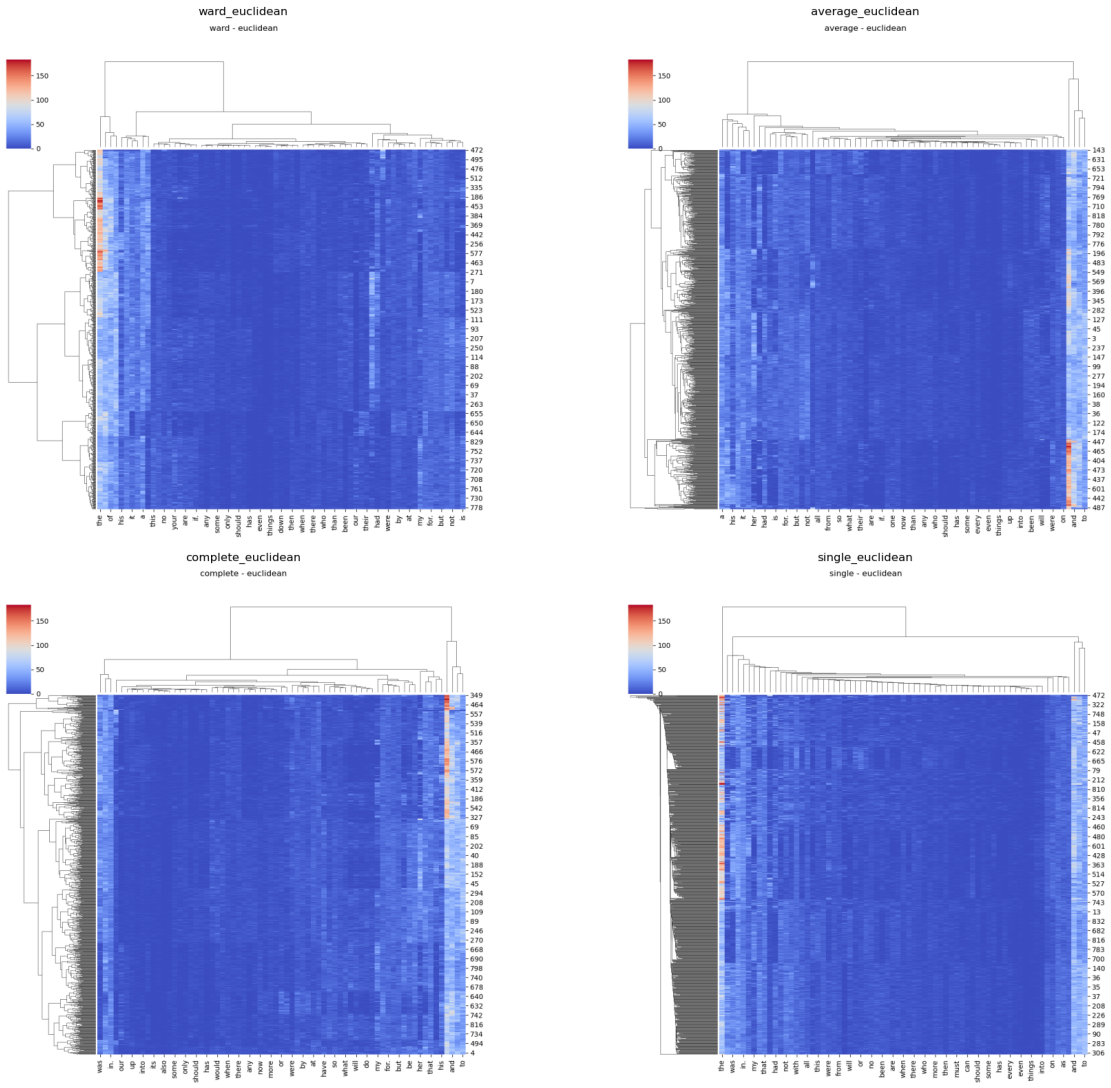
[6]:
```
# Plot side-by-side heatmaps
fig, axes = plt.subplots(1, 2, figsize=(28, 10), sharey=True)

# Original
sns.heatmap(fit_data, cmap="Reds", ax=axes[0], cbar=True)
axes[0].set_xticks(np.arange(len(word_labels)))
axes[0].set_xticklabels(word_labels, rotation=90)
axes[0].set_title("Spectral Biclustering")
axes[0].set_xlabel("Words")
axes[0].set_ylabel("Chapters")

# Log-scaled
sns.heatmap(log_data, cmap="Reds", ax=axes[1], cbar=True)
axes[1].set_xticks(np.arange(len(word_labels)))
axes[1].set_xticklabels(word_labels, rotation=90)
axes[1].set_title("Spectral Biclustering (Log-Scaled)")
axes[1].set_xlabel("Words")
axes[1].set_ylabel("")

plt.tight_layout()
plt.savefig('Media/viz/03/03_spectral_biclustering_heatmaps')
plt.show()
```



1.2 Hierarchial Biclustering

[7]:
```
# Define top 4 method-metric combinations (all Euclidean)
combinations = [
    ('ward', 'euclidean'),
    ('average', 'euclidean'),
    ('complete', 'euclidean'),
    ('single', 'euclidean'),
]
```

[8]:
```
img_paths = []
for method, metric in combinations:
    g = sns.clustermap(X, method=method, metric=metric, cmap='coolwarm')
    g.fig.suptitle(f'{method} - {metric}', y=1.05)
    img_path = f'Media/viz/03/single_plots/clustermap_{method}_{metric}.png'
    g.savefig(img_path, bbox_inches='tight')
    plt.close(g.fig)
    img_paths.append(img_path)

fig, axes = plt.subplots(2, 2, figsize=(20, 16))
for ax, path in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(path.split('/')[-1].replace('clustermap_', '').replace('.png',␣
 ↪''))
    ax.axis('off')

plt.tight_layout()
plt.savefig('Media/viz/03/03_hierarchial_biclustering_heatmaps')
plt.show()
```



This clustermap visualizes word frequency across chapters using hierarchical clustering on both rows (chapters) and columns (words). Chapters with similar word usage patterns are grouped together, as are words that tend to co-occur across the same sets of chapters. The dendrograms reveal hidden structure in the data, highlighting natural groupings based on stylistic similarity without any labels. This helps identify clusters of similar chapters and potentially stylistically meaningful groups of words.

However, we can see that the interpretation from these heatmaps is tricky so lets try and reduce the features with low variance patterns. So we will do some initial feature filtering, then perform the same heatmap visualizations!

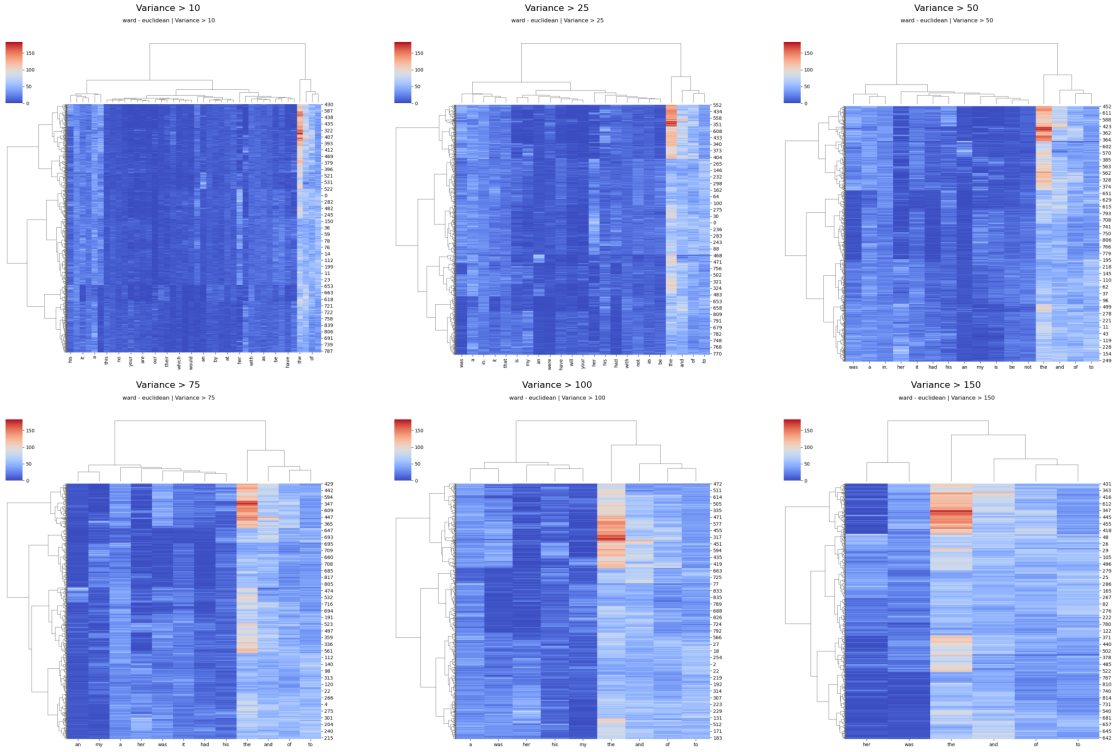
[9]:
```
X_filtered_dict = {'var_threshold':[], 'X_filtered_df':[]}
thresholds_list = [10, 25, 50, 75, 100, 150]
combinations = [('ward', 'euclidean')]
```

```
for alpha in thresholds_list: # Loop over variance filter threshold as alpha␣
 ↪increase, more features are dropped!
    selector = VarianceThreshold(threshold=alpha)
    X_reduced = selector.fit_transform(X)
    selected_columns = X.columns[selector.get_support()] # Get the column names␣
 ↪that passed this variance threshold
    X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)
    X_filtered_dict['var_threshold'].append(alpha)
    X_filtered_dict['X_filtered_df'].append(X_filtered)
```

Plotting a grid across variance thresholds as follows.

[10]:
```
img_paths = []
for alpha in thresholds_list:
    selector = VarianceThreshold(threshold=alpha)
    try:
        X_reduced = selector.fit_transform(X)
        selected_columns = X.columns[selector.get_support()]
        if len(selected_columns) == 0:
            continue
        X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)

        for method, metric in combinations:
            g = sns.clustermap(X_filtered, method=method, metric=metric,␣
 ↪cmap='coolwarm')
            g.fig.suptitle(f'{method} - {metric} | Variance > {alpha}', y=1.05)
            img_path = f'Media/viz/03/single_plots/
 ↪reduced_clustermap_{method}_{metric}_var{alpha}.png'
            g.savefig(img_path, bbox_inches='tight')
            plt.close(g.fig)
            img_paths.append((img_path, alpha))
    except ValueError:
        continue  # Skip thresholds that drop all features

n_cols = 3
n_rows = -(-len(img_paths) // n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(22, 7 * n_rows))

for ax, (path, alpha) in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(f'Variance > {alpha}', fontsize=12)
    ax.axis('off')

plt.tight_layout()
```

```
plt.savefig('Media/viz/03/03_hierarchical_biclustering_variance_grid.png')
plt.show()
```



This feature selection helps to make the patterns between the stop words (features) and the chapters a lot more distinguishable by dropping the sparse features that experience little variance across chapters. From this we can see bands that may indicate a shift in semantics across chapters!

04_clustering_comparison

April 17, 2025

Load necessary libraries.

[1]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap

### Msc
import warnings

### OOP
from ml_utils import ClusterEvaluator
```

Load dataset.

[2]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
 ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
X = X.to_numpy() # change pd.DataFrame to np.ndarray
authors = df['Author'].values  # n_samples-length array
```

We can also assign a hyperparameter $K = 4$. This represents the number of clusters (which we already know; there are 4 authors). Later on we will be hyperparameter tuning for some K based on stability!

[3]:
```
K_ = 4
```

Let us also define a dictionary to store our results.

[4]:
```
accuracy_dict = {}
```

1 Clustering Methods

1.1 Kmeans++

[5]:
```
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300) #␣
 ↪K-means++ initialization
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'London'}
The accuracy of Kmeans++ = 0.9096313912009513
```

1.2 Gaussian Mixture Models

[6]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X)
y_gmm = gmm.predict(X)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9512485136741974
```

1.3 Spectral Clustering

Spectral clustering already had a dimensional reduction by design, i.e., spectral clustering is essentially spectral embedding followed by kmeans! Therefore, we expect this 'raw' method to perform best compared to the other methods without any form of dimensionality reduction!

[7]:
```
spectral = SpectralClustering(n_clusters=K_, affinity='nearest_neighbors')
y_spectral = spectral.fit_predict(X)
```

```
accuracy, mapping = ClusterEvaluator(y_spectral,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['spectral clustering'] = accuracy
```

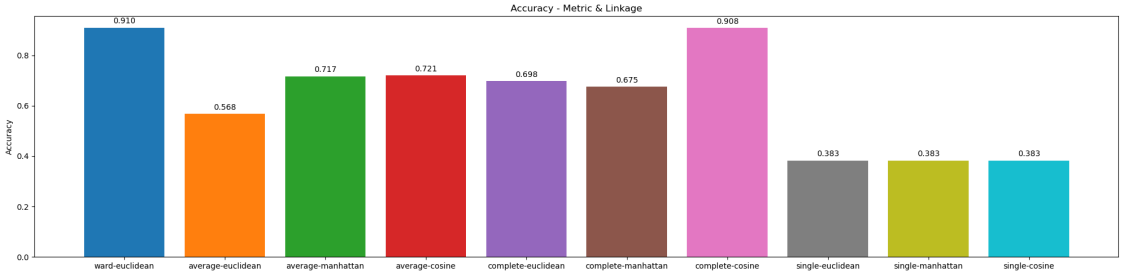
```
The mapping based on mode = {0: 'Shakespeare', 1: 'Austen', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9881093935790726
```

1.4 Hierarchial Clustering

[8]:
```
# Establish all possible combinations for linkage and metric!
methods = []
linkage_methods = ['ward', 'average', 'complete', 'single']
metrics = ['euclidean', 'manhattan', 'cosine']

for linkage in linkage_methods:
    for metric in metrics:
        if linkage == 'ward' and metric != 'euclidean':
            continue  # ward only supports euclidean
        methods.append((linkage, metric))

# Loop through all combinations and determine best accuracy!
hierarchical_accuracy_dict = {}
vals = df['Author'].values
for linkage, metric in methods:
    hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
    y_hierarchical = hierarchical.fit_predict(X)  # Fit and predict in one step
    accuracy, mapping = ClusterEvaluator(y_hierarchical,vals).accuracy # Use␣
 ↪OOP code
    # print(f'The mapping based on mode = {mapping}')
    # print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
    hierarchical_accuracy_dict[f'{linkage}-{metric}'] = accuracy
```

[9]:
```
keys = list(hierarchical_accuracy_dict.keys())
vals = list(hierarchical_accuracy_dict.values())
colors = plt.colormaps.get_cmap('tab10').colors

fig, ax = plt.subplots(figsize=(20, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
```

```
plt.savefig('Media/viz/04/04_hier_accuracy_across_metric_linkage')
plt.show()
```

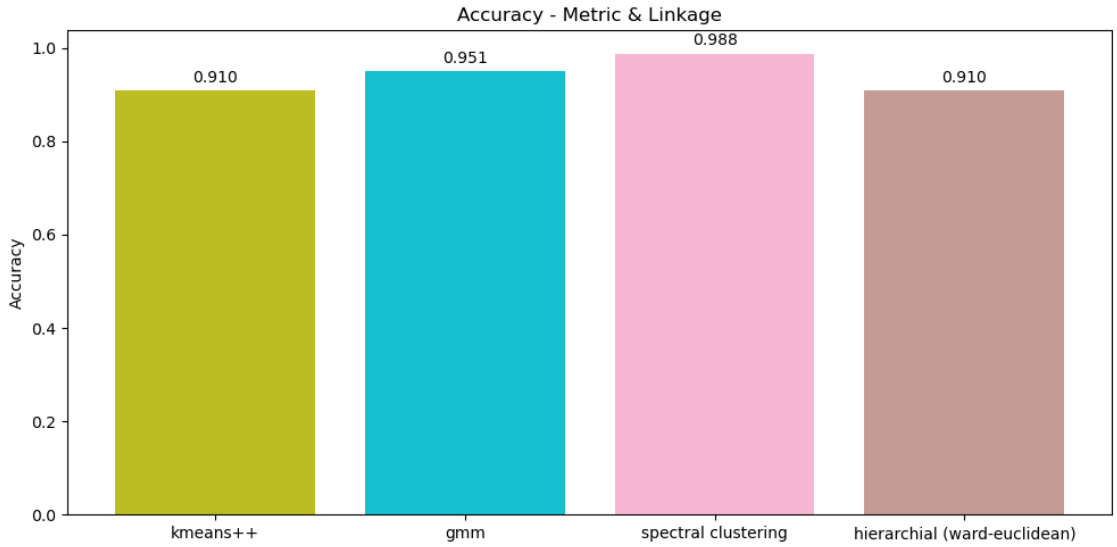


Lets store the highest accuracy params to compare across other methods - ward + euclidean.

[10]:
```
accuracy_dict['hierarchial (ward-euclidean)'] =␣
↪hierarchical_accuracy_dict['ward-euclidean']
```

1.5 Comparison

[11]:
```
keys = list(accuracy_dict.keys())
vals = list(accuracy_dict.values())

colors = ['#bcbd22', '#17becf', '#f7b6d2', '#c49c94']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods')
plt.show()
```



Clearly, spectral clustering performs the best because it has a build in dimensional reduction! Therefore, now let us apply UMAP to all these methods and recompare.

2 Dimensional Reduction + Clustering Methods

Use UMAP as dimensionality reduction for all methods (except spectral clustering as this already uses spectral embedding so we will not repeat this).

[12]:
```
warnings.filterwarnings("ignore", category=UserWarning, module="umap") #␣
↪Suppress the specific UMAP warning on parralelism
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3)
X_umap = umap_model.fit_transform(X)
```

Store results in the following dictionary to compare across methods.

[13]:
```
umap_accuracy_dict = {}
umap_accuracy_dict['spectral clustering'] = accuracy_dict['spectral clustering']
```

2.1 Kmeans++

[14]:
```
# Run K-means++ on UMAP-reduced data
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300)
kmeans.fit(X_umap)
y_kmeans = kmeans.predict(X_umap)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
```

```
umap_accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.2 Gaussian Mixture Models

[15]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X_umap)
y_gmm = gmm.predict(X_umap)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
umap_accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Shakespeare', 1: 'London', 2: 'Austen', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.3 Hierarchial Clustering

[16]:
```
linkage = 'ward'
metric = 'euclidean'
hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
y_hierarchical = hierarchical.fit_predict(X_umap)  # Fit and predict in one step
accuracy, mapping = ClusterEvaluator(y_hierarchical,df['Author'].values).
 ↪accuracy # Use OOP code
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
umap_accuracy_dict[f'hierarchial ({linkage}-{metric})'] = accuracy
```

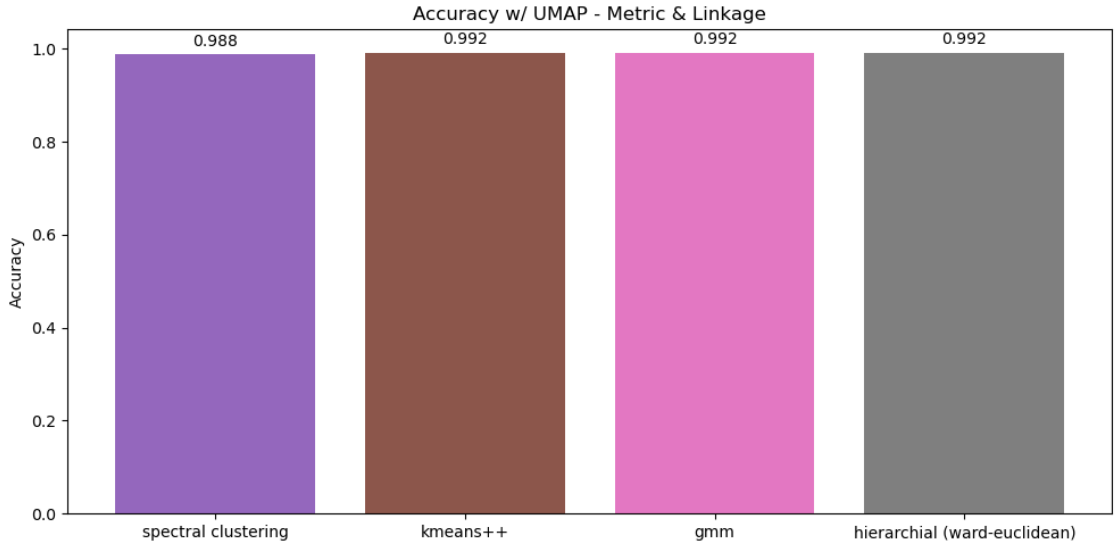
```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy hierarchial clustering with ward and euclidean = 0.9916765755053508
```

2.4 Comparison

As we can see, all these methods (with the exception of spectral clustering as this did not use UMAP) converge to the same accuracy. This is because after applying UMAP we get very clearly defined clusters so there is not much/any room for these methods to diverge in their clustering assignments! Therefore they will yield the same accuracy; this makes sense!

[17]:
```
keys = list(umap_accuracy_dict.keys())
vals = list(umap_accuracy_dict.values())
```

```
colors = ['#9467bd', '#8c564b', '#e377c2', '#7f7f7f']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy w/ UMAP - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods_with_umap')
plt.show()
```



05_generalizibility_silhouette_score

April 17, 2025

Load necessary libraries.

[9]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap
from sklearn.metrics import silhouette_samples

### Msc
import warnings
from PIL import Image

### OOP
from ml_utils import SilhouetteEvaluator, ClusterEvaluator, make_gmm,␣
 ↪make_hierarchical, make_spectral
```

Load dataset.

[10]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array

X_clean = df.drop(columns=['Author'])
X = X_clean.to_numpy()
```

```
# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = umap_model.fit_transform(X)
```

1 Validation

The method with the highest silhouette score generalizes best!

1.1 Kmeans++

Kmeans++ with and without UMAP:

1.1.1 Generalizibility

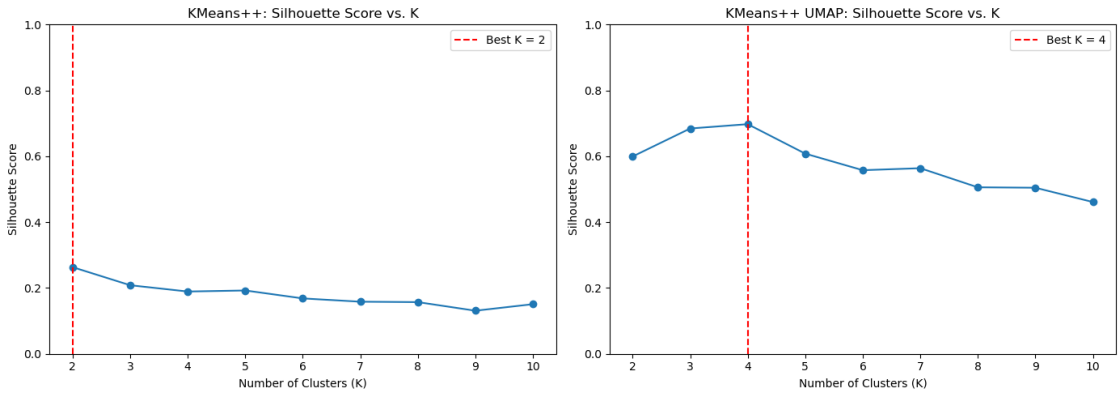
[11]:
```
# Kmeans++
print('\033[1m' + 'Kmeans++ without dimension reduction:' + '\033[0m')
evaluator_kmeans = SilhouetteEvaluator(X, KMeans, k_range=range(2, 11),␣
 ↪init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Kmeans++ with UMAP
print('\033[1m' + 'Kmeans++ with dimension reduction (UMAP):' + '\033[0m')
evaluator_kmeans_umap = SilhouetteEvaluator(X_umap, KMeans, k_range=range(2,␣
 ↪11), init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

y_min = 0
y_max = 1

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_kmeans.plot("KMeans++", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_kmeans_silhouette_score')
plt.show()
```

Kmeans++ without dimension reduction:

```
The scores across K = {2: 0.2632255060236338, 3: 0.20836751818875782, 4:
0.18927180616599434, 5: 0.1924461042273646, 6: 0.16832953504579296, 7:
0.15834519448558668, 8: 0.15708860765446275, 9: 0.13097719253825474, 10:
0.15080653215185813}
Best K: 2
```

Kmeans++ with dimension reduction (UMAP):

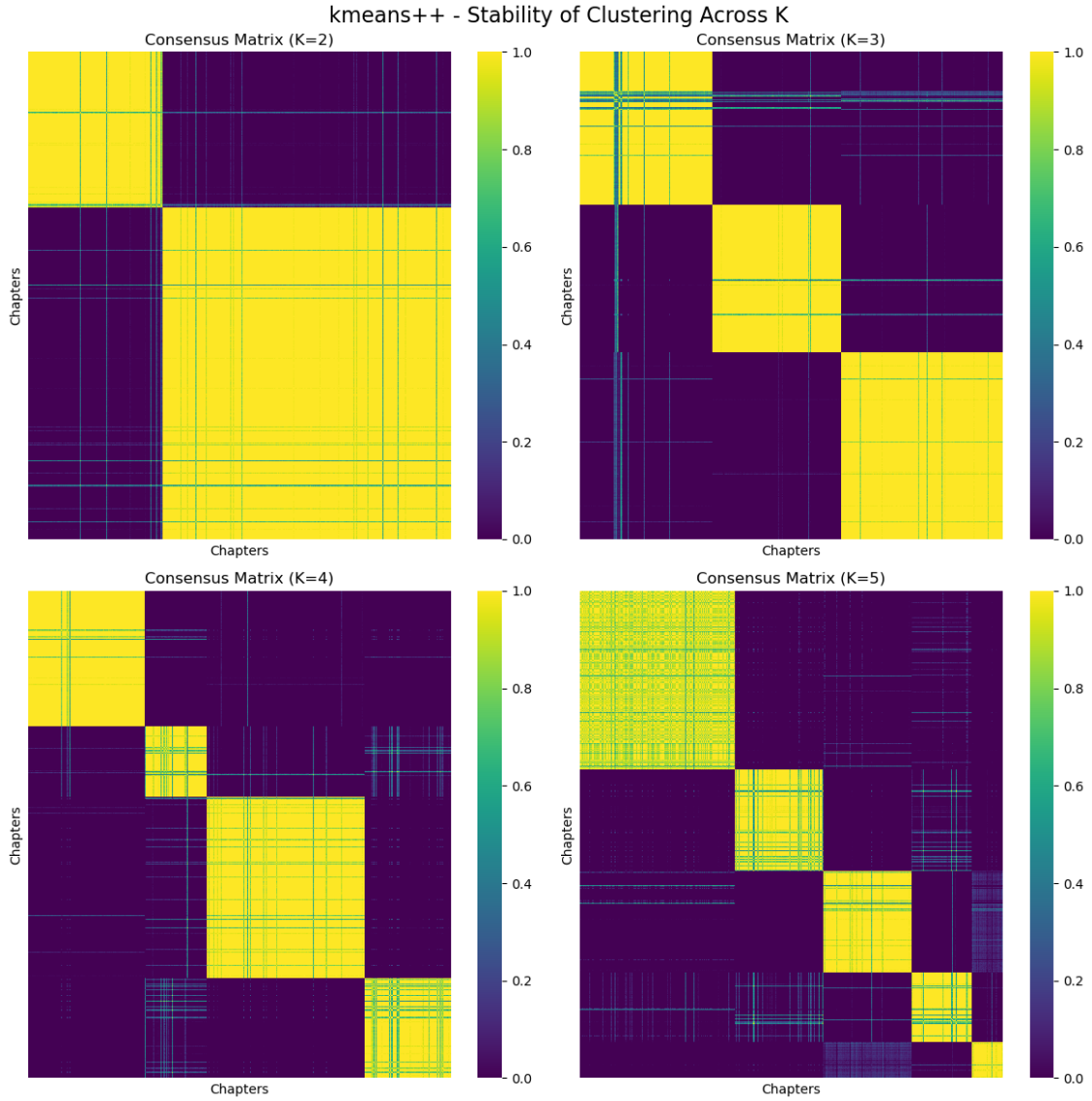
```
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60805935, 6: 0.55773854, 7: 0.5637968, 8: 0.505771, 9: 0.5044795, 10:
0.4610399}
Best K: 4
```



1.1.2 Stability

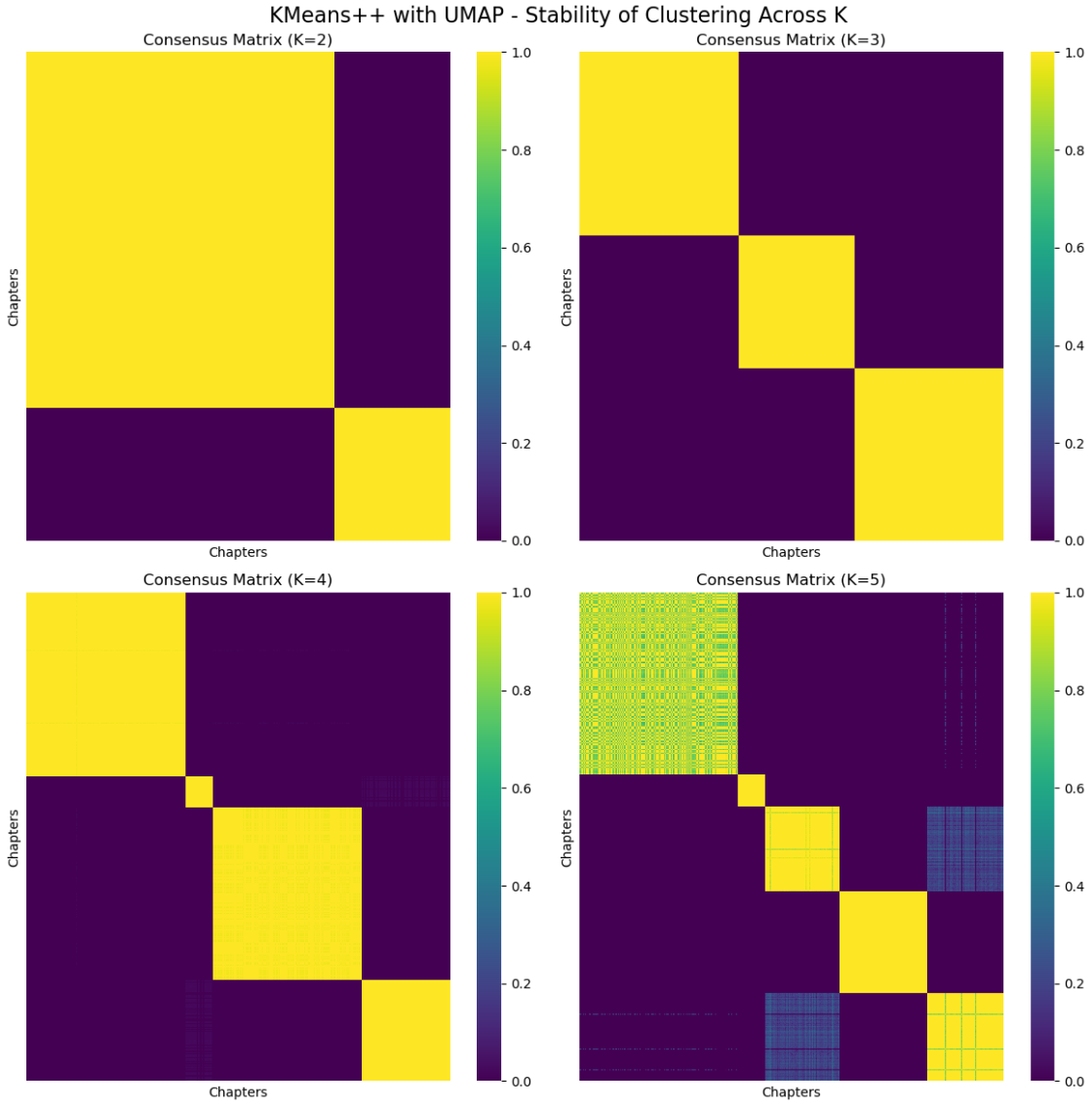
[12]:
```
method = 'kmeans++'

rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_train = X[train_idx]  # Using saved NumPy array
```

```
        km = KMeans(n_clusters=K, random_state=n, n_init=10)
        cluster_labels = km.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, random_state=42, n_init=10).
↪fit_predict(X)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans')
plt.show()
```



[13]:
```
method = 'KMeans++ with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
```

```
    sampled_matrix = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        km = KMeans(n_clusters=K, n_init=10, init='k-means++', random_state=i)
        labels = km.fit_predict(X_sample)

        sampled_matrix[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus_matrix[np.ix_(idx_sample, idx_sample)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=sampled_matrix != 0)
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, n_init=10, init='k-means++',␣
↪random_state=42).fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus_matrix[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans_umap')
plt.show()
```



1.2 Gaussian Mixture Models

1.2.1 Generalizibility

[14]:
```
# GMM
print('\033[1m' + 'GMM without dimension reduction:' + '\033[0m')
evaluator_gmm = SilhouetteEvaluator(X, make_gmm(), k_range=range(2, 11))
scores, best_k = evaluator_gmm.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# GMM with UMAP
print('\033[1m' + 'GMM with dimension reduction (UMAP):' + '\033[0m')
```

```
evaluator_gmm_umap = SilhouetteEvaluator(X_umap, make_gmm(), k_range=range(2,␣
↪11))
scores, best_k = evaluator_gmm_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_gmm.plot("GMM", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()

plt.savefig('Media/viz/05/05_gmm_silhouette_score')
plt.show()
```

GMM without dimension reduction:

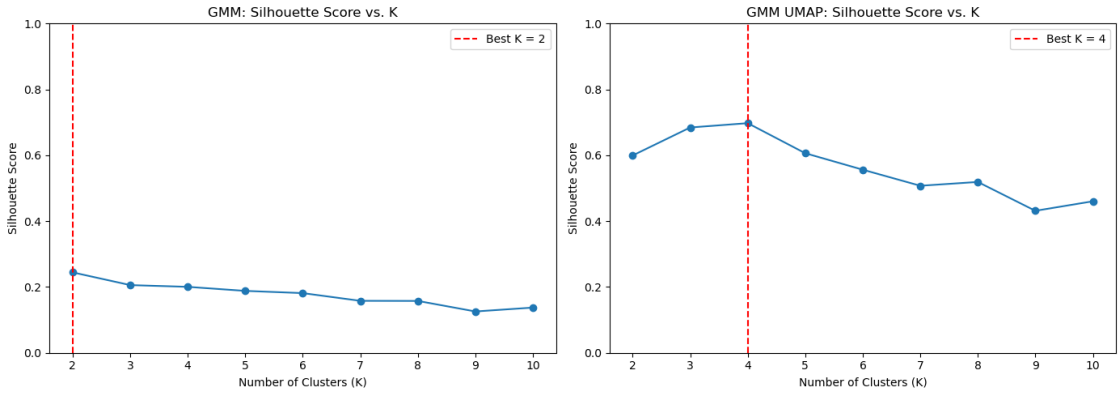
The scores across K = {2: 0.24434605026692022, 3: 0.20578434655866124, 4: 0.2004213780628659, 5: 0.18814970124375807, 6: 0.18136222277608166, 7: 0.15798484734742102, 8: 0.15765164283870803, 9: 0.1256388670716472, 10: 0.1374306014397366}

Best K: 2

GMM with dimension reduction (UMAP):

The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5: 0.60627675, 6: 0.55619246, 7: 0.5074211, 8: 0.51900756, 9: 0.43136632, 10: 0.4602903}

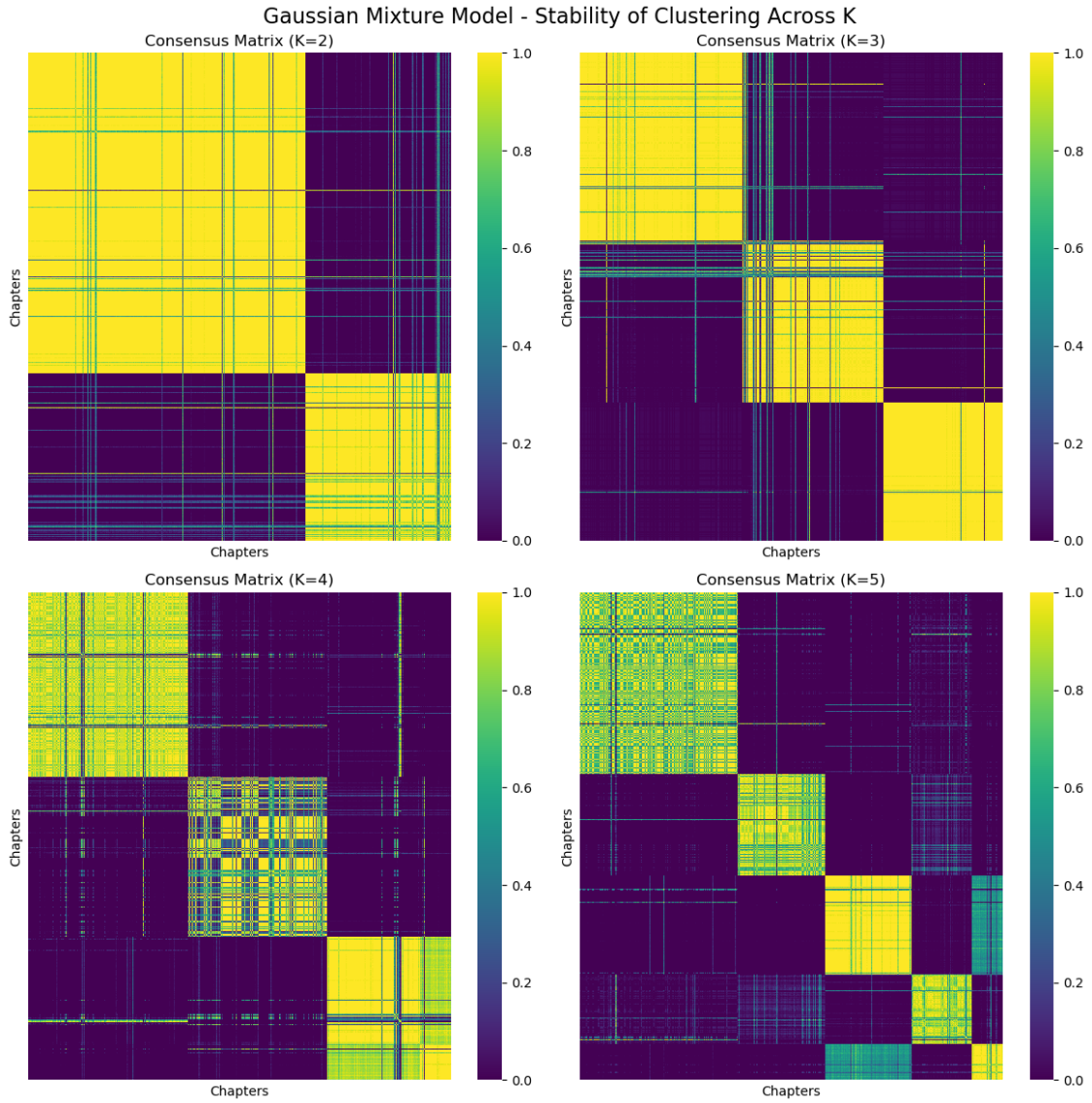
Best K: 4



1.2.2 Stability

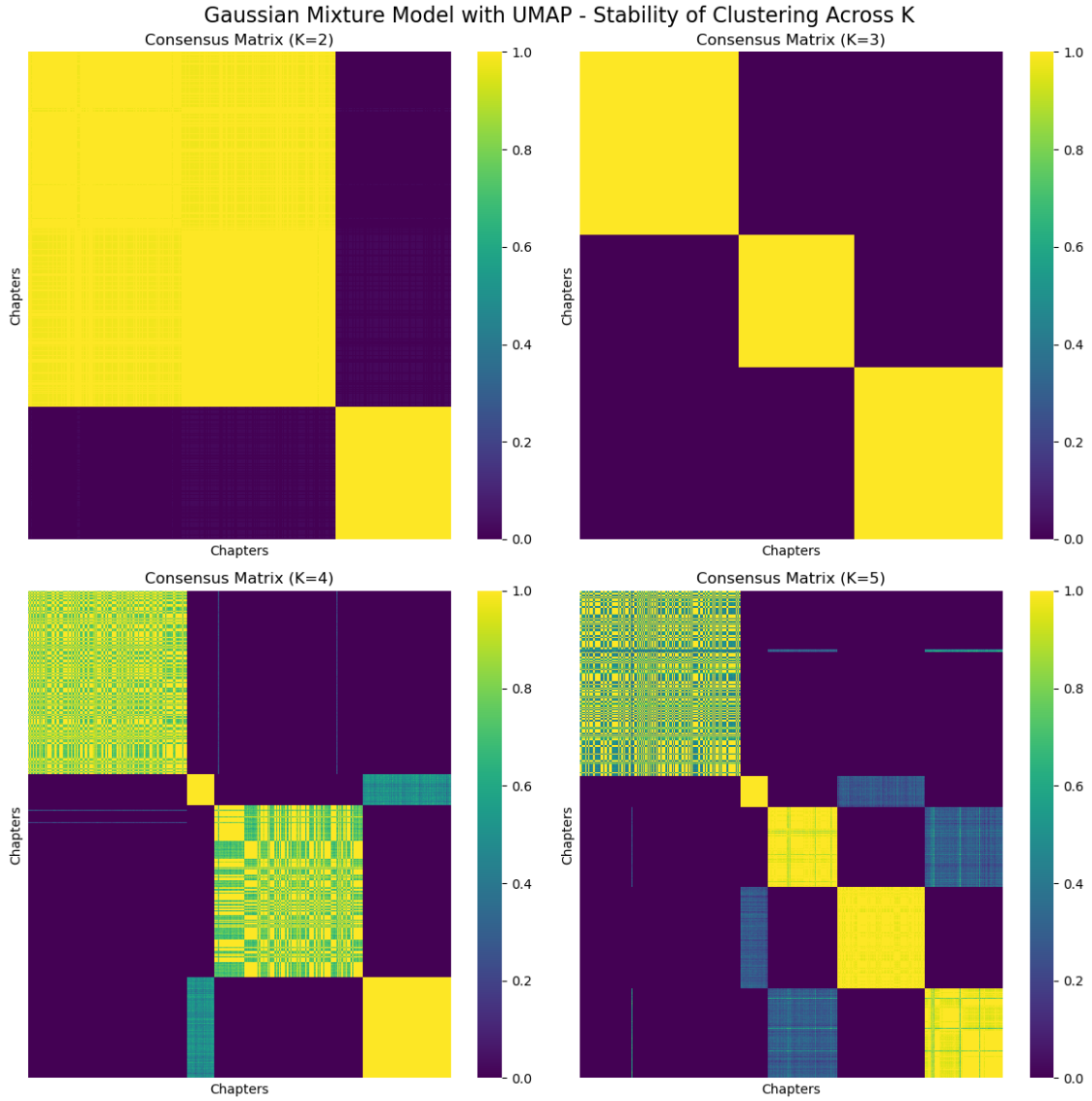
[15]:
```
method = 'Gaussian Mixture Model'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]  # NumPy version for GMM

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X).
↪predict(X)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
```

```
fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm')
plt.show()
```



[16]:
```
method = 'Gaussian Mixture Model with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()
```

```
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_sample = X_umap[idx_sample]

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X_umap).
 ↪predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm_umap')
plt.show()
```



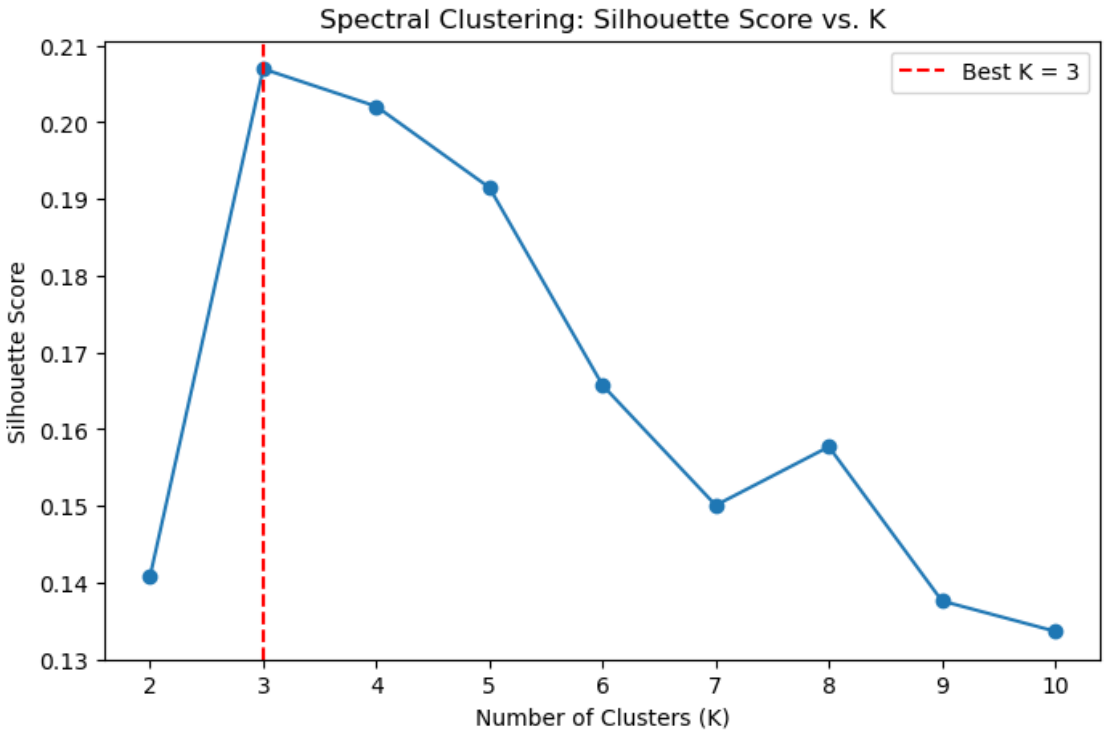
1.3 Spectral Clustering

1.3.1 Generalizibility

[17]:
```
print('\033[1m' + 'Spectral Clustering:' + '\033[0m')
evaluator_spectral = SilhouetteEvaluator(X, make_spectral(affinity =␣
 ↪'nearest_neighbors'), k_range=range(2, 11))
scores, best_k = evaluator_spectral.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

evaluator_spectral.plot('Spectral Clustering')
```

```
plt.savefig('Media/viz/05/05_spectral_silhouette_score')
```

```
Spectral Clustering:
The scores across K = {2: 0.1408300984760483, 3: 0.20696881798137212, 4:
0.2020762541851448, 5: 0.1914837165759898, 6: 0.16570630789830376, 7:
0.15006558354299535, 8: 0.1577000857169114, 9: 0.13758976585902763, 10:
0.1336400256361507}
Best K: 3
```



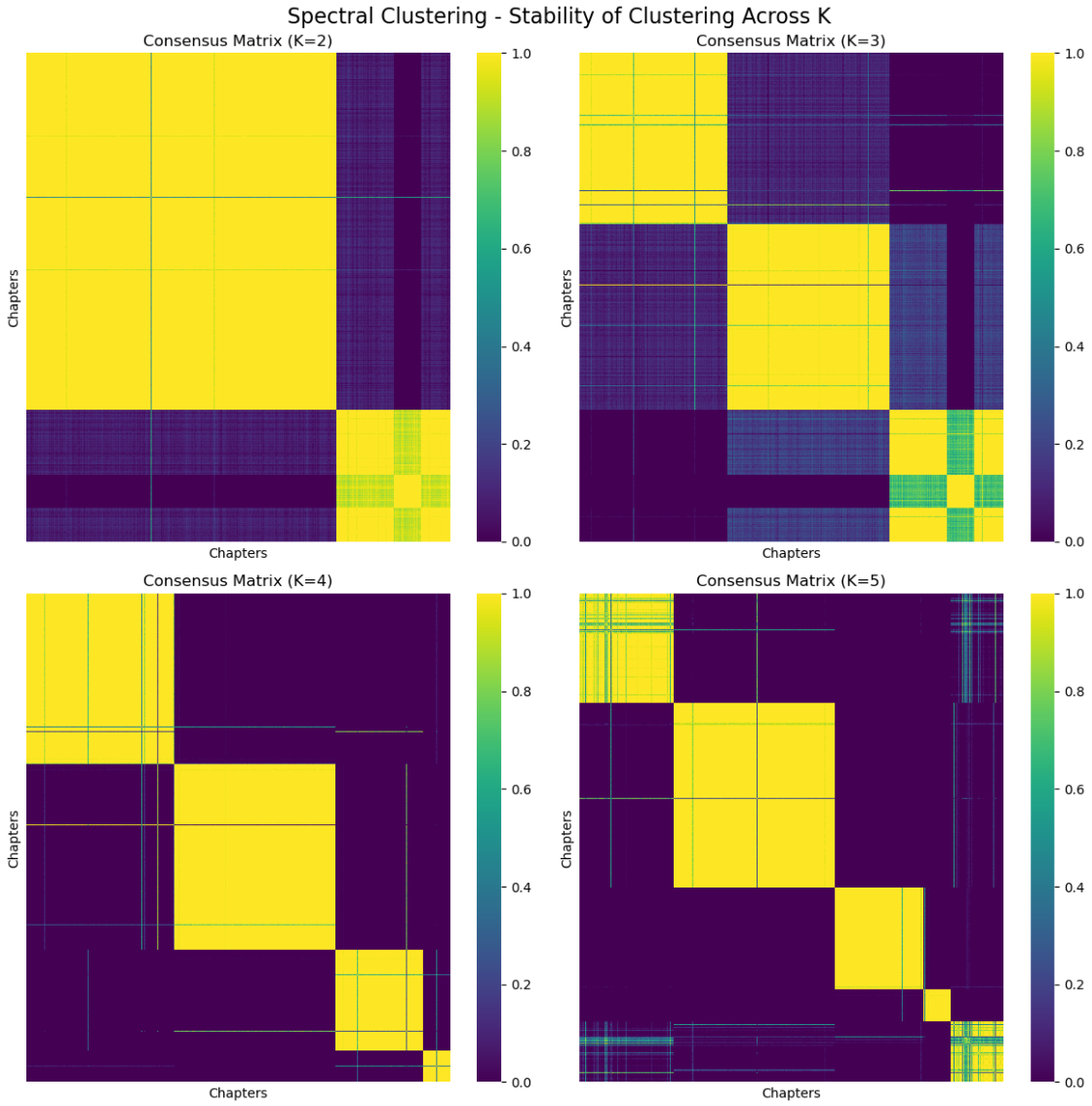
Although the silhouette score is commonly used to evaluate clustering performance, it assumes that clusters are spherical in shape, which does not align with the strengths of spectral clustering. Spectral clustering excels at uncovering non-linear and arbitrarily shaped cluster structures, which silhouette score is not well-equipped to quantify. Therefore, the low silhouette score observed here may understate the actual performance of spectral clustering. In fact, the spectral embedding reveals visually distinct and meaningful clusters (see notebook 02), suggesting that the method is effective despite the numerical metric.

1.3.2 Stability

[18]:
```
## METHOD
method = 'Spectral Clustering'

rng = np.random.default_rng(42)
iterations = 100
```

```
K_values = [2, 3, 4, 5]
n_samples = X_clean.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_train = X_clean.iloc[train_idx].to_numpy()

        sc = SpectralClustering(n_clusters=K, affinity='nearest_neighbors',␣
 ↪assign_labels='kmeans', random_state=n)
        cluster_labels = sc.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
 ↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = SpectralClustering(n_clusters=K,␣
 ↪affinity='nearest_neighbors', assign_labels='kmeans', random_state=42).
 ↪fit_predict(X_clean)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
 ↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
```

```
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_spectral')
plt.show()
```



$K = 4$ is the clear winner!

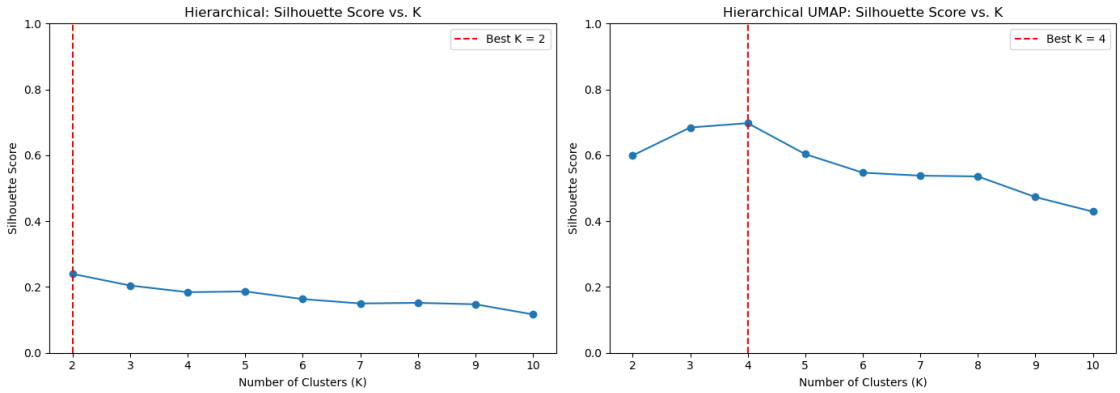
1.4 Hierarchial Clustering

Using the parameters 'ward' and 'euclidean' (because we have seen in notebook 04 that these have the highest accuracy)!

1.4.1 Generalizibility

[19]:
```
# Hierarchical Clustering
print('\033[1m' + 'Hierarchical Clustering without dimension reduction:' +␣
 ↪'\033[0m')
evaluator_hier = SilhouetteEvaluator(X, make_hierarchical(linkage='ward',␣
 ↪metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Hierarchical Clustering with UMAP
print('\033[1m' + 'Hierarchical Clustering with dimension reduction (UMAP):' +␣
 ↪'\033[0m')
evaluator_hier_umap = SilhouetteEvaluator(X_umap,␣
 ↪make_hierarchical(linkage='ward', metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_hier.plot("Hierarchical", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_silhouette_score_across_methods.png')
plt.show()
```

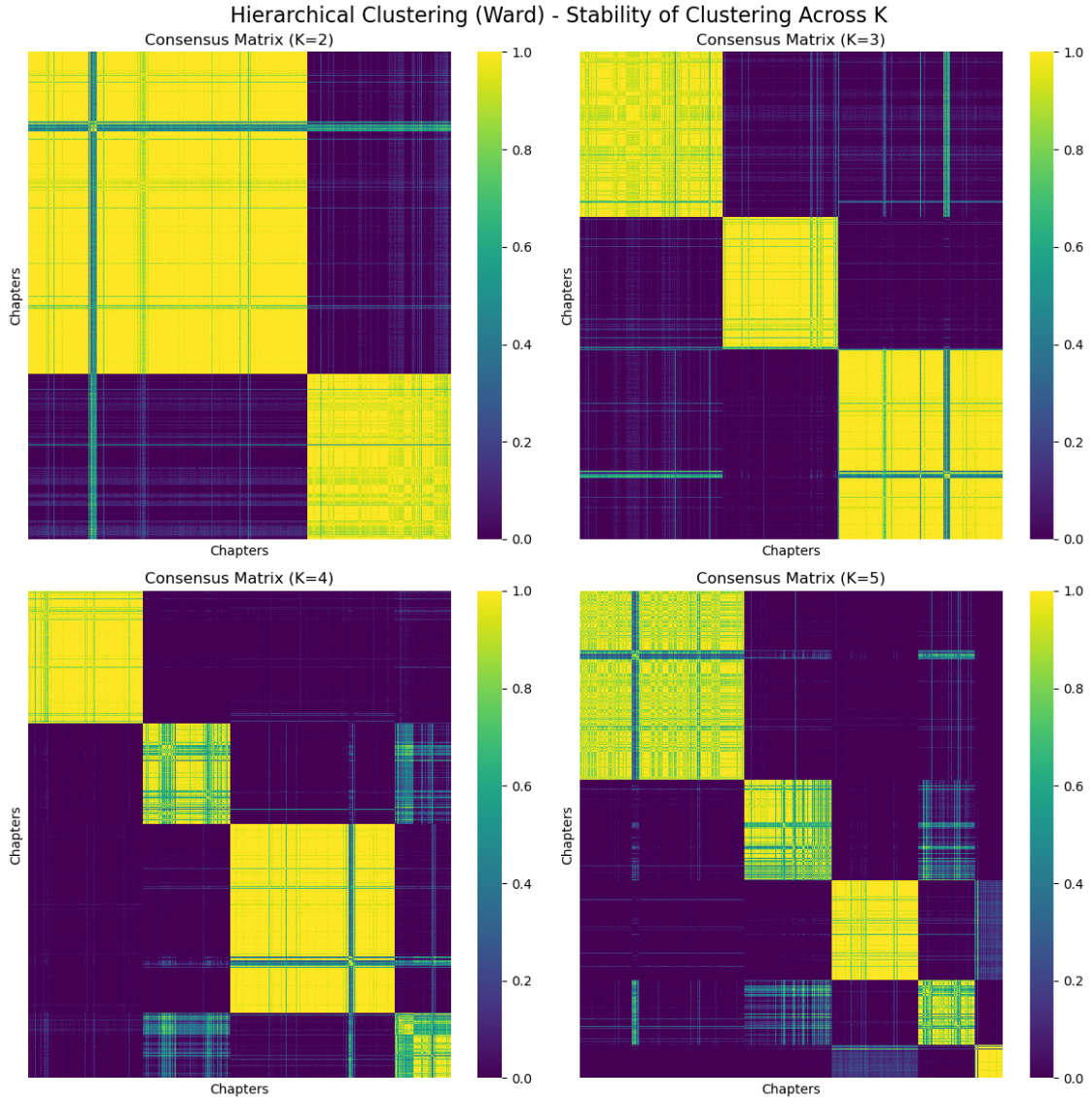
```
Hierarchical Clustering without dimension reduction:
The scores across K = {2: 0.23967574687100793, 3: 0.20437373269232859, 4:
0.18421197778513682, 5: 0.18654800991576861, 6: 0.16319550456880302, 7:
0.14995813568113975, 8: 0.1518500219855489, 9: 0.14745055233611987, 10:
0.1167264049290792}
Best K: 2
Hierarchical Clustering with dimension reduction (UMAP):
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60349095, 6: 0.5471597, 7: 0.5380937, 8: 0.53579855, 9: 0.47313103, 10:
0.42873582}
Best K: 4
```



1.4.2 Stability

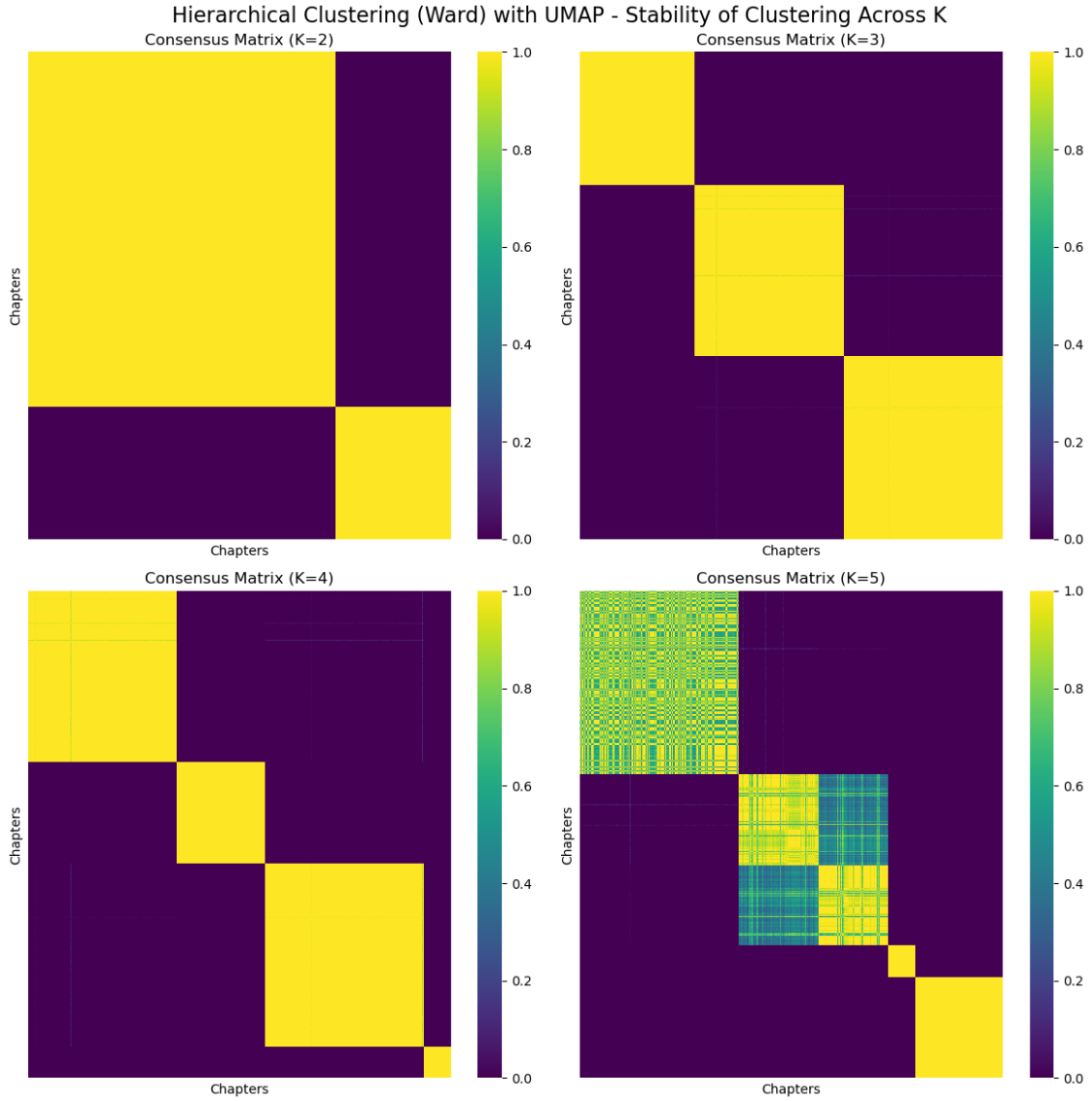
[20]:
```
method = 'Hierarchical Clustering (Ward)'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X)
```

```
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical')
plt.show()
```



[21]:
```
method = 'Hierarchical Clustering (Ward) with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
```

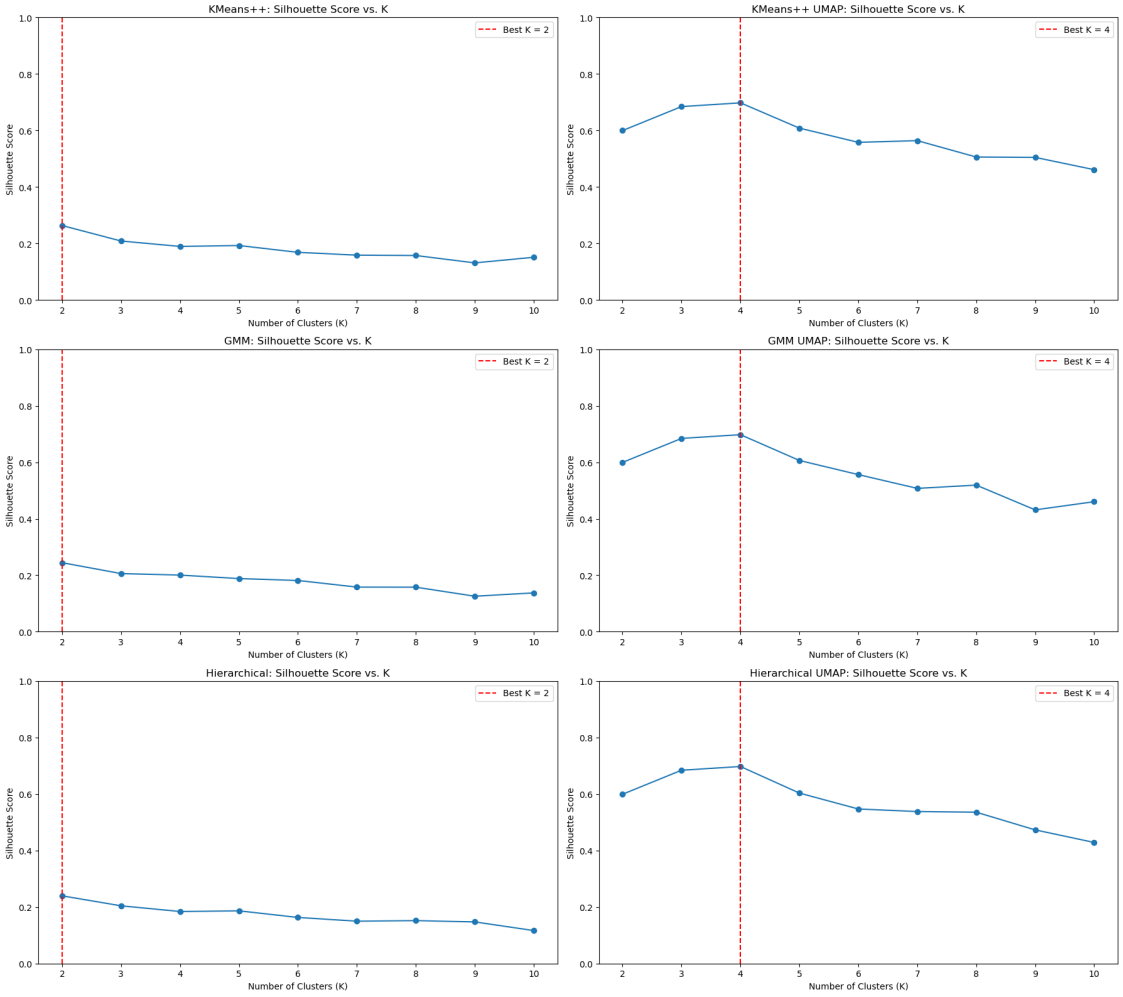
```
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical_umap')
plt.show()
```



2 Comparison Viz

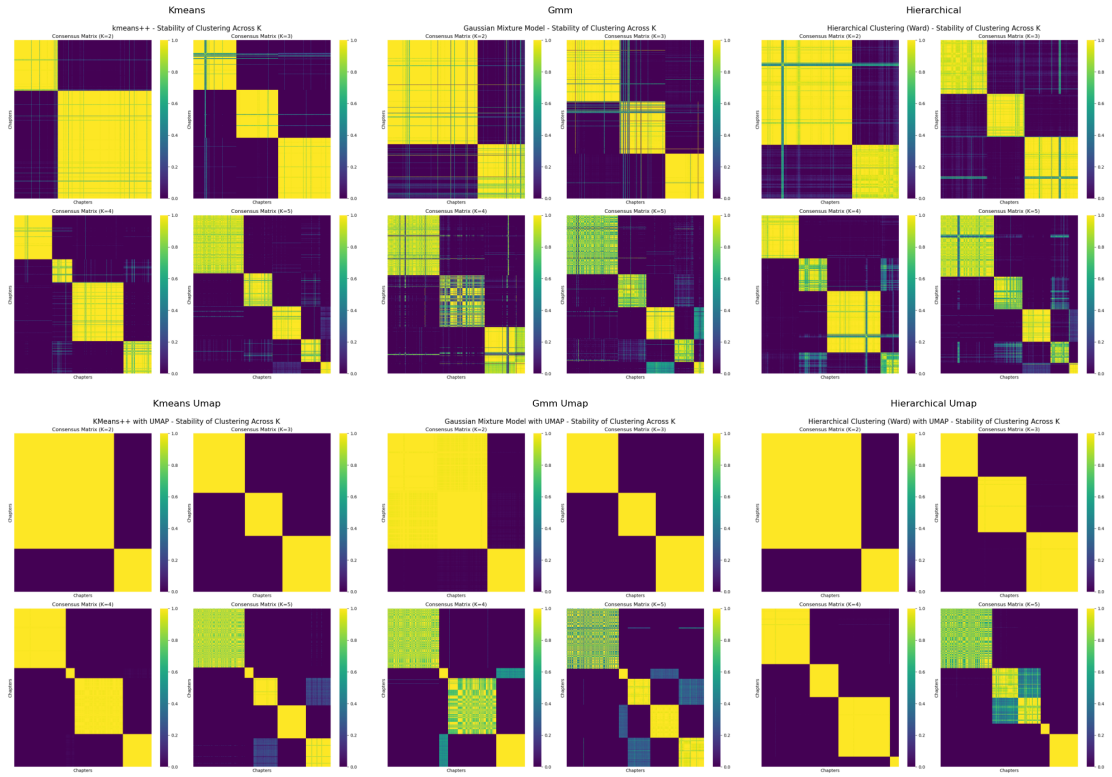
[22]:
```
fig, axs = plt.subplots(3, 2, figsize=(18, 16))

evaluator_kmeans.plot("KMeans++", ax=axs[0,0])
axs[0,0].set_ylim(y_min, y_max)
evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[0,1])
axs[0,1].set_ylim(y_min, y_max)

evaluator_gmm.plot("GMM", ax=axs[1,0])
axs[1,0].set_ylim(y_min, y_max)
```

```
evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1,1])
axs[1,1].set_ylim(y_min, y_max)

evaluator_hier.plot("Hierarchical", ax=axs[2,0])
axs[2,0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[2,1])
axs[2,1].set_ylim(y_min, y_max)

plt.savefig('Media/viz/05/05_silhouette_score_across_methods')
plt.tight_layout()
```



[26]:
```
heatmap_paths = [
    "Media/viz/05/05_consensus_heatmaps_kmeans.png",
    "Media/viz/05/05_consensus_heatmaps_gmm.png",
```

```
    "Media/viz/05/05_consensus_heatmaps_hierarchical.png",
    "Media/viz/05/05_consensus_heatmaps_kmeans_umap.png",
    "Media/viz/05/05_consensus_heatmaps_gmm_umap.png",
    "Media/viz/05/05_consensus_heatmaps_hierarchical_umap.png"
]

heatmap_images = [Image.open(path) for path in heatmap_paths]

fig, axes = plt.subplots(2, 3, figsize=(20, 14), constrained_layout=True)
axes = axes.flatten()
for ax, img, path in zip(axes, heatmap_images, heatmap_paths):
    ax.imshow(img)
    ax.set_title(
        os.path.basename(path)
          .replace("05_consensus_heatmaps_", "")
          .replace(".png", "")
          .replace("_", " ")
          .title(),
        fontsize=12
    )
    ax.axis('off')
plt.savefig("Media/viz/05/05_all_consensus_heatmaps_grid.png")
plt.show()
```



06_em_algorithm_fit_gmm

April 17, 2025

Load necessary libraries.

[2]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

### ML packages
from scipy.stats import multivariate_normal
from sklearn.mixture import GaussianMixture
import umap
from sklearn.metrics import adjusted_rand_score
from scipy.spatial.distance import cdist

### Msc
import warnings

### OOP Code
from ml_utils import GaussianMixtureEM
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array
X = df.drop(columns=['Author'])

# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = pd.DataFrame(umap_model.fit_transform(X))
```

EM Algorithm for Multivariate Gaussian Mixture Model

Let the density of $x \in \mathbb{R}^d$ be: $p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x \mid \mu_k, \Sigma_k)$

where: - π_k are the mixing proportions (with $\sum_k \pi_k = 1$), - $\mu_k \in \mathbb{R}^d$ is the mean of component k, - $\Sigma_k \in \mathbb{R}^{d \times d}$ is the covariance matrix.

E-step

$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i|\mu_j,\Sigma_j)}$ where $\mathcal{N}(x \mid \mu, \Sigma)$ is the multivariate Gaussian density: $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i$

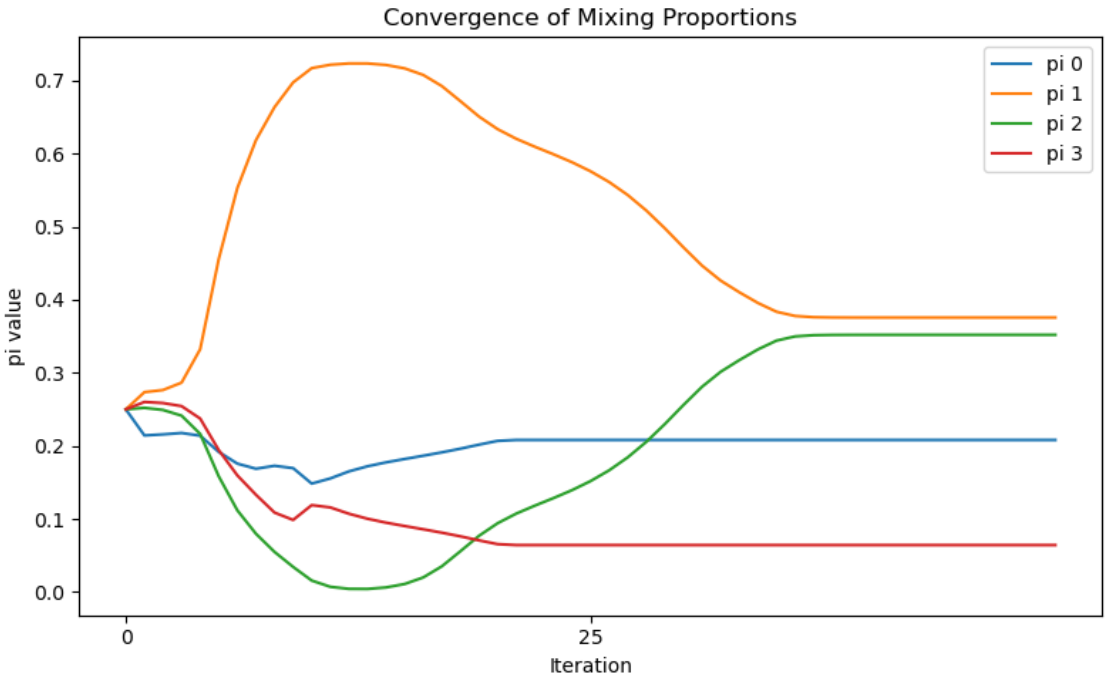
Update covariance: $\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

Update mixing weights: $\pi_k = \frac{N_k}{N}$

Repeat E-step and M-step until convergence (e.g., change in log-likelihood is below a threshold).

[4]:
```
# code in -> ml_utils.py
# code also pasted at appendix
model = GaussianMixtureEM(K=4, num_iterations=50, allow_singular=False)
results = model.fit_fast(X_umap)
```

[5]:
```
pis_dict = results['pis_dict']
iterations = [key for key in pis_dict if key.startswith('Iteration_')]
iterations_sorted = sorted(iterations, key=lambda x: int(x.split('_')[1]))

pi_matrix = np.array([pis_dict['Initial'][0]] + [pis_dict[it][0] for it in␣
 ↪iterations_sorted])

plt.figure(figsize=(8, 5))
for k in range(pi_matrix.shape[1]):
    plt.plot(range(len(pi_matrix)), pi_matrix[:, k], label=f'pi {k}')

plt.title('Convergence of Mixing Proportions')
plt.xlabel('Iteration')
plt.ylabel('pi value')

steps = 25
plt.xticks(range(0, len(iterations_sorted), steps))

plt.legend()
plt.tight_layout()
plt.savefig('Media/viz/06/06_em_convergence_plot')
plt.show()
```



1 Compare

Now to compare my manually EM function to the built in Sklearn package!

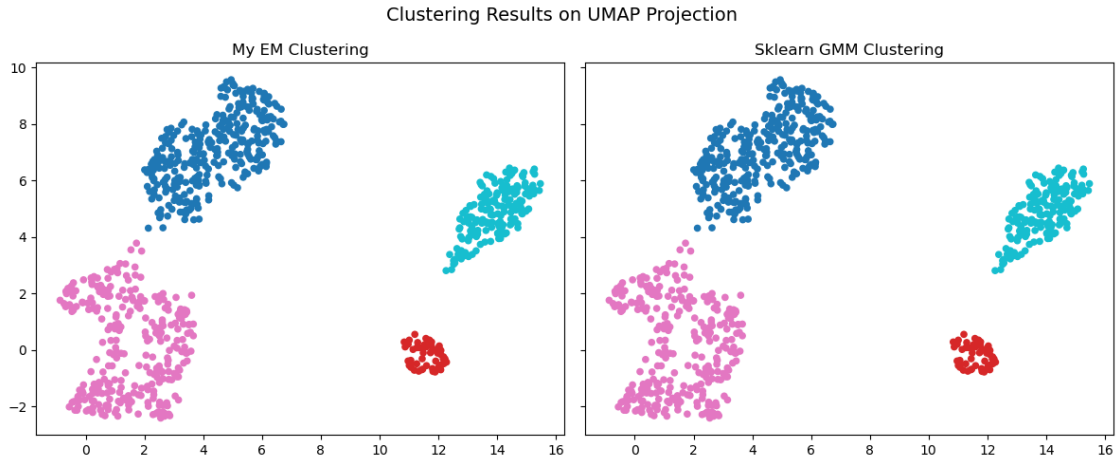
[6]:
```
sk_model = GaussianMixture(n_components=4, random_state=42)
sk_model.fit(X_umap)

# Your final results
my_pi = results['pis_dict'][f'Iteration_{model.num_iterations - 1}'][0]
my_mu = np.array(results['means'])
my_cov = np.array(results['cov'])

# Sklearn results
sk_pi = sk_model.weights_
sk_mu = sk_model.means_
sk_cov = sk_model.covariances_
```

[7]:
```
dist_matrix = cdist(my_mu, sk_mu)
matches = np.argmin(dist_matrix, axis=1)

print("Matching components based on mean proximity:")
for i, j in enumerate(matches):
    print(f"Component {i} → Sklearn {j} | Distance: {dist_matrix[i, j]:.4f}")
```

```python
print("\nMixing weights comparison:")
for i, j in enumerate(matches):
    print(f"Component {i}: mine={my_pi[i]:.4f} | sklearn={sk_pi[j]:.4f}")

print("\nMean vector L2 distances:")
for i, j in enumerate(matches):
    delta = np.linalg.norm(my_mu[i] - sk_mu[j])
    print(f"Component {i}: || _mine -  _sklearn|| = {delta:.4f}")

my_labels = np.argmax(results['gamma'], axis=1)
sk_labels = sk_model.predict(X_umap)
ari = adjusted_rand_score(my_labels, sk_labels)

print(f"\nAdjusted Rand Index: {ari:.4f}")

remapped_labels = np.zeros_like(my_labels)
for i, j in enumerate(matches):
    remapped_labels[my_labels == i] = j

X_umap_np = X_umap.to_numpy() if isinstance(X_umap, pd.DataFrame) else X_umap

fig, axs = plt.subplots(1, 2, figsize=(12, 5), sharex=True, sharey=True)

axs[0].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=remapped_labels,␣
 ↪cmap='tab10', s=20)
axs[0].set_title("My EM Clustering")

axs[1].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=sk_labels, cmap='tab10',␣
 ↪s=20)
axs[1].set_title("Sklearn GMM Clustering")

plt.suptitle("Clustering Results on UMAP Projection", fontsize=14)
plt.tight_layout()
plt.savefig('Media/viz/06/06_label_2dim_comparison_viz')
plt.show()
```

```
Matching components based on mean proximity:
Component 0 → Sklearn 3 | Distance: 0.0000
Component 1 → Sklearn 0 | Distance: 0.0000
Component 2 → Sklearn 2 | Distance: 0.0001
Component 3 → Sklearn 1 | Distance: 0.0000

Mixing weights comparison:
Component 0: mine=0.2081 | sklearn=0.2081
Component 1: mine=0.3757 | sklearn=0.3757
Component 2: mine=0.3520 | sklearn=0.3520
Component 3: mine=0.0642 | sklearn=0.0642
```

```
Mean vector L2 distances:
Component 0: || _mine -  _sklearn|| = 0.0000
Component 1: || _mine -  _sklearn|| = 0.0000
Component 2: || _mine -  _sklearn|| = 0.0001
Component 3: || _mine -  _sklearn|| = 0.0000

Adjusted Rand Index: 1.0000
```

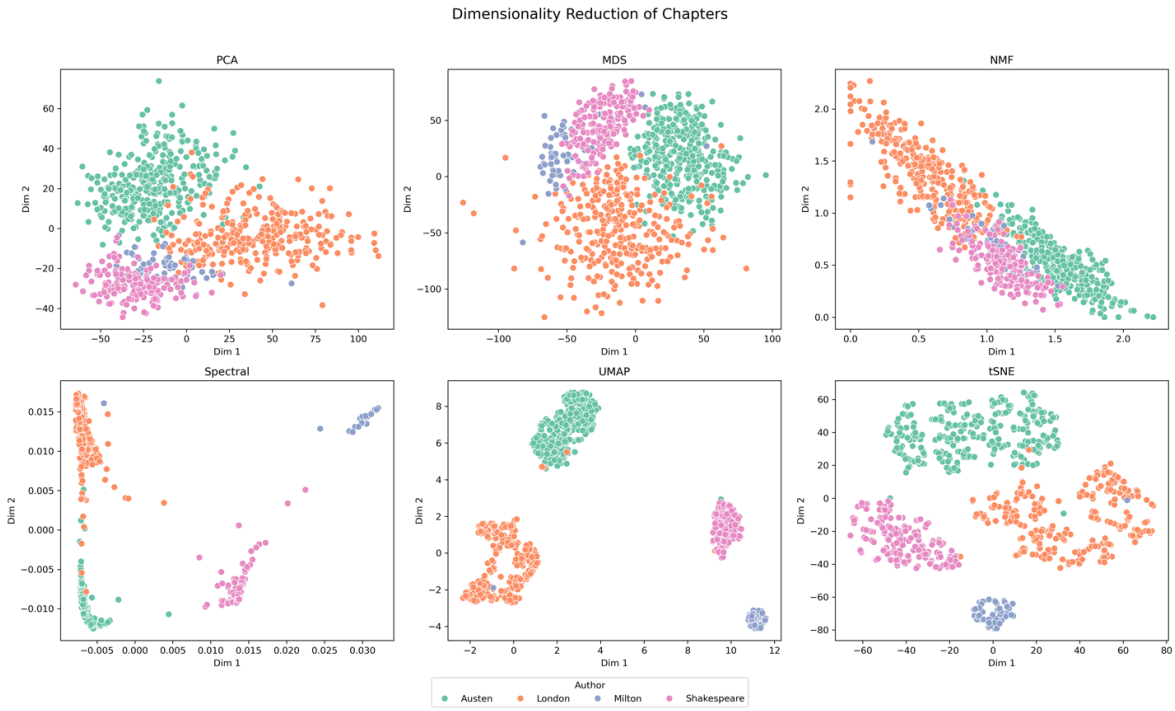


Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!

Visualizations

■ *observations (book chapters)*

Implementing PCA, MDS, NMF, Spectral Embedding, UMAP, t-SNE and validating with author labels (observations = chapters) gives the following visualizations.

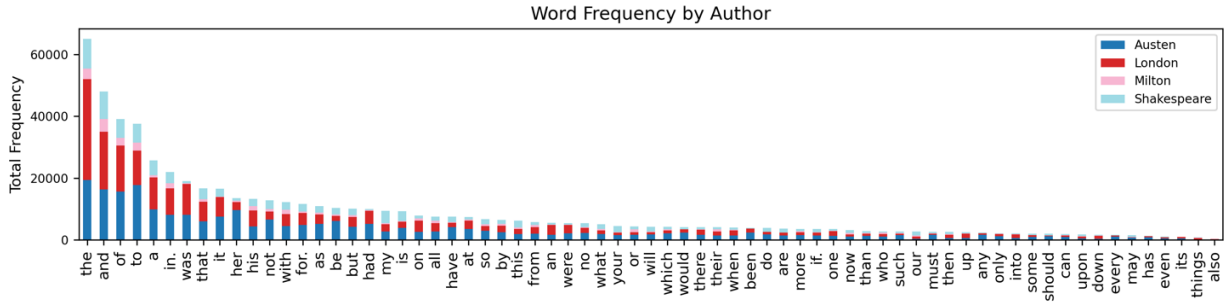


The linear methods perform worse in creating distinguishable clusters (without labels to validate PCA, MDS, and NMF would just look like "blobs" of observations) leading to more overlap and less informative projections; the structure of the data appears to be non-linear, evidenced by the superior performance of non-linear dimensionality reduction techniques. Among these, UMAP stands out—it produces well-separated, spherical clusters that significantly enhance interpretability and visual clarity. Compared to other non-linear approaches like t-SNE and Spectral Embedding, UMAP strikes a balance between global and local structure, revealing distinct author-based groupings.

In terms of interpretability, the visual above supports the claim that generally for this dataset (among the tested methods) the ***'best' interpretation = UMAP***.

■ *features (words)*

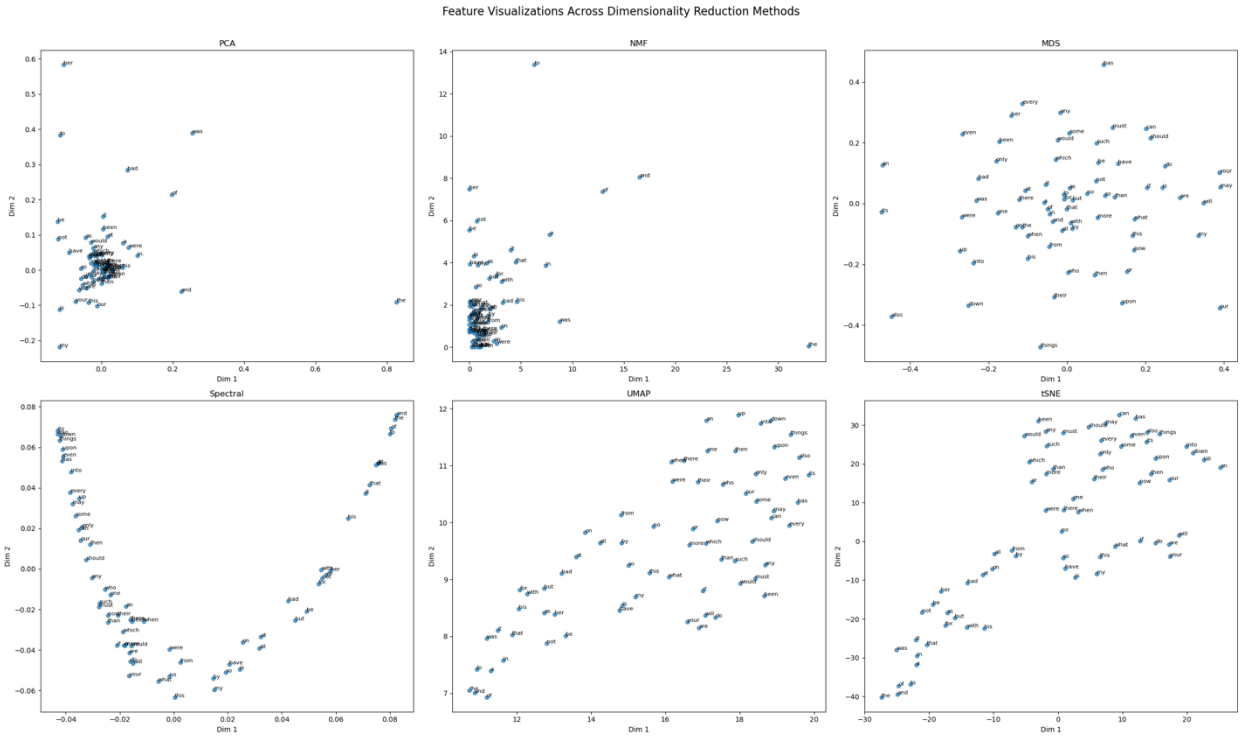
First off, the simplest visualization for the features/words is just a bar plot of word count across words!



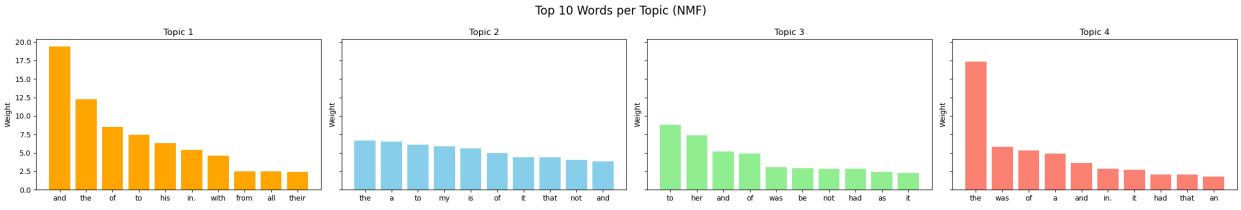
To further analyze patterns among features, I used their co-occurrences across chapters, without knowledge of authorship, to explore whether semantically or stylistically coherent word groups emerge. NMF is particularly well-suited for analyzing features (words) in an unsupervised

setting because it produces a parts-based, non-negative decomposition of the document-term matrix, allowing each topic to be represented as an additive combination of real words. This leads to intuitive and interpretable results, where the top contributing words for each topic reveal coherent semantic or stylistic themes—such as narrative tone, reflective voice, or dialogue. In contrast, methods like PCA yield components with mixed signs that are harder to interpret, and nonlinear methods like UMAP and Spectral Embedding embed words in low-dimensional space without preserving clear word-level semantics. Unlike these methods, NMF enables both dimensionality reduction and human-interpretable insights, making it the most effective choice for visualizing and understanding word-level structure in the data.

For example, visualizations on the features can be given by the following (below). Although this provides some information it is not evident and easily interpreted. So this method is **not** preferable for visual interpretation!



Going back to analyzing the NMF topics for semantic themes we can provide the following visualization of the top 10 words per topic (highest weight). This visualization in combination with the topic weights per chapter

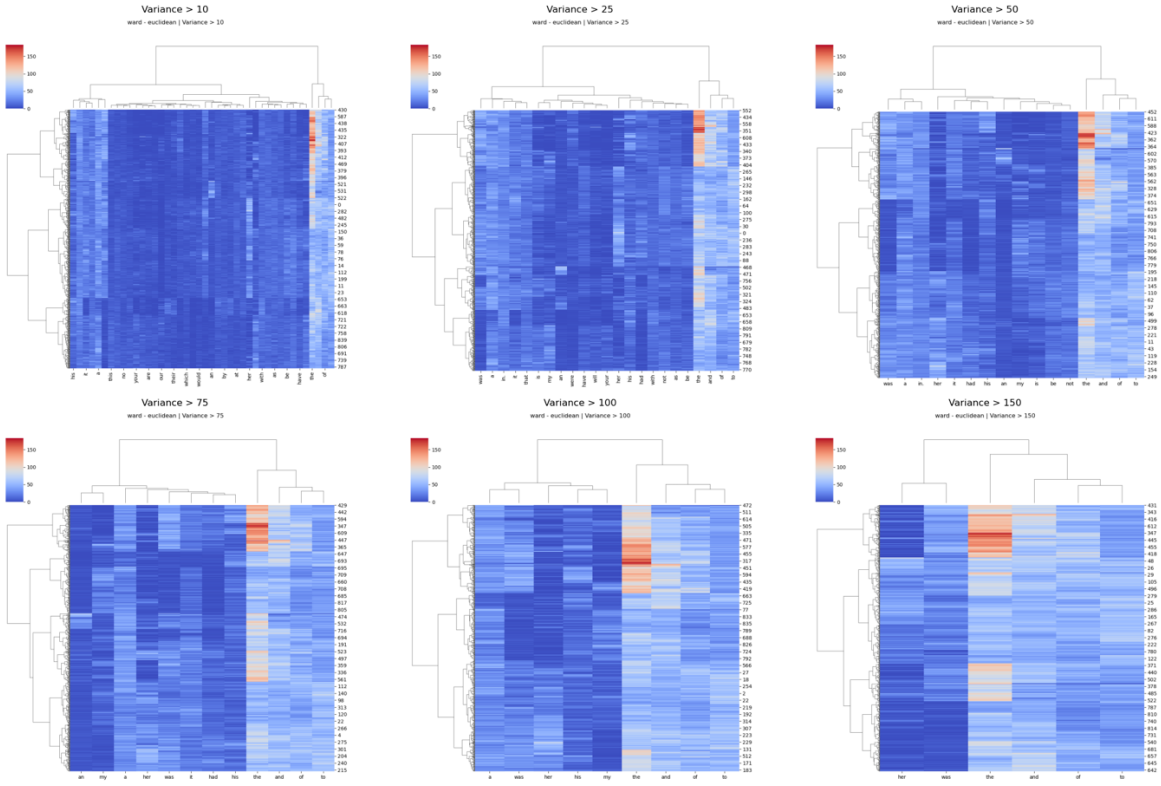


Using this it is far easier to establish semantic themes and distinguish the chapters from another according to which chapters are composed of which topics! The four topics extracted via NMF reveal distinct semantic patterns. **Topic 1** appears to reflect structural scaffolding of narrative prose, with high-weighted function words like *and*, *the*, *of*, *to*, and *with* suggesting continuous sentence construction, descriptive flow, and background exposition. **Topic 2** leans into a more introspective or active narrative stance, driven by frequent use of *my*, *is*, *to*, and *that*, pointing toward subjectivity, dialogue, or first-person accounts. **Topic 3** evokes personal and relational storytelling, emphasized by words such as *to*, *her*, *was*, *be*, and *had*, which are often found in emotionally driven or character-focused scenes. Lastly, **Topic 4** is anchored in definitive and declarative language—*the*, *was*, *and*, *had*, and *it*—potentially signaling reflective, philosophical, or climactic exposition. Together, these topic-word distributions highlight nuanced stylistic layers across authors, helping distinguish thematic patterns embedded in the text. This visual alone provides significant interpretation about the features.

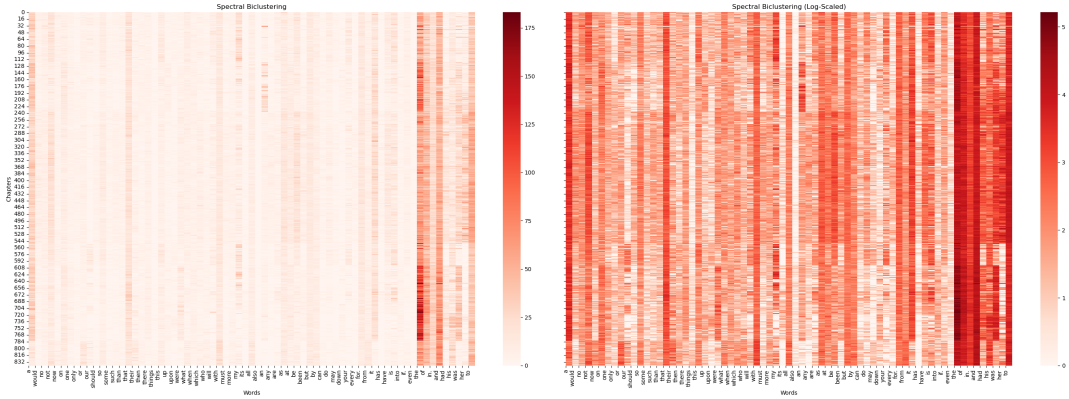
In terms of interpretability, the ***'best/great' interpretation*** is provided by ***NMF***. Now this leads to **1Ac** where we tie together these trends in words to the features which provides further value to interpretation!

■ *both - observations & features*

To visualize both observations and features I used Spectral Bi-clustering and Hierarchical Bi-clustering. To reduce noise in the heatmaps, I filtered out the highest variance features based on some threshold. I have plotted the features retained for each threshold across multiple thresholds below. The x-axis contains the words with high variance across chapters where significant chapters are plotted on the y-axis. If we look at the sparsest heatmap, for the highest threshold of variance > 150 (bottom right) we see bands emerge which provide significant interpretations. For example, if we go back to the NMF topics, for the word '*the'* we can see the red band for chapters 347 and 445 meaning it is likely that these chapters contain topic 1 or topic 4!



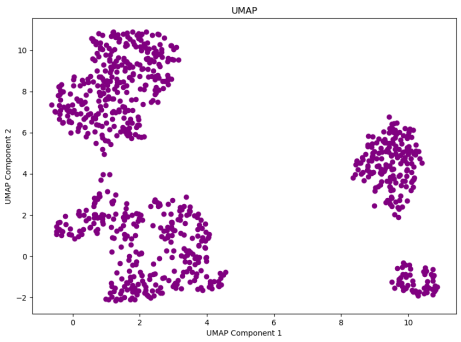
In other words, the features that have high contract among chapters allows us to interpret potential themes/topics among the chapters! I have also produced a heatmap using the Spectral Bi-clustering method which provided a decent visualization. However, clearly Hierarchical Bi-clustering outperforms this method.

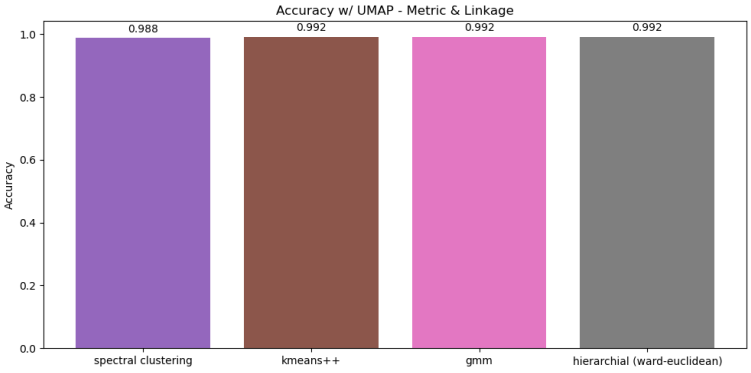


In terms of interpretability, the ***'best/great' interpretation*** is provided by ***Hierarchical Bi-clustering'***.

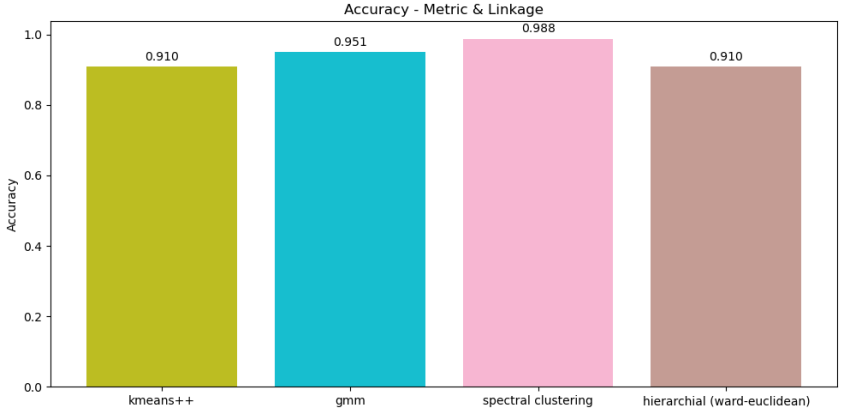
Clustering Methods

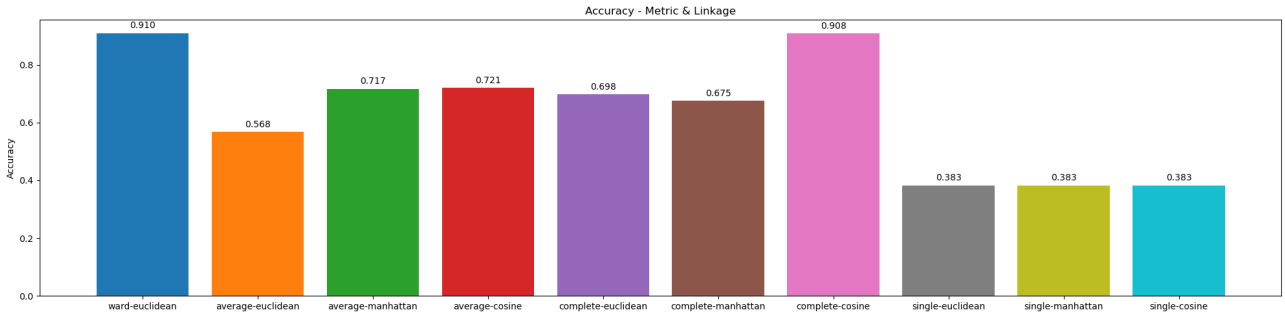
Since UMAP proved to be the superior dimension reduction technique for this dataset, I compared the clustering accuracy across methods on the UMAP transformed data (except for spectral clustering as this already has a built-in dimension reduction – spectral embedding). The results were not surprisingly the same across all methods at 0.992 because UMAP did such a good job at creating clusters that there were no points where the methods would diverge in predicting label outcomes! To make it evident, look at the UMAP data (without labels) – there are 4 clusters, and which points belongs to which clusters! Therefore, apply any methods to UMAP will yield the same labels for each point.





Now spectral clustering performs spectral embedding, so the method accuracy diverges from the methods processed on UMAP transformed data. Since UMAP dimension reduction made comparison redundant, I applied the same techniques to the raw dataset! Furthermore, I ran hierarchical clustering across many hyperparameter values for metrics and linkage.

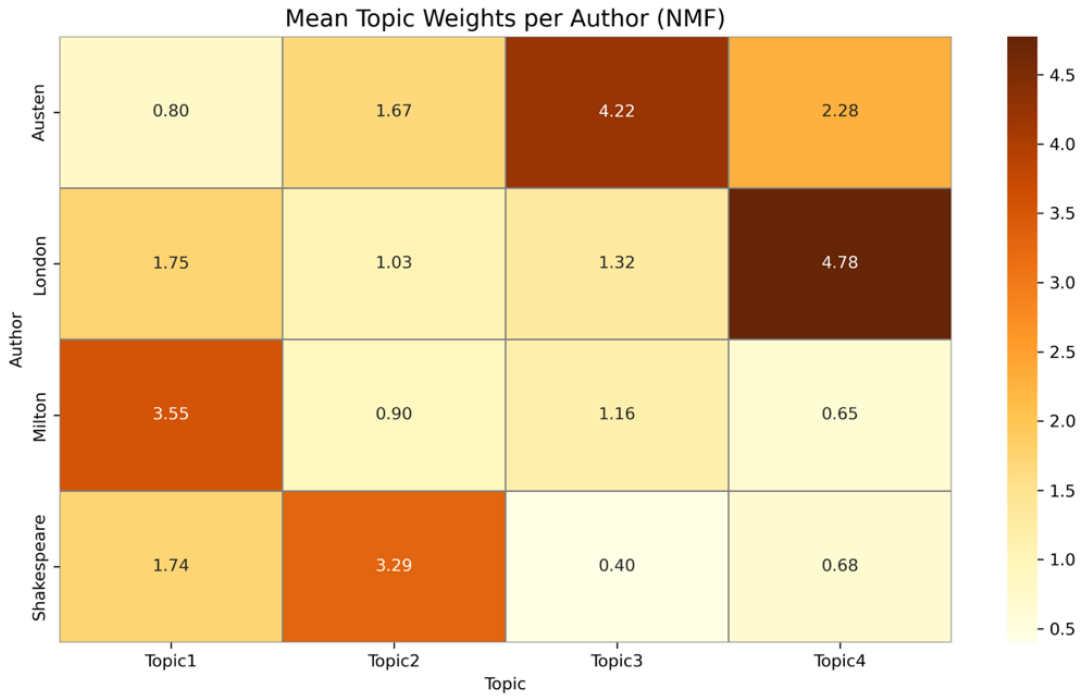




Without UMAP, performance dropped slightly yet still maintained strong results (of course spectral embedding yielded the same results). Among the metric and linkage parameters, Ward linkage with Euclidean distance yielded the highest accuracy, rivaling the performance of UMAP-based methods, while others like single-linkage performed poorly. Overall, UMAP significantly improves clustering accuracy and consistency, though the choice of metric and linkage remains critical when UMAP is not used

Pattern Recognition

Now using supervised learning to predict the author given the distribution of words per chapter! For this task I went back to the NMF topics because they revealed a lot about the dataset. I aimed to identify stylistic or feature-based patterns across the authors using an unsupervised learning approach. Recall that I applied NMF to uncover latent topics from the word frequency data, generating a document-topic matrix representing how strongly each chapter aligns with each topic. To determine which words were important for each topic, I visualized the top 10 words for each component, revealing clear semantic themes. Now using that work, I grouped by author and computed the mean topic weights and visualized them as a heatmap.



This shows that while no topic is exclusive to a single author, there are strong preferences — for instance, ***Milton* scores highest on Topic 1, *Shakespeare* on Topic 2, *Austen* on Topic 3, and *London* on Topic 4**. Since each author has a distinct #1 topic, I am inclined to predict the chapter based on this simple mapping of author to highest topic weight, i.e,

```
mapping = {'Topic1': 'Milton', 'Topic2': 'Shakespeare', 'Topic3': 'Austen', 'Topic4': 'London'}
```

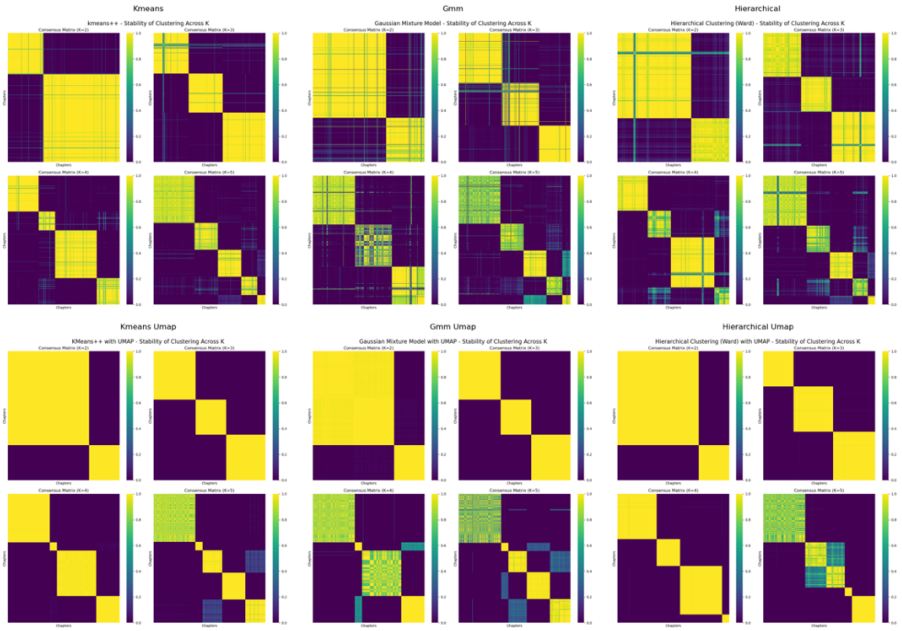
This representation suggests distinct stylistic or grammatical tendencies across authors. To evaluate how informative these patterns are, I split the data into **80% training** and **20% testing**, learned a mapping from dominant topic to author based on training data, and predicted authors on the test set using only the highest-weighted topic. This method correctly identified the author **91.7% of the time**, demonstrating that the unsupervised topic distributions captured meaningful, distinguishable patterns in writing style across authors. Now, we can claim that if the word distribution within a chapter aligns strongly with a topic, we can map that chapter to one of the known authors!

```
top_10_words_per_topic = {'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all', 'their'],
                          'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not', 'and'],
                          'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as', 'it'],
                          'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that', 'an']
}
```

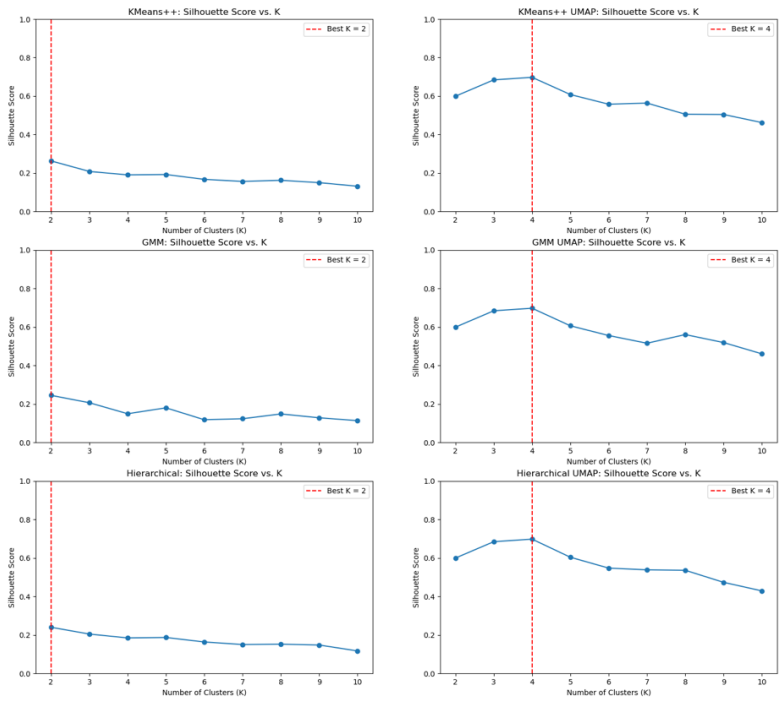
The words that are significant within topics and topics significant across authors provide predictive capabilities!

Validation

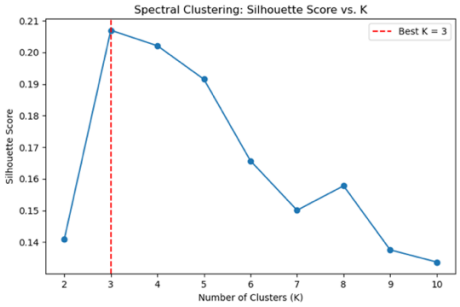
To validate our hyperparameter K without author labels, I used generalizability and stability as metrics.

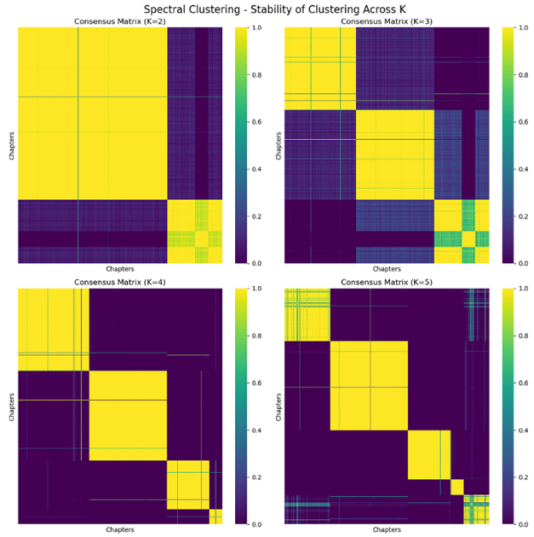


To gain insights to the generalizability, I calculated the silhouette scores across all methods (with and without the dimension reduction UMAP). From the visualization (right) you can see that before applying UMAP the silhouette score was very low across all K indicating but after applying UMAP, the silhouette score drastically increased and peaked at $K = 4$ across Kmeans++, GMM, and Hierarchial Clustering! To evaluate the stability, I used a consensus matrix, visualized above. GMM yielded a poor consensus matrix (likely due to problems in the initialization) but in combination with the silhouette scores we can confidently support the claim that $K = 4$! Furthermore, if we base our hyperparameter validation across all methods the we see a trend in stability score peaking at $K = 4$ and the consensus matrix showing strong stability with 4 clusters.



Spectral clustering can perform poorly under stability analysis because it is highly sensitive to small perturbations in the data or similarity matrix. Unlike centroid-based methods such as K-Means, which rely on explicit distance metrics and generally converge to consistent partitions, spectral clustering depends on the eigen structure of a graph Laplacian derived from data affinities. Even slight changes in the dataset can significantly alter the eigenvectors, leading to different cluster assignments across runs. As a result, traditional stability metrics may underestimate the effectiveness of spectral clustering, not due to poor clustering performance, but due to the method's inherent sensitivity to initialization and graph construction parameters. This is demonstrated below, as the silhouette scores all appeared very low and that it "peaked" at $K = 3$ but very marginally. Inspecting the $K = 3$ consensus matrix makes it clear that the stability for $K = 3$ is very poor and so it makes more sense to select the next highest silhouette score at $K = 4$ (which is essentially the same value as the stability score for $K = 3$). This deduction aligns with all the other methods.





In conclusion, Hierarchial Clustering and Kmeans++ are the most stable and generalize the best for this dataset! The results of analyzing the stability and generalizability lead to a clear hyperparameter of $K = 4$ clusters.

EM Algorithm

The EM algorithm for the Multivariate Gaussian Mixture Model

Start - Initialize

$\mu_k^{(0)} \sim$ Uniform sample from $\{x_1, \dots, x_N\}$, $\forall k \in \{1, \dots, K\}$

$\Sigma_k^{(0)} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$, $\forall k \in \{1, \dots, K\}$

$\pi_k^{(0)} = \frac{1}{K}$, $\forall k \in \{1, \dots, K\}$

E-step

$\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$ where $\mathcal{N}(x_i \mid \mu_j, \Sigma_j)$ is the multivariate Gaussian density $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik} x_i$

Update covariance: $\Sigma_k = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

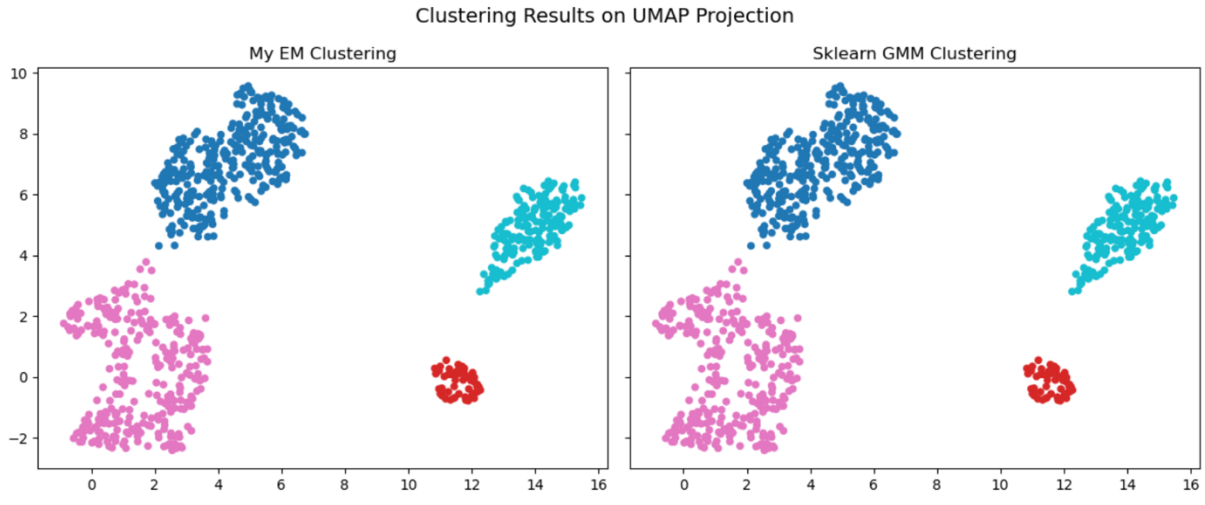
Update mixing weights: $\pi_k = \frac{N_k}{N}$

Running this on the authors algorithm on the 00_authors.csv dataset with the following code (fit_slow uses for loops and then I vectorized it using numpy for speed into fit_fast). See code here (alternatively I have attached a screenshot below).

```python
class GaussianMixtureEM:
    def __init__(self, K, num_iterations, allow_singular=True):
        self.K = K
        self.num_iterations = num_iterations
        self.allow_singular = allow_singular

    def fit_slow(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # E-step
            for i in range(n_rows):
                xi = X.iloc[i].values
                denom = 0
                for k in range(self.K):
                    numerator = pis[k] * multivariate_normal.pdf(xi, mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                    gamma[i, k] = numerator
                    denom += numerator
                gamma[i, :] /= denom

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]
            means = [np.sum([gamma[i, k] * X_array[i, :] for i in range(n_rows)],axis=0) / Nk[k] for k in range(self.K)]
            cov = [np.sum([gamma[i, k] * np.outer(X_array[i, :] - means[k], X_array[i, :] - means[k]) for i in range(n_rows)],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]
            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma
        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}

    def fit_fast(self, X):
        epsilon = 1e-6
        X_array = X.to_numpy()
        n_rows, n_cols = X.shape

        means = X.sample(n=self.K).to_numpy()
        shared_cov = np.cov(X_array, rowvar=False, ddof=1)
        cov = [shared_cov.copy() for _ in range(self.K)]
        pis = [1 / self.K] * self.K
        gamma = np.zeros((n_rows, self.K))

        pis_dict = {'Initial': [pis.copy()]}
        pis_dict.update({f'Iteration_{i}': [] for i in range(self.num_iterations)})

        for iter in range(self.num_iterations):
            # Vectorized E-step
            log_pdf_matrix = np.zeros((n_rows, self.K))
            for k in range(self.K):
                rv = multivariate_normal(mean=means[k], cov=cov[k], allow_singular=self.allow_singular)
                log_pdf_matrix[:, k] = np.log(pis[k] + 1e-12) + rv.logpdf(X_array)

            max_log = np.max(log_pdf_matrix, axis=1, keepdims=True)
            log_gamma = log_pdf_matrix - max_log
            gamma = np.exp(log_gamma)
            gamma /= gamma.sum(axis=1, keepdims=True)

            # M-step
            Nk = [np.sum(gamma[:, j]) for j in range(self.K)]

            means = [np.sum(gamma[:, k][:, np.newaxis] * X_array,axis=0) / Nk[k]for k in range(self.K)]

            cov = [np.sum(gamma[:, k][:, np.newaxis, np.newaxis] *(X_array - means[k])[:, :, np.newaxis] @ (X_array - means[k])[:, :,np.newaxis],axis=0) / Nk[k] + epsilon * np.eye(n_cols) for k in range(self.K)]

            pis = np.array(Nk) / n_rows
            pis_dict[f'Iteration_{iter}'].append(pis.copy())

        self.gamma = gamma

        return {'pis_dict': pis_dict, 'Nk': Nk, 'means': means, 'cov': cov, 'gamma': gamma}
```

The results of my EM-algorithm against Sklearn EM-algorithm for GMM on the authors data set after applying UMAP yielded the same results!

```
Adjusted Rand Index: 1.0000
```



Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!

01_linear_dimension_reductionV2

April 17, 2025

Download the necessary libraries.

Link to data - https://raw.githubusercontent.com/DataSlingers/clustRviz/master/data/authors.rda

Install these if necessary.

[48]:
```
# %pip install scikit-learn --quiet
# %pip install adjustText --quiet
# %pip install umap-learn --quiet
# %pip install wordcloud
```

[49]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.decomposition import FastICA, NMF, KernelPCA, PCA, TruncatedSVD
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE
from sklearn.model_selection import train_test_split
from sklearn.manifold import MDS

## Msc
from adjustText import adjust_text
from itertools import combinations
from wordcloud import WordCloud
```

Load dataset.

[50]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values
df
```

[50]:
```
           Author   a  all  also  an  and  any  are  as  at  …  was  were  \
0          Austen  46   12     0   3   66    9    4  16  13  …   40    11
1          Austen  35   10     0   7   44    4    3  18  16  …   27    13
2          Austen  46    2     0   3   40    1   13  11   9  …   24     6
3          Austen  40    7     0   4   64    3    3  20  13  …   26    10
4          Austen  29    5     0   6   52    5   14  17   6  …   23     5
..            …  ..  …   …  ..  …   …   …  ..  ..  …  …   …
836  Shakespeare  32    4     0   6   33    0    7   8   4  …    0     1
837  Shakespeare  16    5     0   5   49    1    6  10   3  …    1     1
838  Shakespeare  22   15     0   3   48    0    9  10   2  …    4     0
839  Shakespeare  25    4     0   8   59    3    6   7   3  …    3     4
840  Shakespeare  26    4     0   2   62    0    4   7   4  …    5     0

     what  when  which  who  will  with  would  your
0       7     5      6    8     4     9      1     0
1       5     7      7    3     5    14      8     0
2      10     4      6    4     5    15      3     9
3       3     6     10    5     3    22      4     3
4       8     4     13    2     4    21     10     0
..    …   …     …  …   …   …     …   …
836    13     2      3    3    11    17      5    10
837     6     5      6    0    11    20      2     7
838    16     2      2    0    12    15      1    10
839    11     2      2    2    22    23      4     5
840    13     2      5    3    11    19      0     3

[841 rows x 70 columns]
```

[51]:
```
book_id = df['Author']
book_id.value_counts() # 4 different books ; w/ 317 - Austen, ..., 55 - Milton.
```

[51]:
```
Author
Austen         317
London         296
Shakespeare    173
Milton          55
Name: count, dtype: int64
```

- Unsupervised learning: drop columns [‘Authors’] and determine patterns with words across chapters using unsupervised learning methods.
- We will later come back to these labels we dropped to validate our results.
- Note, a row represents a book chapter with each column representing the word counts of key words in that chapter.

1 Linear Methods

1.1 PCA

PCA Theory

Given a centered data matrix $X \in \mathbb{R}^{n \times p}$ with: - n: observations (chapters), - p: features (words),

PCA uses the SVD: $X = U\Sigma V^\top$

- $U\Sigma \rightarrow$ principal component scores (embedding of chapters),
- V (or `pca.components_`) $\rightarrow$ principal axes (directions for words).

No need to transpose the data to get word embeddings.
Using `fit_transform(X)` for chapters, and `components_.T` for words — where both come from the same PCA fit.

Also note that when fitting a PCA in this dataset: **WE SHOULD NOT SCALE DATA SINCE IT IS WORD COUNT!!!**

1.1.1 Observations

[52]:
```
pca = PCA()
X_pca = pca.fit_transform(X)
word_loadings = pca.components_.T

pca_df = pd.DataFrame(X_pca)
cols = [f'PC{j+1}' for j in range(pca_df.shape[1])]
pca_df = pca_df.rename(columns = {i:cols[i] for i in range(pca_df.shape[1])})
pca_df
```

[52]:

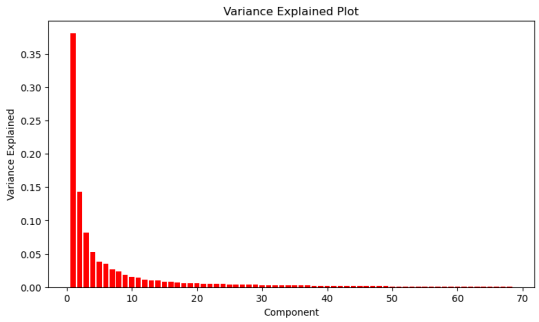
	PC1	PC2	PC3	PC4	PC5	PC6	\
0	-2.265044	43.499301	5.196950	-2.333575	23.359407	22.309224	
1	-2.604648	25.086417	-9.488717	7.748273	19.244916	1.052493	
2	-33.199533	8.667765	-14.833418	3.971572	-6.913595	4.173856	
3	8.098653	21.760546	6.962558	6.683136	15.262020	7.640936	
4	10.031814	6.801164	0.035520	22.731646	10.465248	-5.248171	
..	…	…	…	…	…	…	
836	-64.400961	-28.132705	-21.267908	0.131106	-0.337083	12.068621	
837	-58.313001	-23.417273	4.887866	2.462693	-1.275957	-2.603434	
838	-47.898865	-31.938566	-7.364020	-9.133520	10.442357	14.851150	
839	-39.844905	-29.936659	0.614003	-2.261258	5.007836	14.369278	
840	-34.807687	-38.141056	6.289341	-2.114649	1.435612	13.845645	

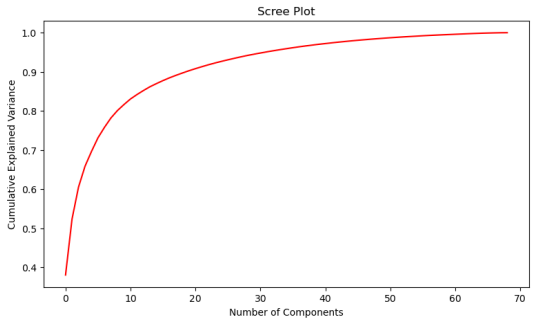
	PC7	PC8	PC9	PC10	…	PC60	PC61	\
0	13.585777	-0.891477	-2.178907	0.062718	…	-2.043904	-3.175850	
1	6.765152	3.650037	-0.746511	-1.753312	…	-2.269775	0.991903	
2	7.094662	-11.728066	1.002140	-4.270366	…	-2.055041	0.864406	
3	11.974431	-5.748289	-1.011280	-4.502115	…	0.590874	0.830834	
4	7.489147	-6.171808	-9.885234	-14.616535	…	1.616164	2.706626	
..	…	…	…	…	…	…	…	

```
836  -5.848328   3.920184 -7.233151   0.239586  … -0.089364 -0.674634
837 -11.452006   4.745220  7.397146  -1.948260  … -0.264725 -1.237238
838 -10.872174  10.228351 -5.141056   3.876664  …  0.938558  1.771321
839  -3.165040   3.620264  1.479972   2.361646  … -0.011889  1.010582
840   3.620534   3.312984 -2.660270   0.521845  …  1.273284  4.486729

         PC62      PC63      PC64      PC65      PC66      PC67      PC68  \
0    1.578725 -2.575981  1.463553 -1.107755 -0.341962  2.639943  3.214014
1    0.013898  1.037391 -1.077602  0.044514 -1.312650  0.800142  0.183083
2   -2.179206  2.564626 -3.672920  0.798756 -0.664101 -1.319190 -0.493611
3   -1.969229  0.937902 -3.603491 -1.244561  1.283435 -0.912942 -0.271062
4   -0.708168 -1.272716 -0.278721 -1.525001  1.746847 -1.004067 -1.272581
..        …         …         …         …         …         …         …
836 -1.165301 -0.470178 -1.348415 -0.240363  0.167233  0.514576 -0.123753
837 -0.170168  1.335993  0.455795  0.942986 -0.266152 -2.256417 -0.182997
838  1.542836 -1.795202  1.242780  2.648138 -0.017259 -2.002290  0.787364
839  1.514307 -0.011713  1.135751  0.803847 -0.028491 -0.803976 -1.307802
840  3.058903  0.497595  0.257034 -0.099739 -0.201420 -0.168980 -0.437193

         PC69
0   -0.594109
1   -0.194564
2    0.165242
3   -0.743841
4   -0.229321
..        …
836 -0.036760
837 -0.021365
838  0.071339
839 -0.026705
840 -0.209892

[841 rows x 69 columns]
```

[53]:
```
fig, ax = plt.subplots(1,2, figsize=(20,5))

ax[0].bar(np.arange(1, pca_df.shape[1]),pca.explained_variance_ratio_[0:pca_df.
 ↪shape[1]-1], color = 'red')
ax[0].set_xlabel('Component')
ax[0].set_ylabel('Variance Explained')
ax[0].set_title('Variance Explained Plot')

ax[1].plot(np.cumsum(pca.explained_variance_ratio_), color = 'red')
ax[1].set_xlabel('Number of Components')
ax[1].set_ylabel('Cumulative Explained Variance')
ax[1].set_title('Scree Plot')
```

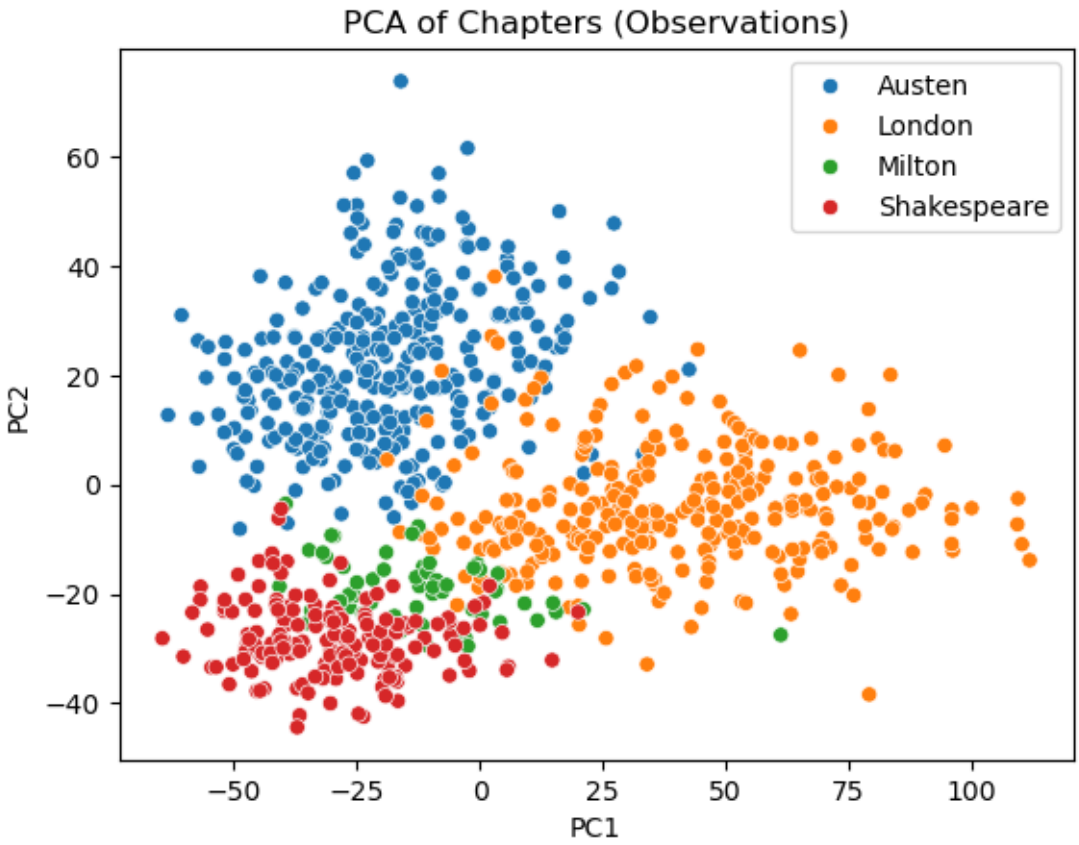
```
plt.savefig('Media/viz/01/01_pca_var_explained_and_screeplot')
plt.show()

print(f'PC1 and PC2 explain {np.sum(pca.explained_variance_ratio_[0:2])*100:.
  ↪3f}% of the variance.')
```






```
PC1 and PC2 explain 52.396% of the variance.
```

[54]:
```
pca_chapters_df = pca_df.loc[:,['PC1','PC2']] # Nested and ordered; extract␣
  ↪first 2 Principal Components
pca_chapters_df['Author'] = df['Author']

sns.scatterplot(data=pca_chapters_df, x='PC1', y='PC2', hue='Author',␣
  ↪palette='tab10')
plt.title('PCA of Chapters (Observations)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.savefig('Media/viz/01/01_pca_chapters')
plt.show()
```



1.1.2 Features

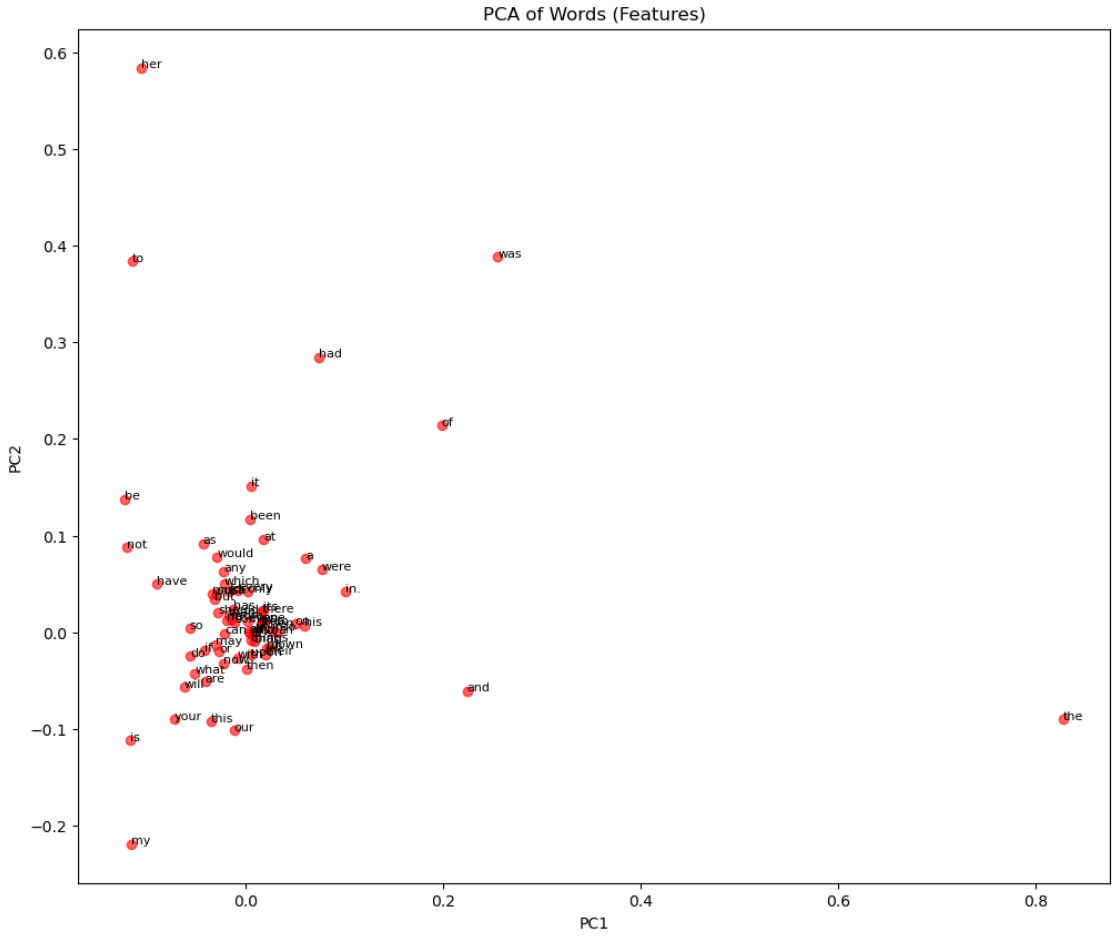
[55]:
```
pca_words_df = pd.DataFrame(word_loadings)
cols = [f'PC{j+1}' for j in range(pca_words_df.shape[1])]
pca_words_df = pca_words_df.rename(columns = {i:cols[i] for i in␣
 ↪range(pca_words_df.shape[1])})
pca_words_df.index = X.columns
pca_words_df
```

[55]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	\
a	0.060676	0.076522	-0.315737	-0.055225	-0.078926	0.263739	0.484181	
all	0.003003	0.000487	0.053015	-0.032618	0.066769	-0.029175	-0.009633	
also	0.007791	-0.000515	0.007859	0.001355	-0.008722	0.000714	0.000660	
an	0.032094	-0.000843	-0.326638	-0.223061	0.188812	-0.163141	0.137205	
and	0.223970	-0.060692	0.667811	-0.081825	-0.015979	0.543313	-0.035569	
…	…	…	…	…	…	…	…	
who	0.003630	-0.001030	0.052789	0.002280	0.002378	-0.013840	-0.007167	
will	-0.062244	-0.056680	-0.001381	0.124291	0.004152	0.024512	-0.127001	
with	-0.008282	-0.026201	0.128355	-0.102281	0.099969	-0.044243	0.033596	
would	-0.029127	0.078012	-0.025317	0.047541	-0.050614	0.034905	-0.062669	

```
your  -0.072387 -0.090066 -0.053952  0.107079 -0.015571  0.009243 -0.028564

             PC8       PC9      PC10  …      PC60      PC61      PC62  \
a      -0.525683  0.264041  0.219213  … -0.005026  0.002127 -0.020590
all     0.042661  0.058567 -0.200501  …  0.007117 -0.043006  0.021078
also    0.007541  0.003267 -0.006968  … -0.002700  0.018310 -0.007202
an      0.060527  0.235075 -0.152501  … -0.012955 -0.026053 -0.005756
and    -0.145092  0.219987 -0.003553  … -0.002065  0.003563  0.004656
…          …        …        …  …        …         …         …
who    -0.006857  0.004959 -0.064097  … -0.030093 -0.166349  0.078814
will   -0.070483 -0.042921  0.024465  …  0.029968  0.013908  0.020761
with   -0.015710  0.041990  0.047793  … -0.006375 -0.009951  0.001977
would  -0.037895 -0.032398 -0.054165  … -0.024179 -0.008643 -0.011250
your    0.019719  0.015591  0.159339  …  0.010928 -0.015083 -0.011472

            PC63      PC64      PC65      PC66      PC67      PC68      PC69
a      -0.020622 -0.005487 -0.001468  0.006522 -0.002964  0.020486  0.001386
all     0.017012 -0.001458  0.008575  0.012527  0.033792 -0.014598  0.001157
also   -0.008381 -0.001825  0.006400 -0.019838 -0.093516  0.056643  0.989992
an      0.020056  0.008385 -0.027075  0.000523  0.005918  0.007497  0.005771
and    -0.002794  0.001242 -0.004503  0.005302 -0.000100  0.000272 -0.005046
…          …         …         …         …         …         …         …
who     0.125654 -0.021114 -0.080313  0.096907  0.061265 -0.000161 -0.017311
will    0.013775  0.006232 -0.059988 -0.017531 -0.003613  0.022391 -0.003819
with   -0.007391  0.009002  0.006421 -0.005216  0.012884  0.008480 -0.000474
would   0.021813  0.022624 -0.031247  0.009858 -0.018687 -0.006684 -0.001789
your   -0.023488 -0.021720  0.023567  0.006809 -0.026835  0.019442 -0.000493

[69 rows x 69 columns]
```

[56]:
```python
plt.figure(figsize=(12, 10))
plt.scatter(pca_words_df['PC1'], pca_words_df['PC2'], alpha=0.6, color = 'red')

for _, row in pca_words_df.iterrows():
    plt.text(row['PC1'], row['PC2'], row.name, fontsize=8)  # ← fixed here
plt.title('PCA of Words (Features)')
plt.xlabel('PC1')
plt.ylabel('PC2');
plt.savefig('Media/viz/01/01_pca_words')
```



This method offers poor visualization and little interpretability. There are other methods that will provide more interpretability on the features such as NMF.

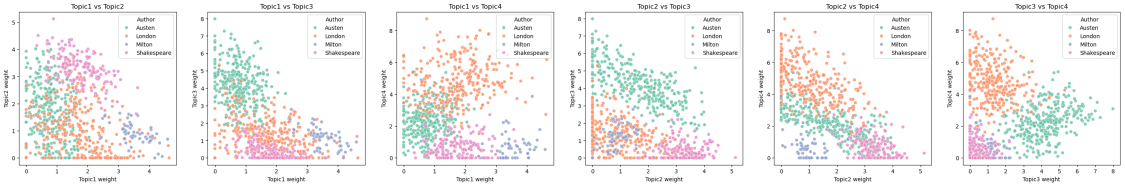
1.2 NMF

NMF may perform strongly here because our word frequencies are non-negative! Furthermore, this method will provide semantic intuition by creating topics.

[57]:
```
k = 4 # hyperparameter -> clusters
model = NMF(n_components=k, init='random', random_state=0, max_iter=500)

# dimensions -> (841 chapters, k topics)
W = model.fit_transform(X)

# dimensions -> (k topics, 69 words)
H = model.components_
```

1.2.1 Observations

[58]:
```
W_df = pd.DataFrame(W, index=X.index, columns=[f'Topic{i+1}' for i in␣
↪range(k)]) # Turn W into a DataFrame with word labels
topic_df = W_df.reset_index(drop=True)
topic_df["Author"] = df["Author"].reset_index(drop=True)
topics = W_df.columns.tolist() # List of topic names
topic_pairs = list(combinations(topics, 2)) # All pairs of topics

n_plots = len(topic_pairs)
fig, axes = plt.subplots(nrows=1, ncols=n_plots, figsize=(6 * n_plots, 5))

if n_plots == 1:
    axes = [axes]

for i, (x_topic, y_topic) in enumerate(topic_pairs):
    ax = axes[i]
    sns.scatterplot(data=topic_df, x=x_topic, y=y_topic, hue="Author",␣
↪palette="Set2", alpha=0.8, ax=ax)
    ax.set_title(f"{x_topic} vs {y_topic}")
    ax.set_xlabel(f"{x_topic} weight")
    ax.set_ylabel(f"{y_topic} weight")
    ax.legend().set_title("Author")

plt.savefig('Media/viz/01/01_nmf_observations')
```



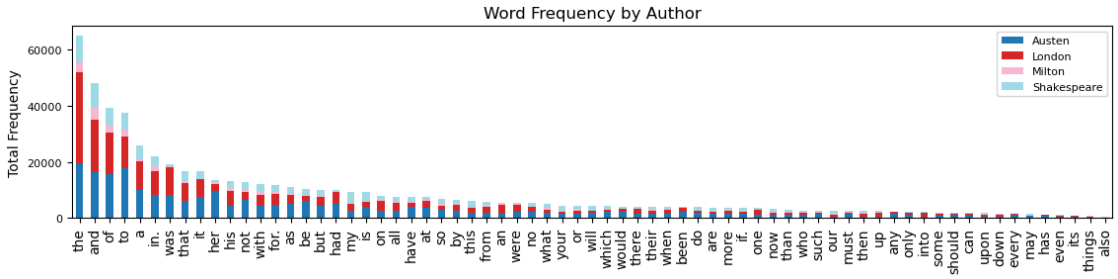
This provides poor interpretability so it is clearly not our first choice when visualizing chapters/observations.

1.2.2 Features

First of all the most simple and easy/inuitive vizualization of the features/words is just a simple total count!

[59]:
```
X_df = X.copy()
X_df['Author'] = authors
author_word_freq = X_df.groupby('Author').sum().T
author_word_freq = author_word_freq.loc[author_word_freq.sum(axis=1).
↪sort_values(ascending=False).index]
```

```
fig, ax = plt.subplots(figsize=(len(author_word_freq)/6, 3))  # adjust based on␣
 ↪# of words
author_word_freq.plot(
    kind='bar',
    stacked=True,
    ax=ax,
    colormap='tab20'
)

plt.title("Word Frequency by Author", fontsize=12)
plt.ylabel("Total Frequency", fontsize=10)
plt.xticks(rotation=90, fontsize=10)
plt.yticks(fontsize=8)
plt.legend(loc='upper right', fontsize=8)
plt.tight_layout()
plt.savefig("Media/viz/01/01_all_word_freq_by_author_stackedbar_condensed.png",␣
 ↪dpi=300)
plt.show()
```



This is where NMF will truly shine.

[60]:
```
n = 10 # Set here -> top words per topic!
```

[61]:
```
word_topic_df = pd.DataFrame(H.T, index=X.columns, columns=[f"Topic{i+1}" for i␣
 ↪in range(H.shape[0])]) # Transpose H to get words as rows, topics as columns
columns = list(word_topic_df.columns)
highest_n_weighted_words_per_topic = {}
for col in columns:
    words = list(word_topic_df[col].sort_values(ascending=False).head(n).index)
    highest_n_weighted_words_per_topic[col] = words
print(highest_n_weighted_words_per_topic)
```

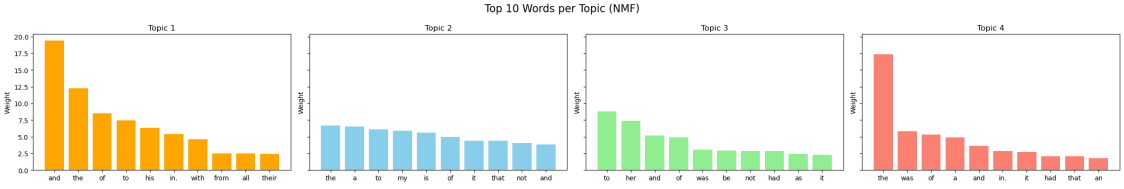
```
{'Topic1': ['and', 'the', 'of', 'to', 'his', 'in.', 'with', 'from', 'all',
'their'], 'Topic2': ['the', 'a', 'to', 'my', 'is', 'of', 'it', 'that', 'not',
'and'], 'Topic3': ['to', 'her', 'and', 'of', 'was', 'be', 'not', 'had', 'as',
'it'], 'Topic4': ['the', 'was', 'of', 'a', 'and', 'in.', 'it', 'had', 'that',
'an']}
```

Below are semantic interpretations of each topic based on the top contributing words:

- **Topic 1 – Structural & Possessive Language**
  Emphasizes grammatical connectors, possession, and sentence scaffolding — likely reflecting narrative structure or formal exposition.
- **Topic 2 – Determiners & Negation**
  Centers on articles, pronouns, and simple negations, suggesting basic sentence formation and assertive language.
- **Topic 3 – Past-Tense Narration**
  Highlights auxiliary verbs and past-tense forms, indicating descriptive or event-driven storytelling.
- **Topic 4 – Temporal & Descriptive Grammar**
  Focuses on narrative tense, articles, and function words often used in unfolding sequences or descriptive prose.

These topics reflect broad grammatical and stylistic features common in text, helping differentiate author styles or narrative structures.

[62]:
```
fig, axes = plt.subplots(1, H.shape[0], figsize=(6 * H.shape[0], 4),␣
 ↪sharey=True)

colors = ['orange', 'skyblue', 'lightgreen', 'salmon']  # or however many␣
 ↪topics you have
for topic_idx, ax in enumerate(axes):
    topic_weights = H[topic_idx]
    top_word_indices = topic_weights.argsort()[::-1][:n]
    top_words = [X.columns[i] for i in top_word_indices]
    top_scores = topic_weights[top_word_indices]
    ax.bar(top_words, top_scores, color=colors[topic_idx % len(colors)])
    ax.set_title(f"Topic {topic_idx + 1}")
    ax.set_ylabel("Weight")

fig.suptitle(f"Top {n} Words per Topic (NMF)", fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/01/01_nmf_top_words_per_topic')
```



Extra Feature Vizuals (not scientific)

[63]:
```
word_freq_dict = X.sum(axis=0).to_dict()

wordcloud = WordCloud(width=1000, height=500, background_color='white').
 ↪generate_from_frequencies(word_freq_dict)

plt.figure(figsize=(15, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of All Features", fontsize=16)
plt.savefig('Media/viz/01/01_word_cloud')
plt.show()
```

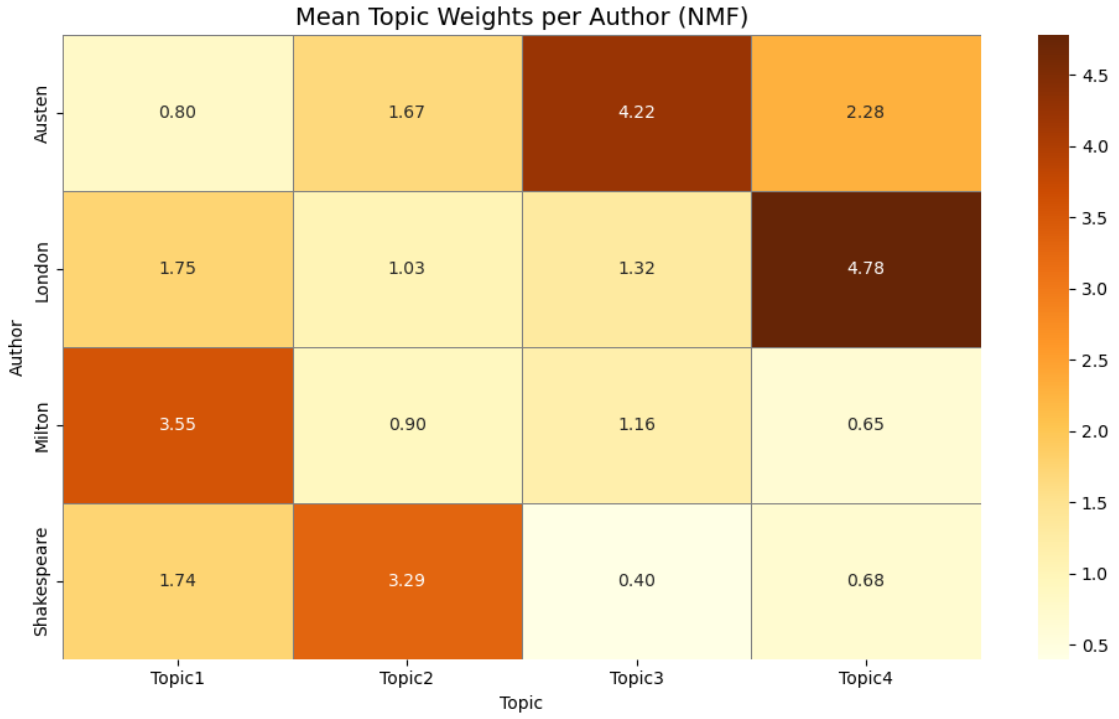
Now let us try and use this to determine which chapters correspond to which topics and predict the author based on the semantics of each topic.

Establish a mapping by grouping by using the labels, grouping by Author and then evaluating the mean topic weights per author! First we should split our data into a training and testing.

[64]:
```
W_df['Authors'] = df['Author']

# Split the data
train_df, test_df = train_test_split(W_df, test_size=0.2, random_state=42,␣
 ↪stratify=W_df['Authors'])

print(f"Training samples: {len(train_df)}")
print(f"Testing samples: {len(test_df)}")
```

```
Training samples: 672
Testing samples: 169
```

[65]:
```
author_topic_means = train_df.groupby('Authors').mean() # Compute mean topic␣
 ↪weights grouped by author
plt.figure(figsize=(10, 6))
sns.heatmap(author_topic_means, annot=True, fmt=".2f", cmap="YlOrBr",␣
 ↪linewidths=0.5, linecolor='gray')
plt.title("Mean Topic Weights per Author (NMF)", fontsize=14)
plt.xlabel("Topic")
plt.ylabel("Author")
plt.tight_layout()
plt.savefig("Media/viz/01/01_nmf_author_topic_heatmap", dpi=300)
```



Mean Topic Weights per Author (NMF)

The heatmap above shows the average topic weights across all chapters written by each author. It is computed by grouping the NMF document-topic matrix (`W`) by author and taking the mean.

Interpretation: - Each cell reflects how strongly a given author tends to express a specific topic. - Higher values indicate that the author frequently uses patterns or structures associated with that topic. - While no topic is exclusive to a single author, we observe distinct preferences: - **Milton** leans heavily on Topic 1 - **Shakespeare** shows strong usage of Topic 2 - **Austen** favors Topic 3 - **London** stands out on Topic 4

This unsupervised representation helps reveal stylistic or grammatical tendencies across authors.

[66]:
```
mapping = {'Topic3': 'Austen', 'Topic4': 'London', 'Topic1': 'Milton', 'Topic2':
 ↪ 'Shakespeare'}
```

Now lets evaluate this mapping on the test set!

[67]:
```
test_df['Highest Weighted Topic'] = test_df.iloc[:, :4].idxmax(axis=1).values
test_df['Likely Author'] = test_df['Highest Weighted Topic'].apply(lambda x:␣
 ↪mapping[x])
correct_proportion = (test_df['Likely Author'] == test_df['Authors']).mean()
print(f'This method was correct {correct_proportion*100:.4f}% of the time')
```

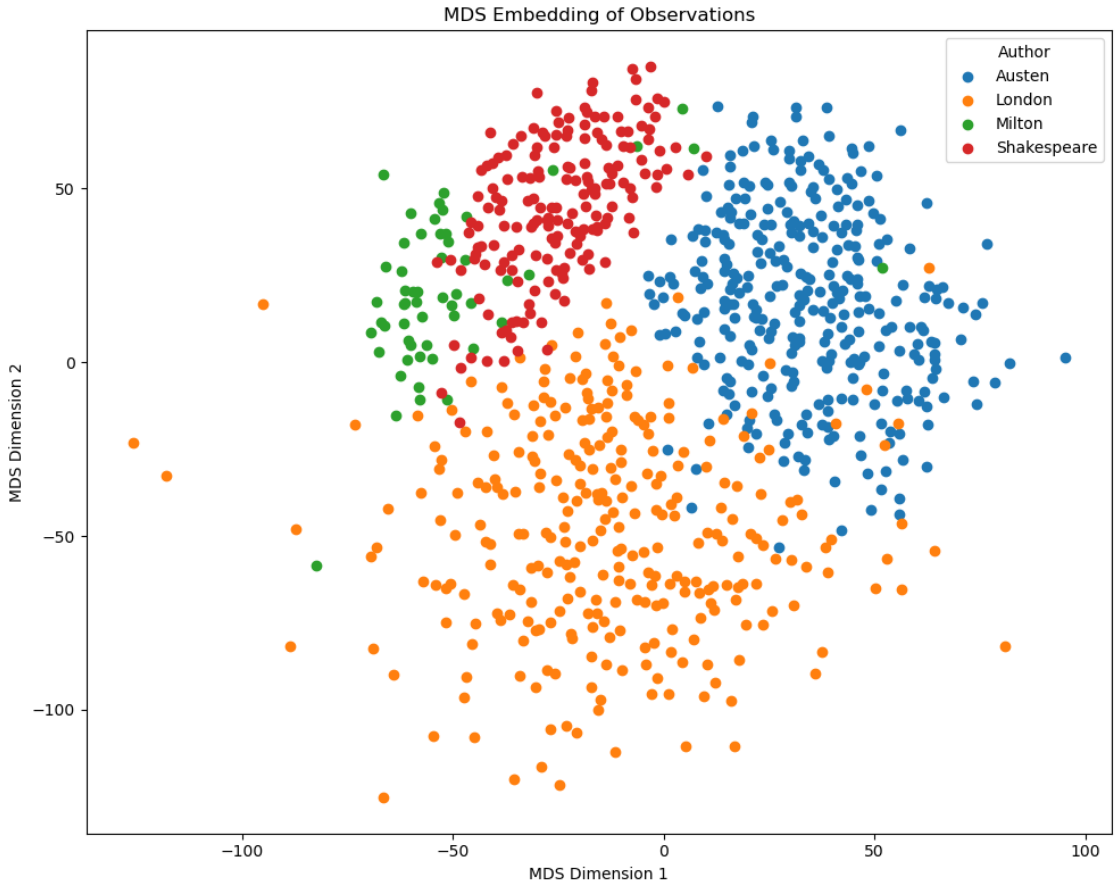
```
This method was correct 91.7160% of the time
```

This yields pretty good results!!

1.3 MDS

1.3.1 Observations

[68]:
```
model = MDS(n_components=2, random_state=42)
X_mds = model.fit_transform(X)

unique_authors = sorted(set(authors))

# Plot
plt.figure(figsize=(10, 8))
for i, author in enumerate(unique_authors):
    mask = authors == author
    plt.scatter(X_mds[mask, 0], X_mds[mask, 1], label=author)

plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.title("MDS Embedding of Observations")
plt.legend(title="Author", loc="best")
plt.tight_layout()
plt.savefig("Media/viz/01/01_mds_observation_viz")
plt.show()
```



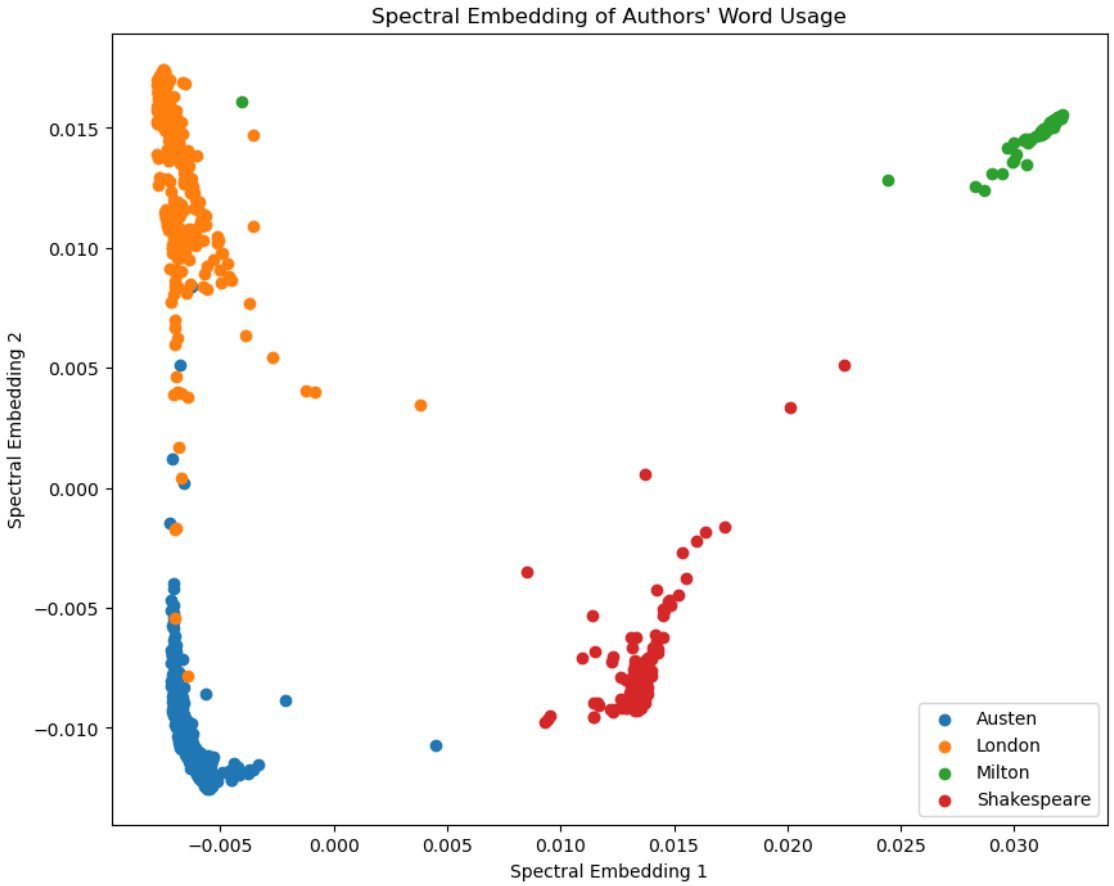
2 Non-Linear Methods

2.1 Spectral Embeddings

2.1.1 Observations

[69]:
```
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10)
X_spec = spectral.fit_transform(X)  # shape: (n_samples, 2)
spectral_df = pd.DataFrame(X_spec, columns=['Dim1', 'Dim2'])

plt.figure(figsize=(10, 8))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
plt.xlabel("Spectral Embedding 1")
plt.ylabel("Spectral Embedding 2")
plt.title("Spectral Embedding of Authors' Word Usage")
plt.legend(loc="lower right")
```

```
plt.savefig('Media/viz/01/01_spectal_obs_viz')
plt.show()
```



2.1.2 Features

Unlike PCA and NMF there are no components we can use to visualize the features in Spectral Embedding. However, we can apply a transpose to our data and then reapply Spectral Embedding.

This procedure will yield results where:

- Two words will be close in the embedding if they tend to appear in the same chapters.
- This gives you a co-occurrence-like structure, driven by chapter usage.
- This is conceptually similar to Latent Semantic Analysis, just via graph-based distance instead of SVD.

[70]:
```
X_transpose = X.T
X_transpose = X_transpose.rename(columns = {i:f'Chapter{i}' for i in range(df.
  ↪shape[0])})
X_transpose
```

[70]:

	Chapter0	Chapter1	Chapter2	Chapter3	Chapter4	Chapter5	Chapter6	\
a	46	35	46	40	29	27	34	
all	12	10	2	7	5	8	8	
also	0	0	0	0	0	0	0	
an	3	7	3	4	6	3	15	
and	66	44	40	64	52	42	44	
…	…	…	…	…	…	…	…	
who	8	3	4	5	2	6	4	
will	4	5	5	3	4	3	9	
with	9	14	15	22	21	18	11	
would	1	8	3	4	10	4	6	
your	0	0	9	3	0	5	4	

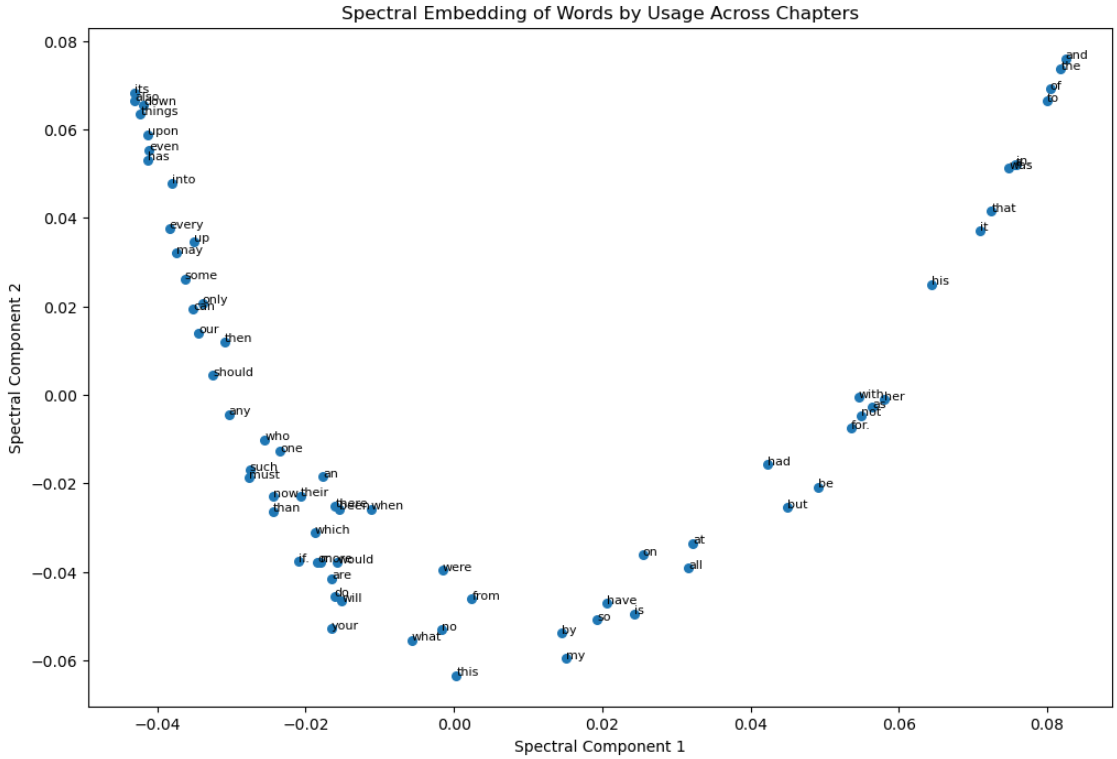
	Chapter7	Chapter8	Chapter9	…	Chapter831	Chapter832	Chapter833	\
a	38	34	54	…	46	48	39	
all	6	12	8	…	4	2	5	
also	1	0	0	…	0	0	0	
an	2	5	6	…	3	9	10	
and	67	50	44	…	43	45	38	
…	…	…	…	…	…	…	…	
who	6	1	3	…	1	0	2	
will	7	2	5	…	7	10	8	
with	15	13	15	…	18	11	26	
would	3	12	6	…	2	6	2	
your	5	5	2	…	3	11	16	

	Chapter834	Chapter835	Chapter836	Chapter837	Chapter838	Chapter839	\
a	22	28	32	16	22	25	
all	13	7	4	5	15	4	
also	0	0	0	0	0	0	
an	5	7	6	5	3	8	
and	47	45	33	49	48	59	
…	…	…	…	…	…	…	
who	4	2	3	0	0	2	
will	9	7	11	11	12	22	
with	12	8	17	20	15	23	
would	6	3	5	2	1	4	
your	7	7	10	7	10	5	

	Chapter840
a	26
all	4
also	0
an	2
and	62
…	…
who	3

```
will           11
with           19
would           0
your            3

[69 rows x 841 columns]
```

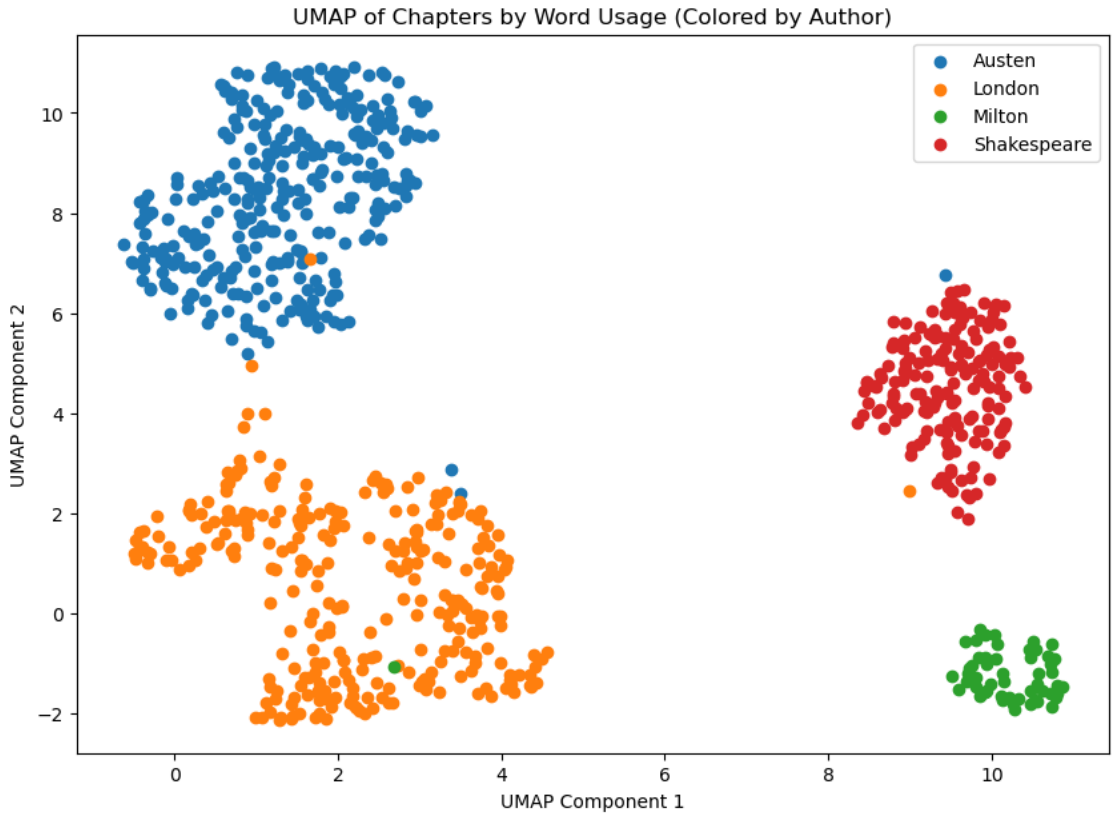
[71]:
```
X_words = X_transpose.to_numpy()
spectral = SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=0)
X_words_spec = spectral.fit_transform(X_words)

plt.figure(figsize=(12, 8))
plt.scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
plt.title("Spectral Embedding of Words by Usage Across Chapters")
plt.xlabel("Spectral Component 1")
plt.ylabel("Spectral Component 2")
plt.savefig('Media/viz/01/01_spectral_feature_viz')
plt.show()
```



2.2 UMAP

2.2.1 Observations

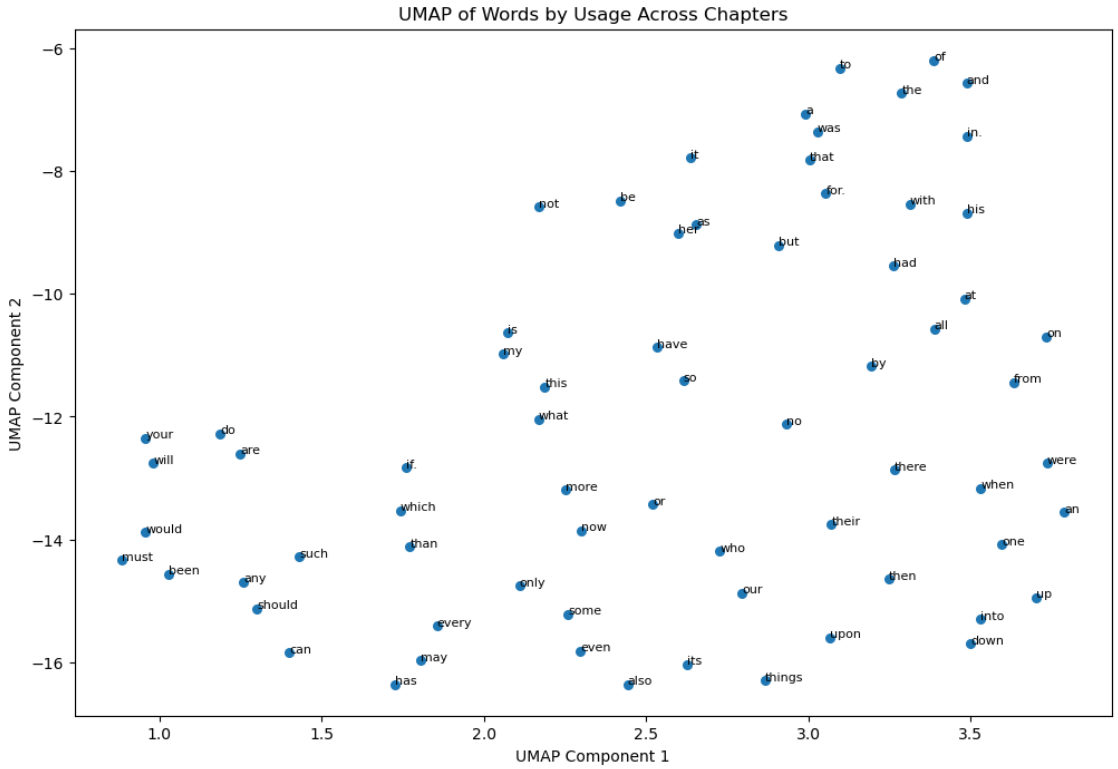
[84]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_umap = umap_model.fit_transform(X.to_numpy())

plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_umap_obs_viz')
plt.show()

plt.figure(figsize=(10, 7))
plt.scatter(X_umap[:,0],X_umap[:,1], color = 'purple')
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.title("UMAP")
plt.savefig('Media/viz/01/01_umap_viz_unlabeled');
plt.close()
```



UMAP does a very good job at creating seperable clusters! This visualization appears to be the best so far when observing observations (chapters)!

2.2.2 Features

[73]:
```
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, n_jobs=-1) # Fit UMAP
X_words_umap = umap_model.fit_transform(X_transpose.to_numpy())

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
plt.title("UMAP of Words by Usage Across Chapters")
plt.xlabel("UMAP Component 1")
plt.ylabel("UMAP Component 2")
plt.savefig('Media/viz/01/01_umap_feature_viz')
plt.show()
```

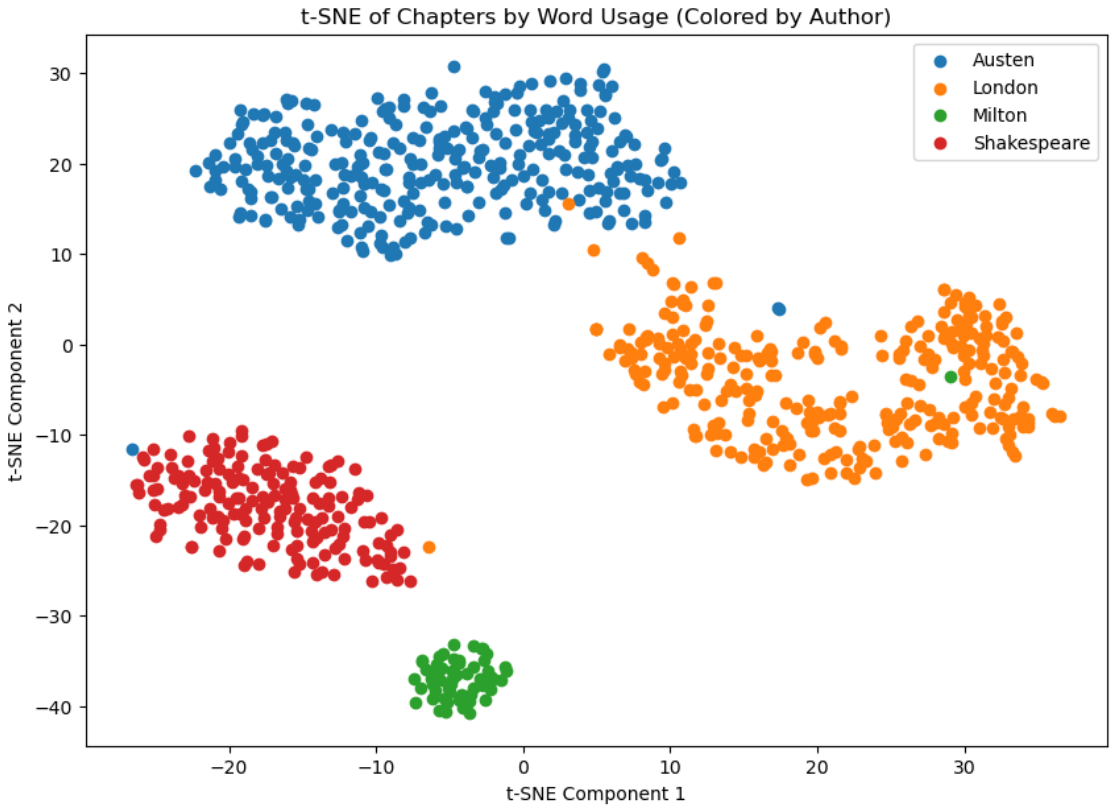


Not very interpretable/helpful.

2.3 tSNE

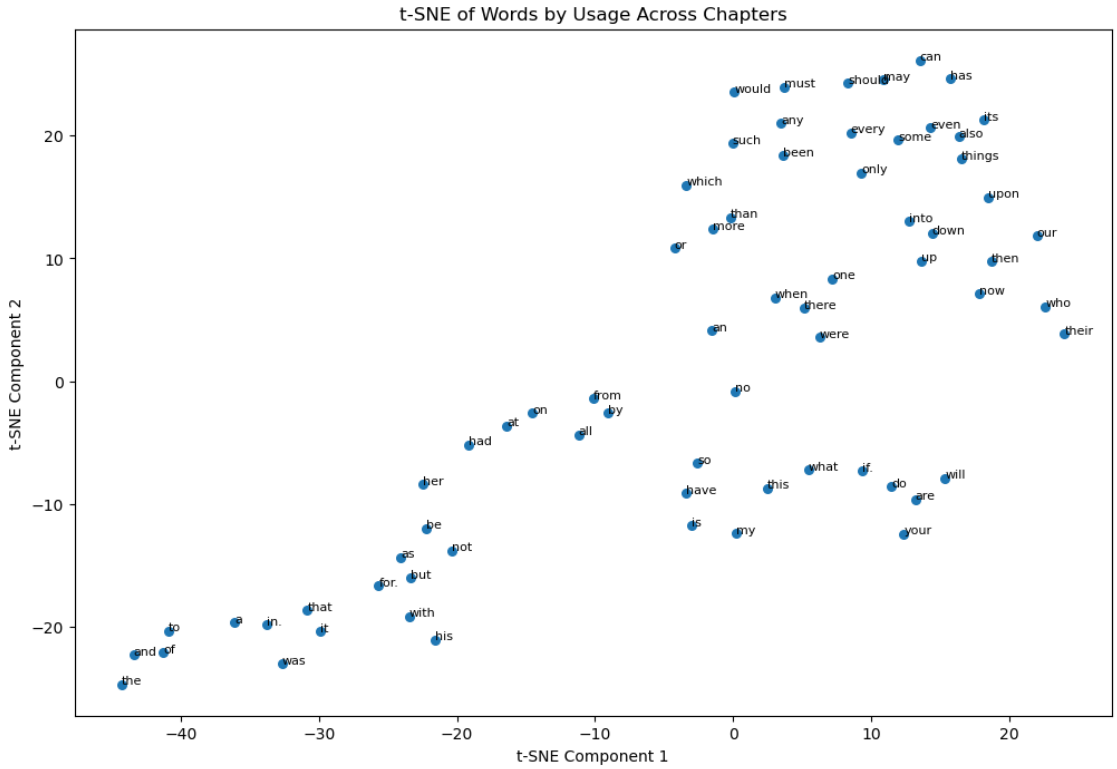
2.3.1 Observations

[74]:
```
tsne = TSNE(n_components=2, perplexity=30, learning_rate='auto') # Fit t-SNE
X_tsne = tsne.fit_transform(X.to_numpy())

# Plot chapters with author labels
plt.figure(figsize=(10, 7))
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    plt.scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)

plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.title("t-SNE of Chapters by Word Usage (Colored by Author)")
plt.legend(loc="upper right")
plt.savefig('Media/viz/01/01_tsne_obs_viz')
plt.show()
```



2.3.2 Features

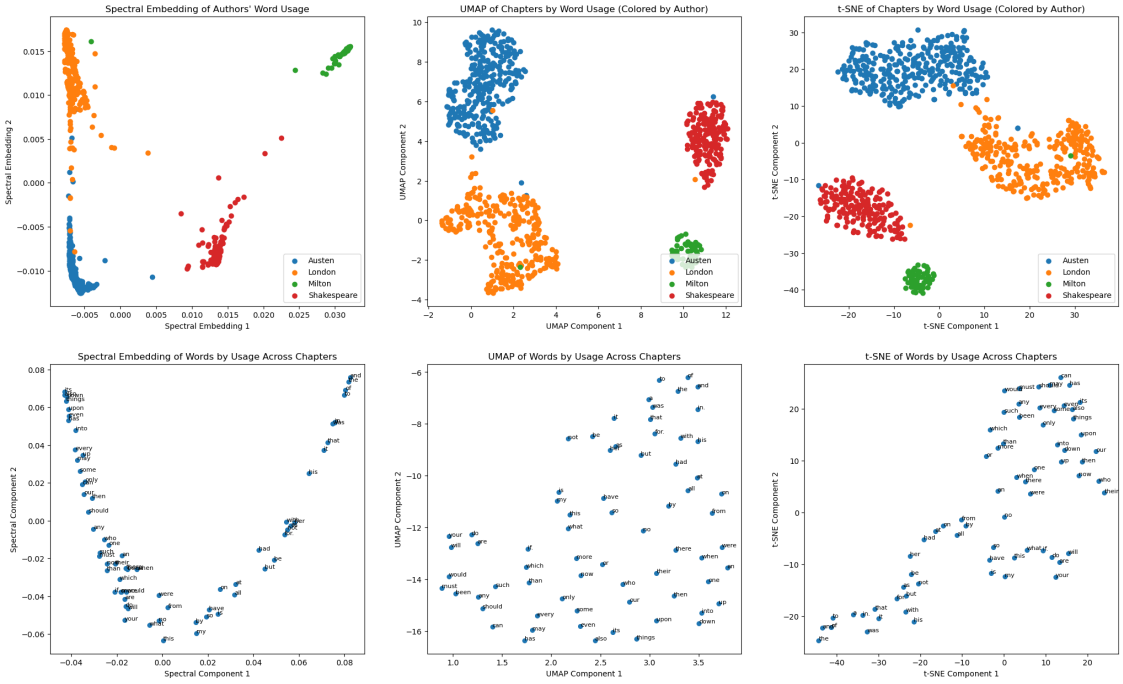
[75]:
```
tsne = TSNE(n_components=2, perplexity=5, learning_rate='auto') # Fit t-SNE
X_words_tsne = tsne.fit_transform(X_words)

# Plot w/ word labels
plt.figure(figsize=(12, 8))
plt.scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    plt.text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)

plt.title("t-SNE of Words by Usage Across Chapters")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.savefig('Media/viz/01/01_tsne_feature_viz')
plt.show()
```



3 Combined Visualizations

[76]:
```
fig, ax = plt.subplots(2, 3, figsize=(25, 15), sharey=False)  # 1 row, 3 columns

# Spectral Embedding
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,0].scatter(X_spec[mask, 0], X_spec[mask, 1], label=author)
ax[0,0].set_xlabel("Spectral Embedding 1")
ax[0,0].set_ylabel("Spectral Embedding 2")
ax[0,0].set_title("Spectral Embedding of Authors' Word Usage")
ax[0,0].legend(loc="lower right")

ax[1,0].scatter(X_words_spec[:, 0], X_words_spec[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,0].text(X_words_spec[i, 0], X_words_spec[i, 1], word, fontsize=8)
ax[1,0].set_title("Spectral Embedding of Words by Usage Across Chapters")
ax[1,0].set_xlabel("Spectral Component 1")
ax[1,0].set_ylabel("Spectral Component 2")

# UMAP
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
```

```
    mask = (authors == author)
    ax[0,1].scatter(X_umap[mask, 0], X_umap[mask, 1], label=author)
ax[0,1].set_xlabel("UMAP Component 1")
ax[0,1].set_ylabel("UMAP Component 2")
ax[0,1].set_title("UMAP of Chapters by Word Usage (Colored by Author)")
ax[0,1].legend(loc="lower right")

ax[1,1].scatter(X_words_umap[:, 0], X_words_umap[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,1].text(X_words_umap[i, 0], X_words_umap[i, 1], word, fontsize=8)
ax[1,1].set_title("UMAP of Words by Usage Across Chapters")
ax[1,1].set_xlabel("UMAP Component 1")
ax[1,1].set_ylabel("UMAP Component 2")

# tSNE
for author in ['Austen', 'London', 'Milton', 'Shakespeare']:
    mask = (authors == author)
    ax[0,2].scatter(X_tsne[mask, 0], X_tsne[mask, 1], label=author)
ax[0,2].set_xlabel("t-SNE Component 1")
ax[0,2].set_ylabel("t-SNE Component 2")
ax[0,2].set_title("t-SNE of Chapters by Word Usage (Colored by Author)")
ax[0,2].legend(loc="lower right")

ax[1,2].scatter(X_words_tsne[:, 0], X_words_tsne[:, 1], s=30)
for i, word in enumerate(X_transpose.index):
    ax[1,2].text(X_words_tsne[i, 0], X_words_tsne[i, 1], word, fontsize=8)
ax[1,2].set_title("t-SNE of Words by Usage Across Chapters")
ax[1,2].set_xlabel("t-SNE Component 1")
ax[1,2].set_ylabel("t-SNE Component 2")

plt.savefig('Media/viz/01/01_nonlinear_viz')
plt.show()
```



UMAP offers the best balance between global and local structure, preserves neighborhood quality, and gives interpretable groupings without as much distortion as t-SNE.

Clusters of words in the UMAP embedding reflect similar usage patterns across chapters. Since word usage is shaped by topic and syntax, these local neighborhoods can be interpreted as reflecting semantic or grammatical similarity. While UMAP does not preserve global distances, local groupings are meaningful.

02_visualizations

April 17, 2025

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
import umap
from sklearn.preprocessing import StandardScaler, LabelEncoder
from scipy.spatial.distance import pdist, squareform
from sklearn.manifold import SpectralEmbedding, TSNE, MDS
from sklearn.decomposition import PCA, NMF, FastICA, TruncatedSVD
from sklearn.metrics import pairwise_distances

## Msc
from adjustText import adjust_text
from itertools import combinations
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df["Author"].values

X_words = X.T.values
feature_names = X.columns
```

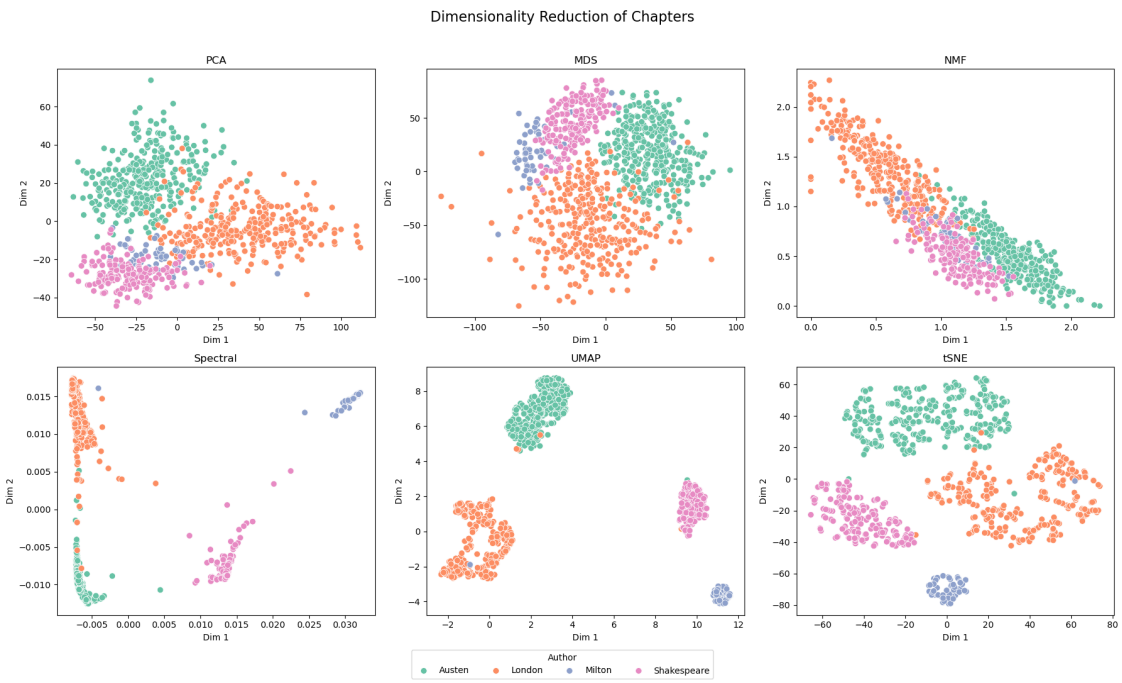
1 Comparison

1.1 Observations

[9]:
```
methods = {
    "PCA": PCA(n_components=2),
    "MDS": MDS(n_components=2, random_state=42),
```

```
    "NMF": NMF(n_components=2, random_state=42, max_iter=1000),
    "Spectral": SpectralEmbedding(n_components=2, affinity="nearest_neighbors",␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

palette = { "Austen": "#66c2a5", "London": "#fc8d62", "Milton": "#8da0cb",␣
 ↪"Shakespeare": "#e78ac3"} # Set consistent colors for each label

embeddings = {name: model.fit_transform(X) for name, model in methods.items()}

fig, axs = plt.subplots(2, int(len(embeddings)/2), figsize=(len(methods)*3, 10))

for ax, (name, embed) in zip(axs.flatten(), embeddings.items()):
    sns.scatterplot(x=embed[:, 0], y=embed[:, 1], hue=authors, palette=palette,␣
 ↪s=50, ax=ax)
    ax.set_title(name)
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")
    ax.legend().remove()

handles, labels = ax.get_legend_handles_labels()
fig.legend(handles, labels, title="Author", loc='lower center', ncol=4,␣
 ↪bbox_to_anchor=(0.5, -0.05))
fig.suptitle("Dimensionality Reduction of Chapters", fontsize=16, y=1.02)

plt.tight_layout()
plt.savefig("Media/viz/02/02_across_methods_obs_viz.png", bbox_inches="tight",␣
 ↪dpi=300)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```

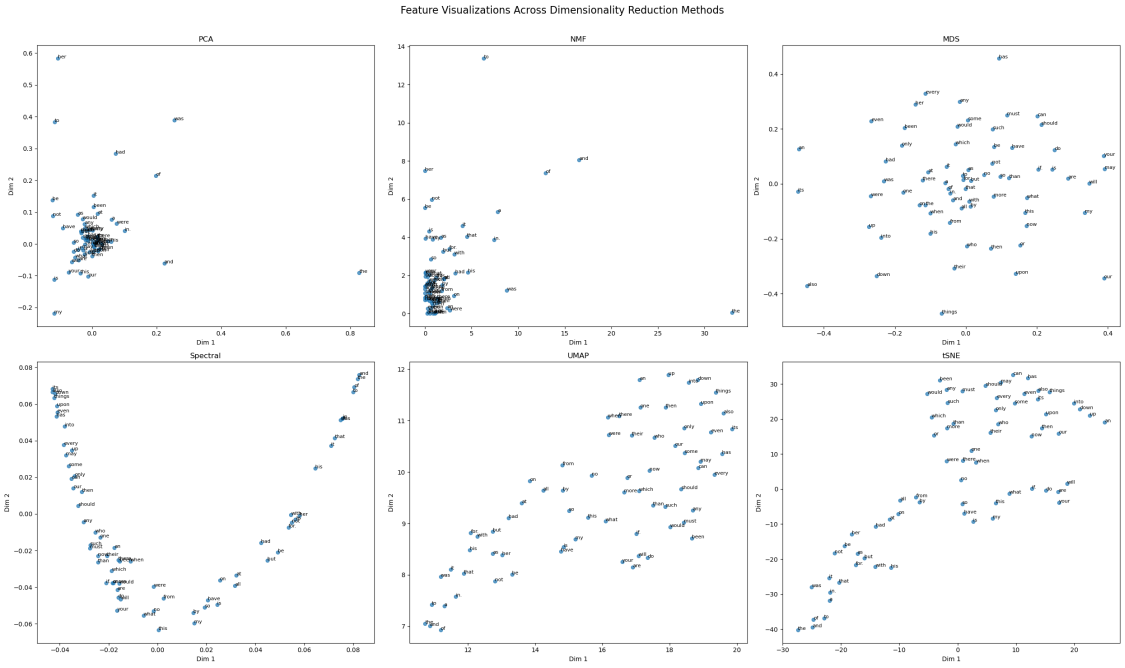


1.2 Features

[5]:
```
methods = {
    "PCA": PCA(n_components=2).fit(X),
    "NMF": NMF(n_components=2, init='random', random_state=42, max_iter=1000).
 ↪fit(X),
    "MDS": MDS(n_components=2, dissimilarity='precomputed', random_state=42),
    "Spectral": SpectralEmbedding(n_components=2, affinity='nearest_neighbors',␣
 ↪n_neighbors=10, random_state=42),
    "UMAP": umap.UMAP(n_components=2, n_neighbors=10, min_dist=0.3,␣
 ↪random_state=42),
    "tSNE": TSNE(n_components=2, perplexity=5, learning_rate='auto',␣
 ↪random_state=42)
}

feature_embeddings = {
    "PCA": methods["PCA"].components_.T,
    "NMF": methods["NMF"].components_.T,
    "MDS": methods["MDS"].fit_transform(pairwise_distances(X_words,␣
 ↪metric="cosine")),
    "Spectral": methods["Spectral"].fit_transform(X_words),
    "UMAP": methods["UMAP"].fit_transform(X_words),
    "tSNE": methods["tSNE"].fit_transform(X_words)
}
```

```
fig, axs = plt.subplots(2, 3, figsize=(25, 15))
axs = axs.flatten()

for ax, (name, coords) in zip(axs, feature_embeddings.items()):
    ax.scatter(coords[:, 0], coords[:, 1], s=30, alpha=0.7)
    for i, word in enumerate(feature_names):
        ax.text(coords[i, 0], coords[i, 1], word, fontsize=8)
    ax.set_title(f"{name}")
    ax.set_xlabel("Dim 1")
    ax.set_ylabel("Dim 2")

plt.suptitle("Feature Visualizations Across Dimensionality Reduction Methods",␣
 ↪fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.97])
plt.savefig("Media/viz/02/02_feature_comparison_all_methods")
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/umap/umap_.py:1952: UserWarning:
n_jobs value 1 overridden to 1 by setting random_state. Use no seed for
parallelism.
  warn(
```



03_biclustering_viz

April 17, 2025

Install these if necessary.

[1]:
```
# %pip install fastcluster --quiet
```

Load necessary libraries.

[2]:
```
## Basics
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

## ML Packages
from sklearn.cluster import SpectralBiclustering, SpectralCoclustering
from sklearn.feature_selection import VarianceThreshold

## Msc
from PIL import Image
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
  ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
authors = df['Author'].values  # n_samples-length array
```

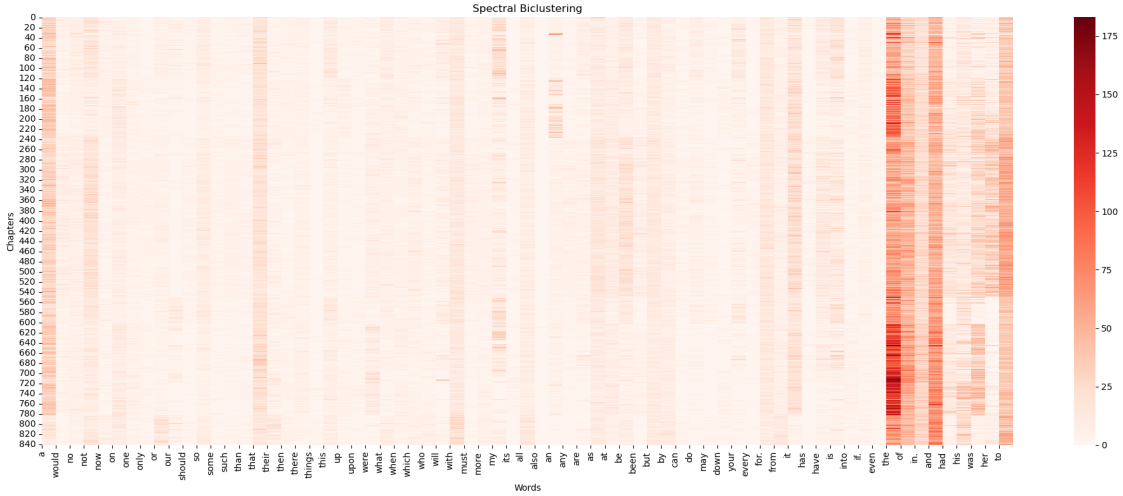
1 Biclustering

1.1 Spectral Biclustering

Let us fit and visualize both the chapters and words using Spectral Biclustering!

[4]:
```
### Fit biclustering model
model = SpectralBiclustering(n_clusters=4, method='log', random_state=0)
model.fit(X)

### Reorder the data
row_order = np.argsort(model.row_labels_)
```

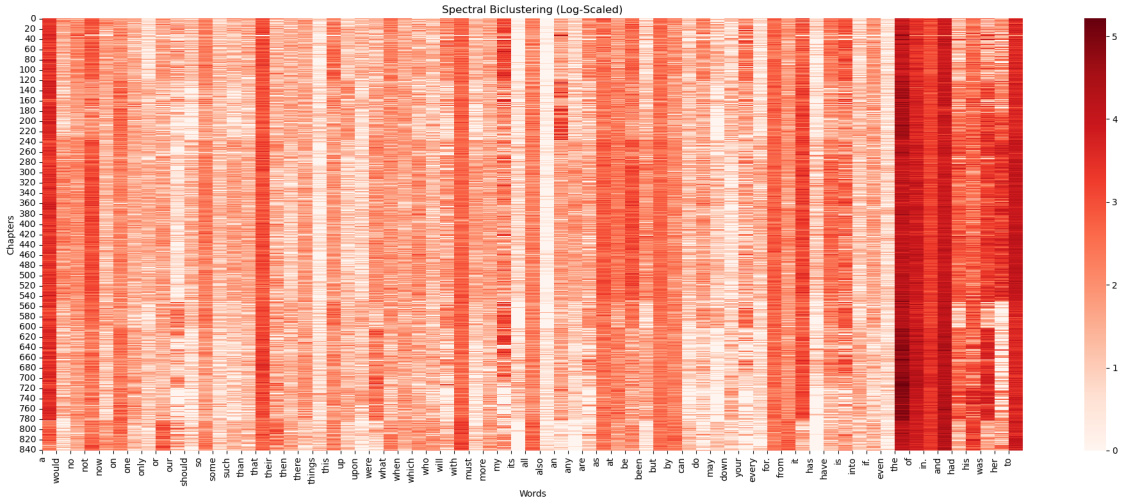
```
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Visualization
plt.figure(figsize=(20, 8))  # wider figurea
sns.heatmap(fit_data, cmap="Reds", cbar=True)
plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/spectral_biclustering_viz')
plt.show()
```



Since words like “the”, “and”, “to”, … are used frequently across chapters on a global level there is not much insights to be deduced here. We want to look for at words with specific chapters having bands. This will help us to differentiate a word that is frequently used across a cluster of chapters indicating stylistic writing by the author! This will help with classifying these chapters to their respective authors!! There is a small problem that the common words are very dark so it is harder to see a contrast in scale so we will convert the scale to log scale to see a larger contrast among the less frequently used words.

1.1.1 Log-Scale (Improve Visualization)

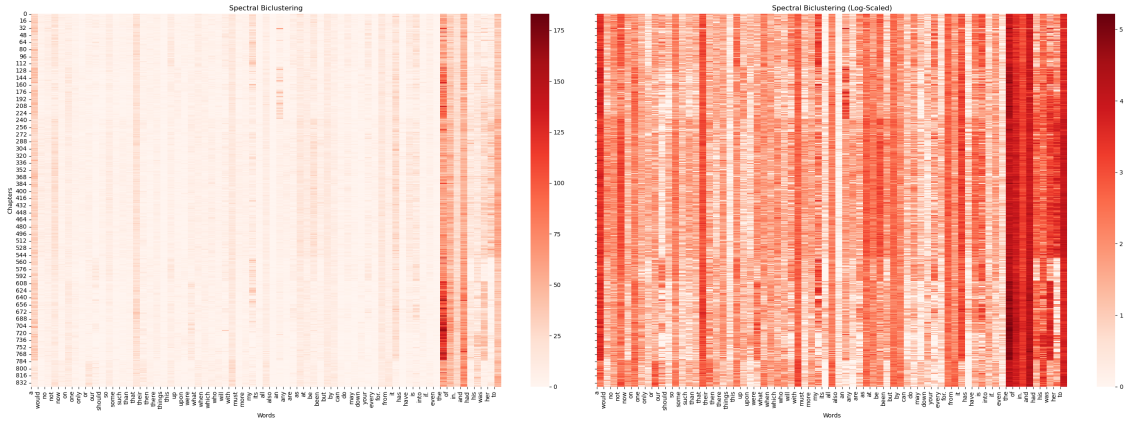
[5]:
```
### Reorder the data
row_order = np.argsort(model.row_labels_)
col_order = np.argsort(model.column_labels_)
fit_data = X.values[row_order][:, col_order]

### Get reordered word labels for x-axis
word_labels = X.columns[col_order]

### Apply log scale to compress high-frequency words like "the"
log_data = np.log1p(fit_data)  # log(1 + x) to avoid log(0)

### Plot the heatmap
plt.figure(figsize=(20, 8))
sns.heatmap(log_data, cmap="Reds", cbar=True)

plt.xticks(ticks=np.arange(len(word_labels)), labels=word_labels, rotation=90)
plt.title("Spectral Biclustering (Log-Scaled)")
plt.xlabel("Words")
plt.ylabel("Chapters")
plt.tight_layout()
plt.savefig('Media/viz/03/single_plots/log_scale_spectral_biclustering_viz')
plt.show()
```



1.1.2 Final Visualization

Plot side by side for visualization/comparison purposes.

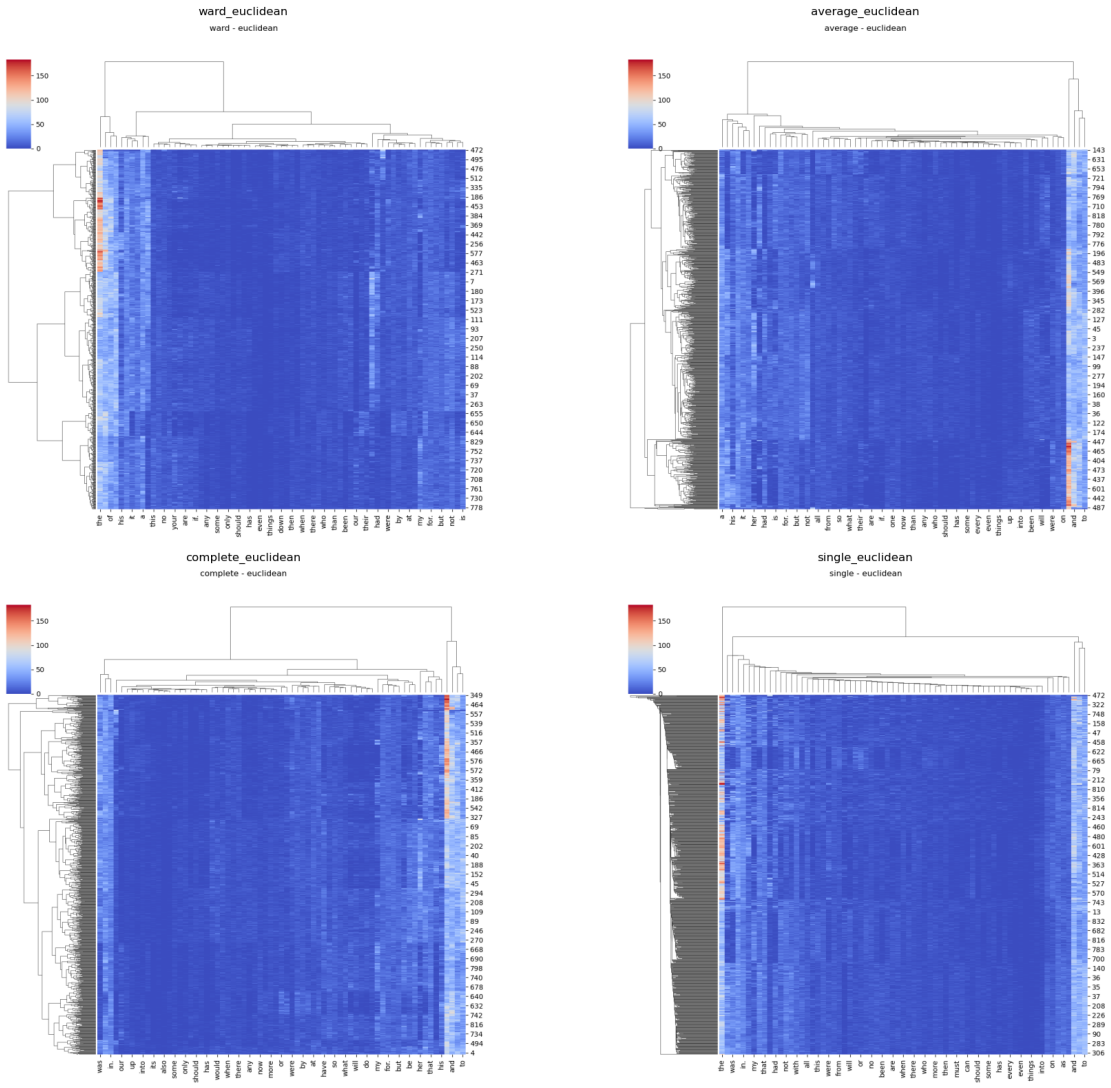
[6]:
```
# Plot side-by-side heatmaps
fig, axes = plt.subplots(1, 2, figsize=(28, 10), sharey=True)

# Original
sns.heatmap(fit_data, cmap="Reds", ax=axes[0], cbar=True)
axes[0].set_xticks(np.arange(len(word_labels)))
axes[0].set_xticklabels(word_labels, rotation=90)
axes[0].set_title("Spectral Biclustering")
axes[0].set_xlabel("Words")
axes[0].set_ylabel("Chapters")

# Log-scaled
sns.heatmap(log_data, cmap="Reds", ax=axes[1], cbar=True)
axes[1].set_xticks(np.arange(len(word_labels)))
axes[1].set_xticklabels(word_labels, rotation=90)
axes[1].set_title("Spectral Biclustering (Log-Scaled)")
axes[1].set_xlabel("Words")
axes[1].set_ylabel("")

plt.tight_layout()
plt.savefig('Media/viz/03/03_spectral_biclustering_heatmaps')
plt.show()
```



1.2 Hierarchial Biclustering

[7]:
```
# Define top 4 method-metric combinations (all Euclidean)
combinations = [
    ('ward', 'euclidean'),
    ('average', 'euclidean'),
    ('complete', 'euclidean'),
    ('single', 'euclidean'),
]
```

[8]:
```
img_paths = []
for method, metric in combinations:
    g = sns.clustermap(X, method=method, metric=metric, cmap='coolwarm')
    g.fig.suptitle(f'{method} - {metric}', y=1.05)
    img_path = f'Media/viz/03/single_plots/clustermap_{method}_{metric}.png'
    g.savefig(img_path, bbox_inches='tight')
    plt.close(g.fig)
    img_paths.append(img_path)

fig, axes = plt.subplots(2, 2, figsize=(20, 16))
for ax, path in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(path.split('/')[-1].replace('clustermap_', '').replace('.png',␣
 ↪''))
    ax.axis('off')

plt.tight_layout()
plt.savefig('Media/viz/03/03_hierarchial_biclustering_heatmaps')
plt.show()
```



This clustermap visualizes word frequency across chapters using hierarchical clustering on both rows (chapters) and columns (words). Chapters with similar word usage patterns are grouped together, as are words that tend to co-occur across the same sets of chapters. The dendrograms reveal hidden structure in the data, highlighting natural groupings based on stylistic similarity without any labels. This helps identify clusters of similar chapters and potentially stylistically meaningful groups of words.

However, we can see that the interpretation from these heatmaps is tricky so lets try and reduce the features with low variance patterns. So we will do some initial feature filtering, then perform the same heatmap visualizations!

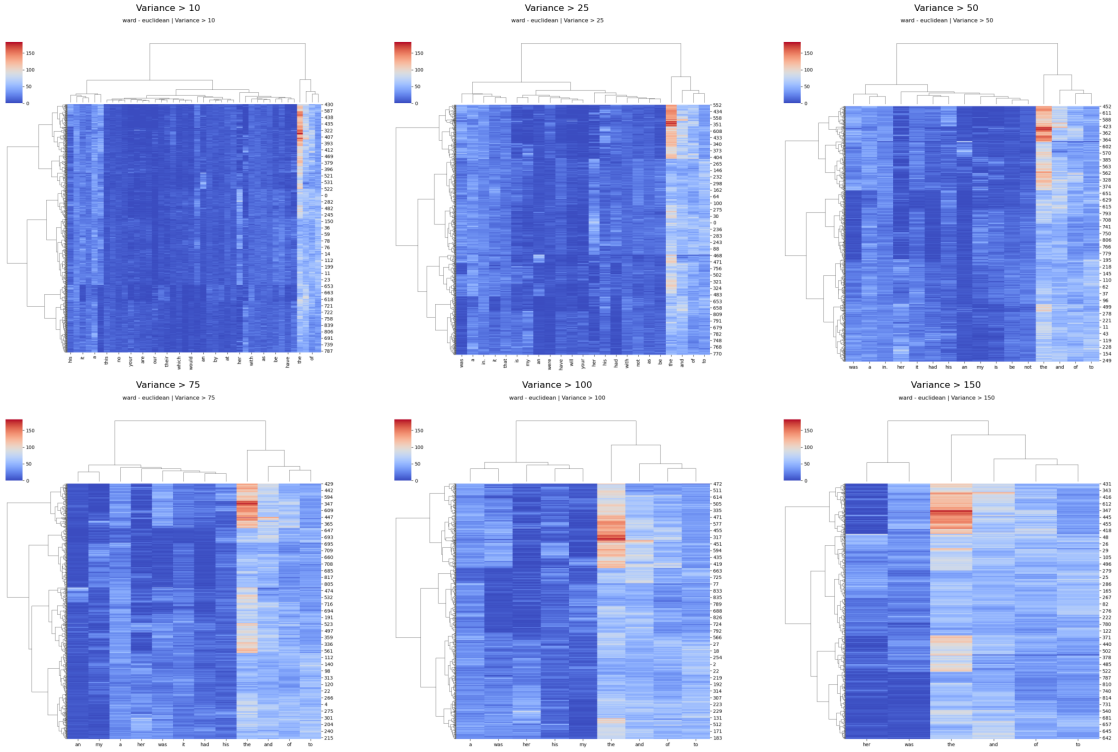
[9]:
```
X_filtered_dict = {'var_threshold':[], 'X_filtered_df':[]}
thresholds_list = [10, 25, 50, 75, 100, 150]
combinations = [('ward', 'euclidean')]
```

```
for alpha in thresholds_list: # Loop over variance filter threshold as alpha␣
 ↪increase, more features are dropped!
    selector = VarianceThreshold(threshold=alpha)
    X_reduced = selector.fit_transform(X)
    selected_columns = X.columns[selector.get_support()] # Get the column names␣
 ↪that passed this variance threshold
    X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)
    X_filtered_dict['var_threshold'].append(alpha)
    X_filtered_dict['X_filtered_df'].append(X_filtered)
```

Plotting a grid across variance thresholds as follows.

[10]:
```
img_paths = []
for alpha in thresholds_list:
    selector = VarianceThreshold(threshold=alpha)
    try:
        X_reduced = selector.fit_transform(X)
        selected_columns = X.columns[selector.get_support()]
        if len(selected_columns) == 0:
            continue
        X_filtered = pd.DataFrame(X_reduced, columns=selected_columns, index=X.
 ↪index)

        for method, metric in combinations:
            g = sns.clustermap(X_filtered, method=method, metric=metric,␣
 ↪cmap='coolwarm')
            g.fig.suptitle(f'{method} - {metric} | Variance > {alpha}', y=1.05)
            img_path = f'Media/viz/03/single_plots/
 ↪reduced_clustermap_{method}_{metric}_var{alpha}.png'
            g.savefig(img_path, bbox_inches='tight')
            plt.close(g.fig)
            img_paths.append((img_path, alpha))
    except ValueError:
        continue  # Skip thresholds that drop all features

n_cols = 3
n_rows = -(-len(img_paths) // n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(22, 7 * n_rows))

for ax, (path, alpha) in zip(axes.flatten(), img_paths):
    img = Image.open(path)
    ax.imshow(img)
    ax.set_title(f'Variance > {alpha}', fontsize=12)
    ax.axis('off')

plt.tight_layout()
```

```
plt.savefig('Media/viz/03/03_hierarchical_biclustering_variance_grid.png')
plt.show()
```



This feature selection helps to make the patterns between the stop words (features) and the chapters a lot more distinguishable by dropping the sparse features that experience little variance across chapters. From this we can see bands that may indicate a shift in semantics across chapters!

04_clustering_comparison

April 17, 2025

Load necessary libraries.

[1]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap

### Msc
import warnings

### OOP
from ml_utils import ClusterEvaluator
```

Load dataset.

[2]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
 ↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
X = X.to_numpy() # change pd.DataFrame to np.ndarray
authors = df['Author'].values  # n_samples-length array
```

We can also assign a hyperparameter $K = 4$. This represents the number of clusters (which we already know; there are 4 authors). Later on we will be hyperparameter tuning for some K based on stability!

[3]:
```
K_ = 4
```

Let us also define a dictionary to store our results.

[4]:
```
accuracy_dict = {}
```

1 Clustering Methods

1.1 Kmeans++

[5]:
```
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300) #␣
 ↪K-means++ initialization
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'London'}
The accuracy of Kmeans++ = 0.9096313912009513
```

1.2 Gaussian Mixture Models

[6]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X)
y_gmm = gmm.predict(X)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9512485136741974
```

1.3 Spectral Clustering

Spectral clustering already had a dimensional reduction by design, i.e., spectral clustering is essentially spectral embedding followed by kmeans! Therefore, we expect this 'raw' method to perform best compared to the other methods without any form of dimensionality reduction!

[7]:
```
spectral = SpectralClustering(n_clusters=K_, affinity='nearest_neighbors')
y_spectral = spectral.fit_predict(X)
```

```
accuracy, mapping = ClusterEvaluator(y_spectral,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['spectral clustering'] = accuracy
```

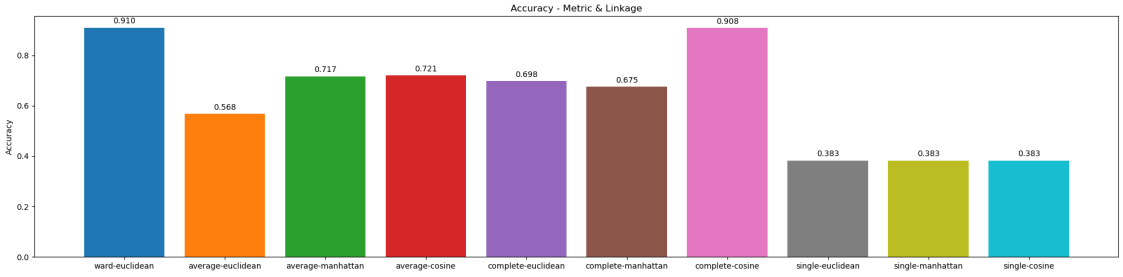
```
The mapping based on mode = {0: 'Shakespeare', 1: 'Austen', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9881093935790726
```

1.4 Hierarchial Clustering

[8]:
```
# Establish all possible combinations for linkage and metric!
methods = []
linkage_methods = ['ward', 'average', 'complete', 'single']
metrics = ['euclidean', 'manhattan', 'cosine']

for linkage in linkage_methods:
    for metric in metrics:
        if linkage == 'ward' and metric != 'euclidean':
            continue  # ward only supports euclidean
        methods.append((linkage, metric))

# Loop through all combinations and determine best accuracy!
hierarchical_accuracy_dict = {}
vals = df['Author'].values
for linkage, metric in methods:
    hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
    y_hierarchical = hierarchical.fit_predict(X)  # Fit and predict in one step
    accuracy, mapping = ClusterEvaluator(y_hierarchical,vals).accuracy # Use␣
 ↪OOP code
    # print(f'The mapping based on mode = {mapping}')
    # print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
    hierarchical_accuracy_dict[f'{linkage}-{metric}'] = accuracy
```

[9]:
```
keys = list(hierarchical_accuracy_dict.keys())
vals = list(hierarchical_accuracy_dict.values())
colors = plt.colormaps.get_cmap('tab10').colors

fig, ax = plt.subplots(figsize=(20, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
```

```
plt.savefig('Media/viz/04/04_hier_accuracy_across_metric_linkage')
plt.show()
```

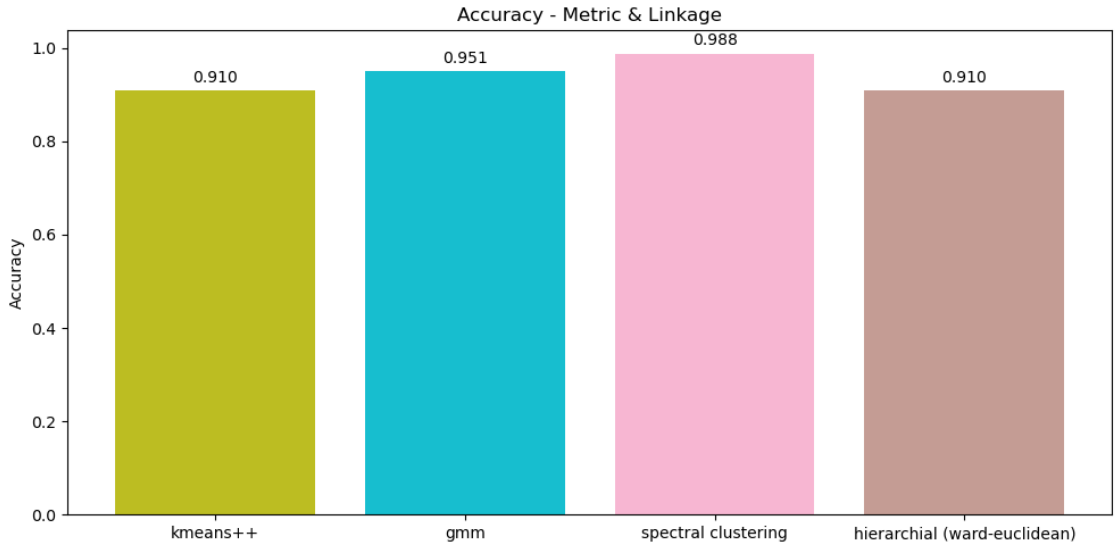


Lets store the highest accuracy params to compare across other methods - ward + euclidean.

[10]:
```
accuracy_dict['hierarchial (ward-euclidean)'] =␣
↪hierarchical_accuracy_dict['ward-euclidean']
```

1.5 Comparison

[11]:
```
keys = list(accuracy_dict.keys())
vals = list(accuracy_dict.values())

colors = ['#bcbd22', '#17becf', '#f7b6d2', '#c49c94']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods')
plt.show()
```



Clearly, spectral clustering performs the best because it has a build in dimensional reduction! Therefore, now let us apply UMAP to all these methods and recompare.

2 Dimensional Reduction + Clustering Methods

Use UMAP as dimensionality reduction for all methods (except spectral clustering as this already uses spectral embedding so we will not repeat this).

[12]:
```
warnings.filterwarnings("ignore", category=UserWarning, module="umap") #␣
↪Suppress the specific UMAP warning on parralelism
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3)
X_umap = umap_model.fit_transform(X)
```

Store results in the following dictionary to compare across methods.

[13]:
```
umap_accuracy_dict = {}
umap_accuracy_dict['spectral clustering'] = accuracy_dict['spectral clustering']
```

2.1 Kmeans++

[14]:
```
# Run K-means++ on UMAP-reduced data
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300)
kmeans.fit(X_umap)
y_kmeans = kmeans.predict(X_umap)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
```

```
umap_accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.2 Gaussian Mixture Models

[15]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X_umap)
y_gmm = gmm.predict(X_umap)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
umap_accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Shakespeare', 1: 'London', 2: 'Austen', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.3 Hierarchial Clustering

[16]:
```
linkage = 'ward'
metric = 'euclidean'
hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
y_hierarchical = hierarchical.fit_predict(X_umap)  # Fit and predict in one step
accuracy, mapping = ClusterEvaluator(y_hierarchical,df['Author'].values).
 ↪accuracy # Use OOP code
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
umap_accuracy_dict[f'hierarchial ({linkage}-{metric})'] = accuracy
```

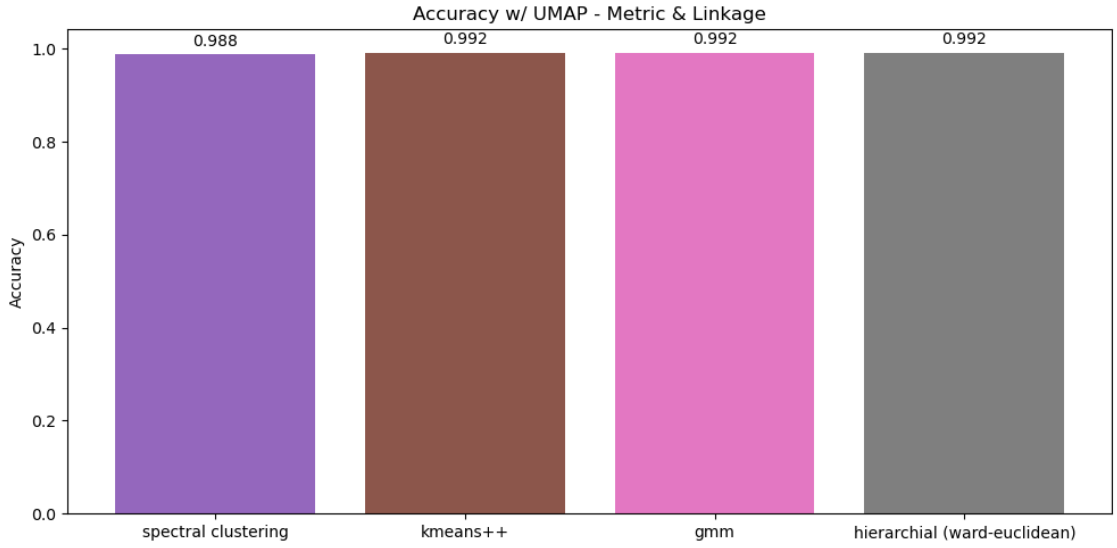
```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy hierarchial clustering with ward and euclidean = 0.9916765755053508
```

2.4 Comparison

As we can see, all these methods (with the exception of spectral clustering as this did not use UMAP) converge to the same accuracy. This is because after applying UMAP we get very clearly defined clusters so there is not much/any room for these methods to diverge in their clustering assignments! Therefore they will yield the same accuracy; this makes sense!

[17]:
```
keys = list(umap_accuracy_dict.keys())
vals = list(umap_accuracy_dict.values())
```

```
colors = ['#9467bd', '#8c564b', '#e377c2', '#7f7f7f']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy w/ UMAP - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods_with_umap')
plt.show()
```



05_generalizibility_silhouette_score

April 17, 2025

Load necessary libraries.

[9]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
 ↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap
from sklearn.metrics import silhouette_samples

### Msc
import warnings
from PIL import Image

### OOP
from ml_utils import SilhouetteEvaluator, ClusterEvaluator, make_gmm,␣
 ↪make_hierarchical, make_spectral
```

Load dataset.

[10]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array

X_clean = df.drop(columns=['Author'])
X = X_clean.to_numpy()
```

```
# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = umap_model.fit_transform(X)
```

1 Validation

The method with the highest silhouette score generalizes best!

1.1 Kmeans++

Kmeans++ with and without UMAP:

1.1.1 Generalizibility

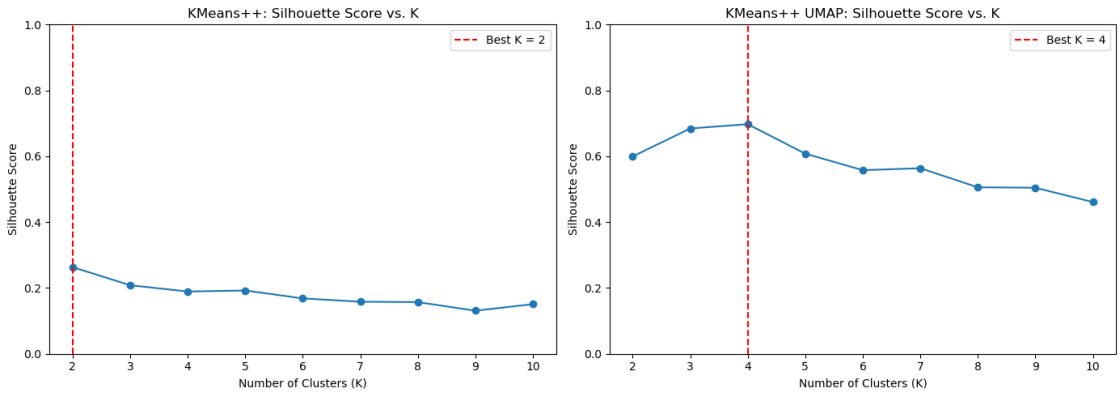
[11]:
```
# Kmeans++
print('\033[1m' + 'Kmeans++ without dimension reduction:' + '\033[0m')
evaluator_kmeans = SilhouetteEvaluator(X, KMeans, k_range=range(2, 11),␣
 ↪init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Kmeans++ with UMAP
print('\033[1m' + 'Kmeans++ with dimension reduction (UMAP):' + '\033[0m')
evaluator_kmeans_umap = SilhouetteEvaluator(X_umap, KMeans, k_range=range(2,␣
 ↪11), init='k-means++', n_init=10, max_iter=300)
scores, best_k = evaluator_kmeans_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

y_min = 0
y_max = 1

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_kmeans.plot("KMeans++", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_kmeans_silhouette_score')
plt.show()
```

Kmeans++ without dimension reduction:

```
The scores across K = {2: 0.2632255060236338, 3: 0.20836751818875782, 4:
0.18927180616599434, 5: 0.1924461042273646, 6: 0.16832953504579296, 7:
0.15834519448558668, 8: 0.15708860765446275, 9: 0.13097719253825474, 10:
0.15080653215185813}
Best K: 2
```

Kmeans++ with dimension reduction (UMAP):

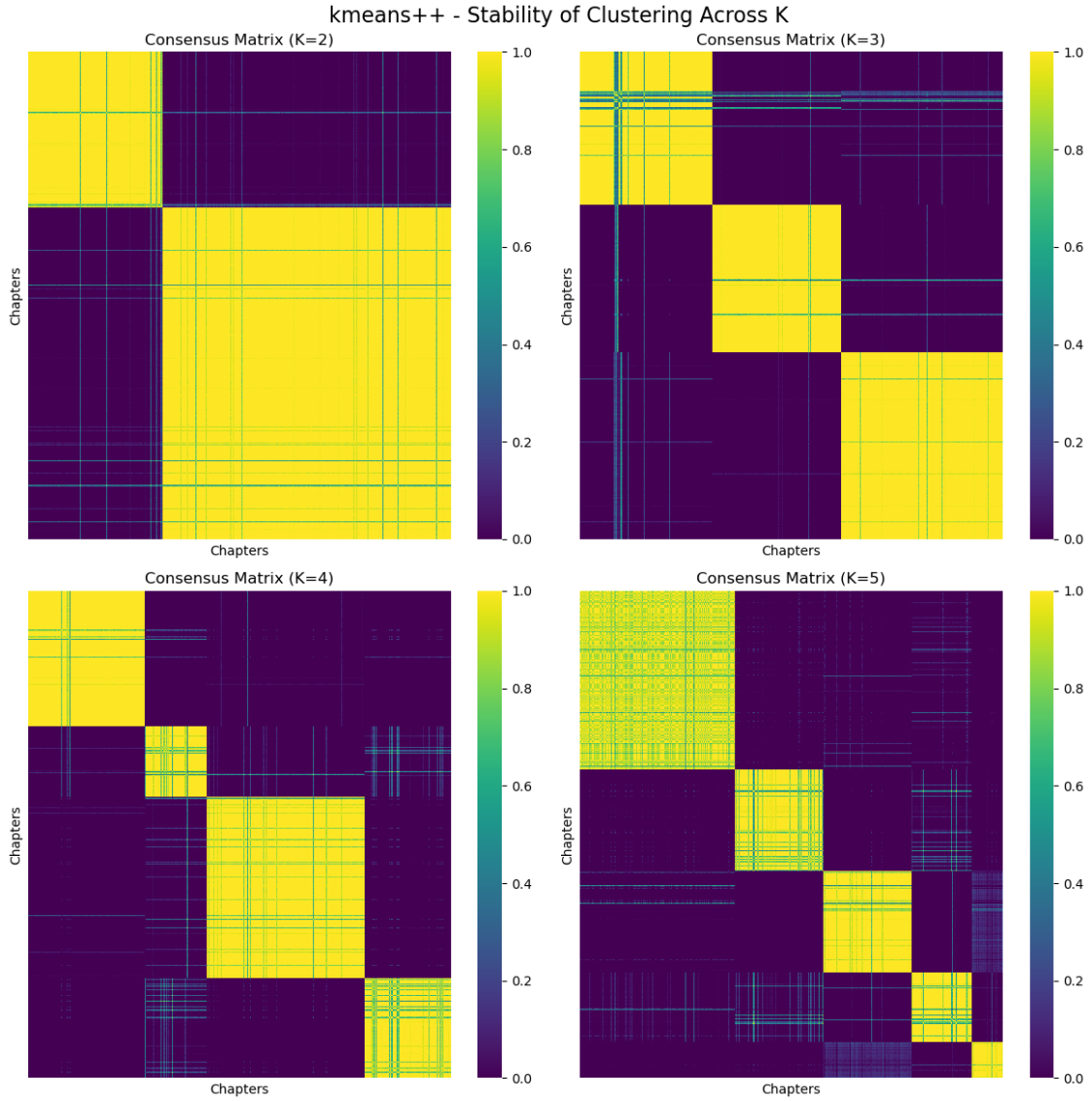
```
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60805935, 6: 0.55773854, 7: 0.5637968, 8: 0.505771, 9: 0.5044795, 10:
0.4610399}
Best K: 4
```



1.1.2 Stability

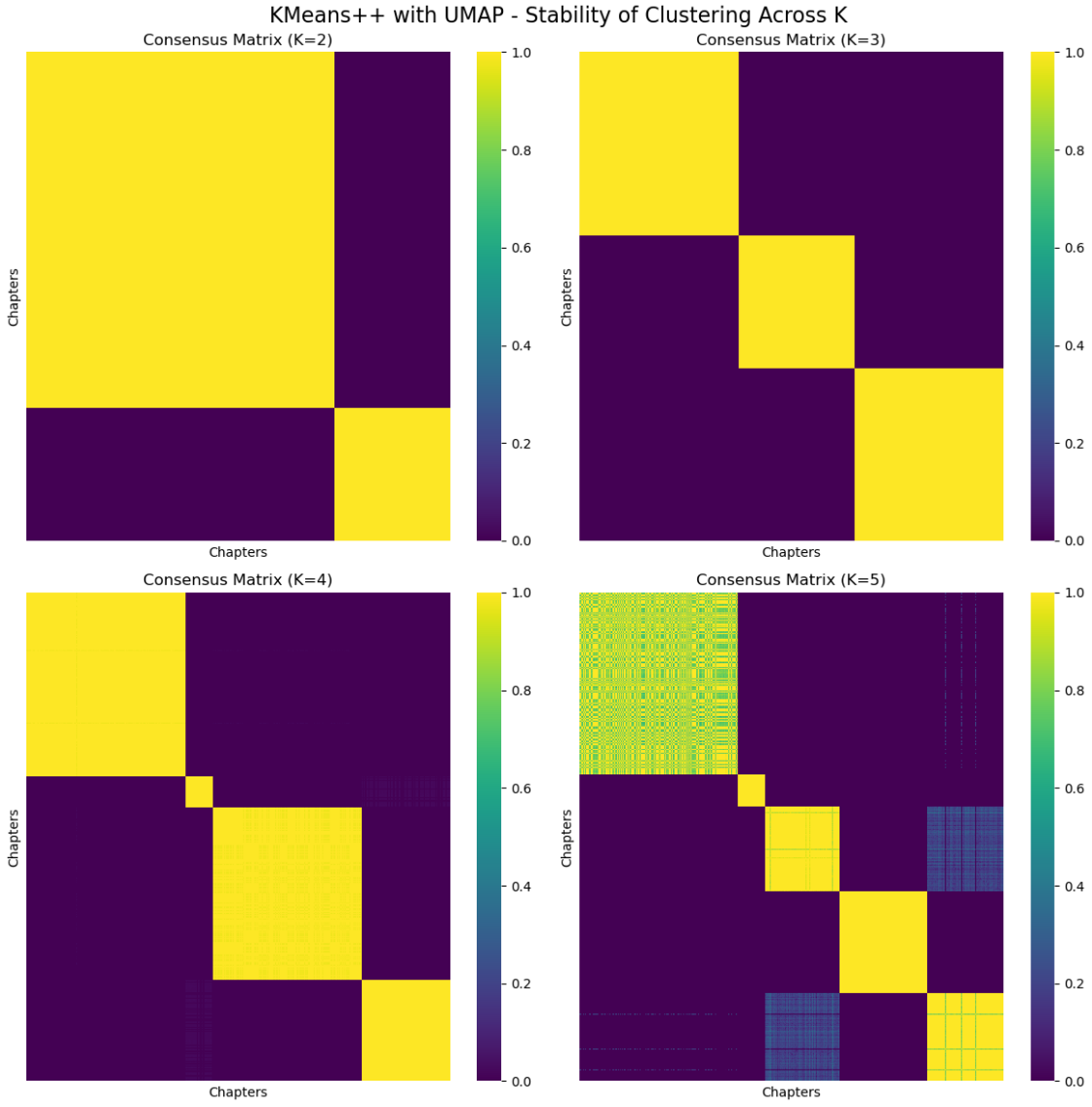
[12]:
```
method = 'kmeans++'

rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_train = X[train_idx]  # Using saved NumPy array
```

```
        km = KMeans(n_clusters=K, random_state=n, n_init=10)
        cluster_labels = km.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, random_state=42, n_init=10).
↪fit_predict(X)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans')
plt.show()
```



[13]:
```
method = 'KMeans++ with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
```

```
    sampled_matrix = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        km = KMeans(n_clusters=K, n_init=10, init='k-means++', random_state=i)
        labels = km.fit_predict(X_sample)

        sampled_matrix[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus_matrix[np.ix_(idx_sample, idx_sample)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
↪where=sampled_matrix != 0)
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = KMeans(n_clusters=K, n_init=10, init='k-means++',␣
↪random_state=42).fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus_matrix[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_kmeans_umap')
plt.show()
```



1.2 Gaussian Mixture Models

1.2.1 Generalizibility

[14]:
```
# GMM
print('\033[1m' + 'GMM without dimension reduction:' + '\033[0m')
evaluator_gmm = SilhouetteEvaluator(X, make_gmm(), k_range=range(2, 11))
scores, best_k = evaluator_gmm.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# GMM with UMAP
print('\033[1m' + 'GMM with dimension reduction (UMAP):' + '\033[0m')
```

```
evaluator_gmm_umap = SilhouetteEvaluator(X_umap, make_gmm(), k_range=range(2,␣
↪11))
scores, best_k = evaluator_gmm_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_gmm.plot("GMM", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()

plt.savefig('Media/viz/05/05_gmm_silhouette_score')
plt.show()
```

GMM without dimension reduction:

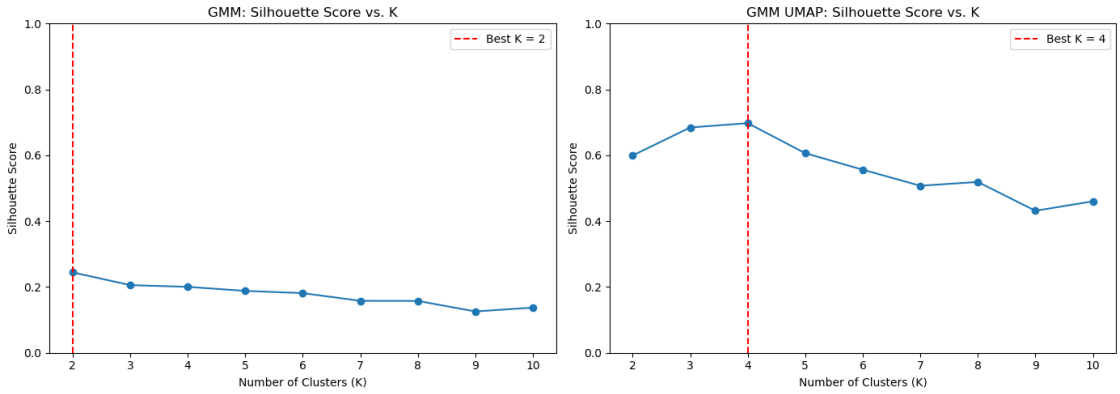
The scores across K = {2: 0.24434605026692022, 3: 0.20578434655866124, 4: 0.2004213780628659, 5: 0.18814970124375807, 6: 0.18136222277608166, 7: 0.15798484734742102, 8: 0.15765164283870803, 9: 0.1256388670716472, 10: 0.1374306014397366}

Best K: 2

GMM with dimension reduction (UMAP):

The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5: 0.60627675, 6: 0.55619246, 7: 0.5074211, 8: 0.51900756, 9: 0.43136632, 10: 0.4602903}

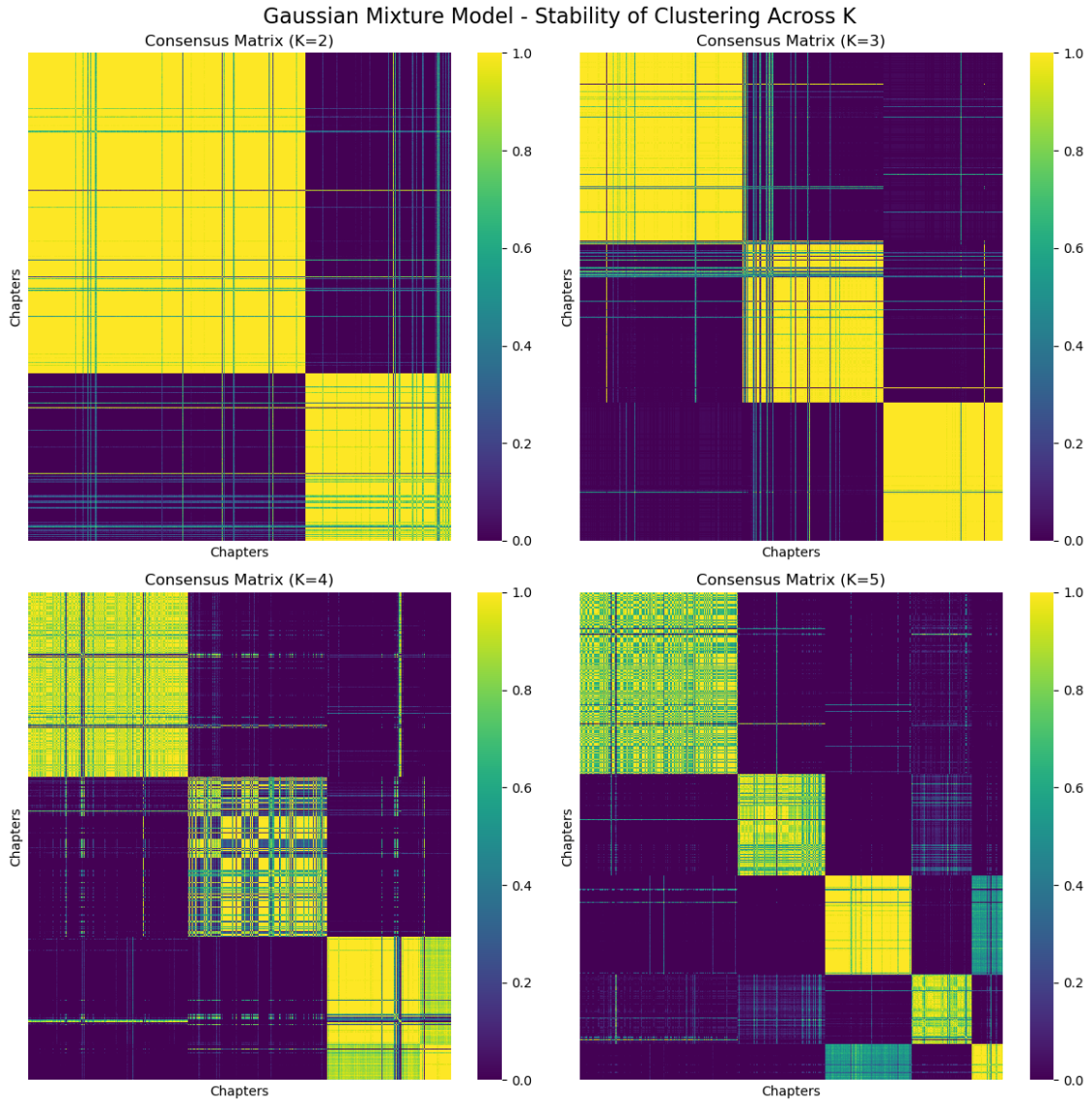
Best K: 4



1.2.2 Stability

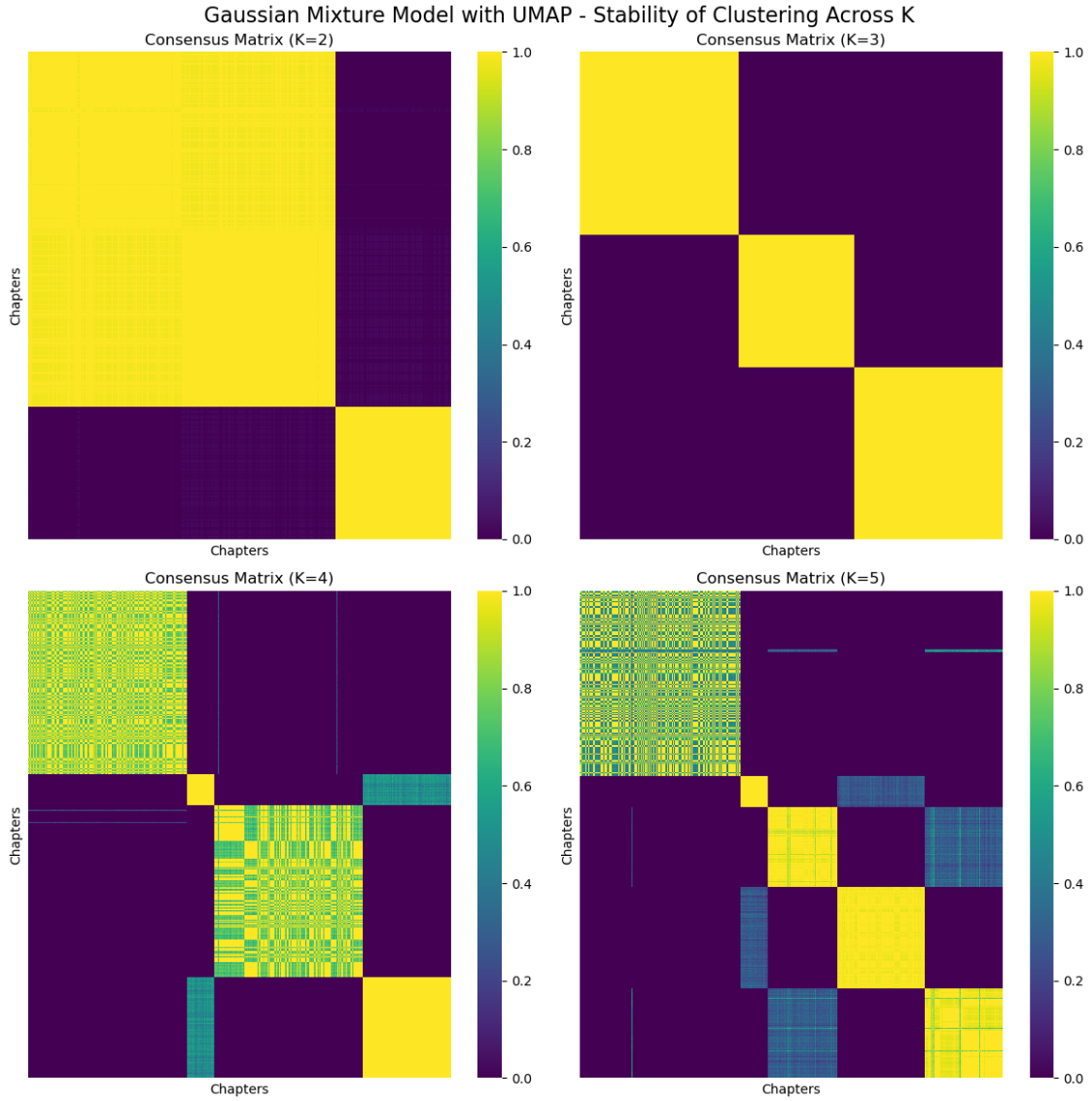
[15]:
```
method = 'Gaussian Mixture Model'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]  # NumPy version for GMM

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X).
↪predict(X)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
```

```
fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm')
plt.show()
```



[16]:
```
method = 'Gaussian Mixture Model with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()
```

```
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_sample = X_umap[idx_sample]

        gmm = GaussianMixture(n_components=K, random_state=i)
        labels = gmm.fit(X_sample).predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = GaussianMixture(n_components=K, random_state=42).fit(X_umap).
 ↪predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_gmm_umap')
plt.show()
```



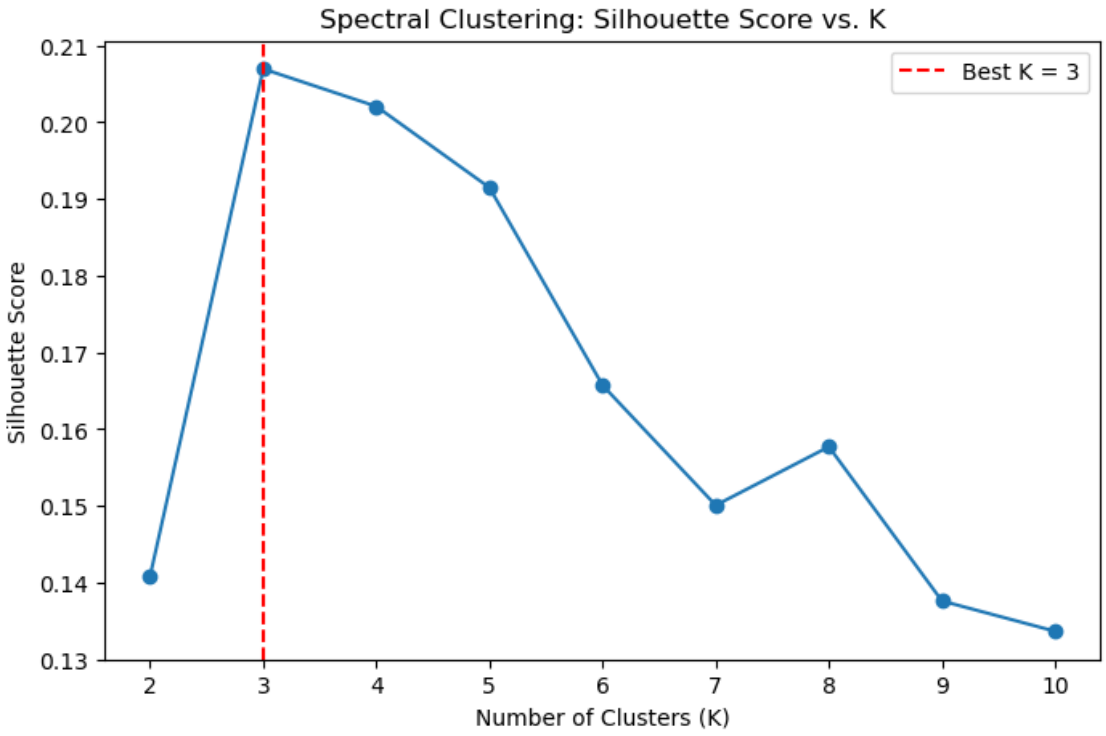
1.3 Spectral Clustering

1.3.1 Generalizibility

[17]:
```
print('\033[1m' + 'Spectral Clustering:' + '\033[0m')
evaluator_spectral = SilhouetteEvaluator(X, make_spectral(affinity =␣
 ↪'nearest_neighbors'), k_range=range(2, 11))
scores, best_k = evaluator_spectral.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

evaluator_spectral.plot('Spectral Clustering')
```

```
plt.savefig('Media/viz/05/05_spectral_silhouette_score')
```

```
Spectral Clustering:
The scores across K = {2: 0.1408300984760483, 3: 0.20696881798137212, 4:
0.2020762541851448, 5: 0.1914837165759898, 6: 0.16570630789830376, 7:
0.15006558354299535, 8: 0.1577000857169114, 9: 0.13758976585902763, 10:
0.1336400256361507}
Best K: 3
```



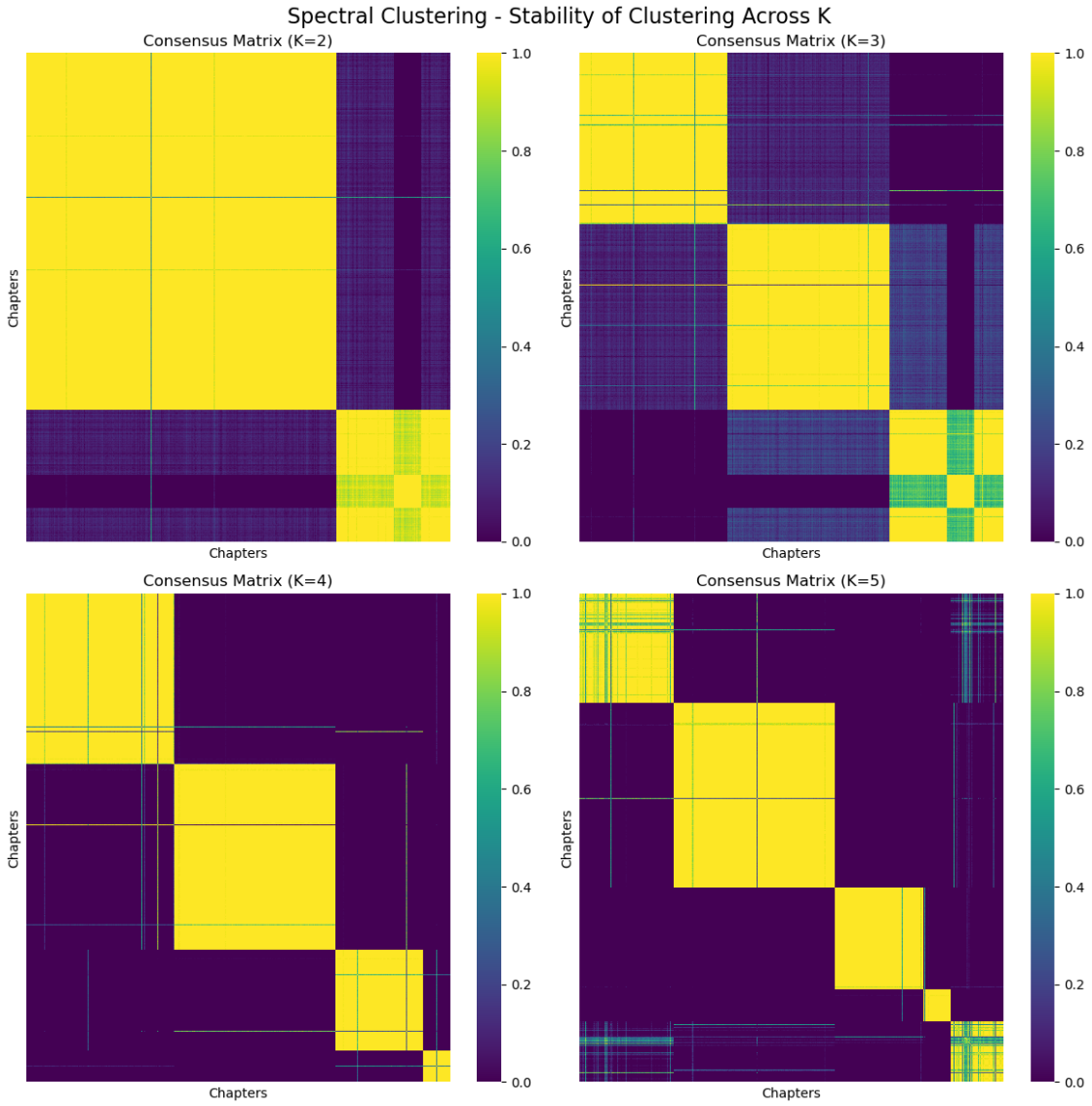
Although the silhouette score is commonly used to evaluate clustering performance, it assumes that clusters are spherical in shape, which does not align with the strengths of spectral clustering. Spectral clustering excels at uncovering non-linear and arbitrarily shaped cluster structures, which silhouette score is not well-equipped to quantify. Therefore, the low silhouette score observed here may understate the actual performance of spectral clustering. In fact, the spectral embedding reveals visually distinct and meaningful clusters (see notebook 02), suggesting that the method is effective despite the numerical metric.

1.3.2 Stability

[18]:
```
## METHOD
method = 'Spectral Clustering'

rng = np.random.default_rng(42)
iterations = 100
```

```
K_values = [2, 3, 4, 5]
n_samples = X_clean.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus_matrix = np.zeros((n_samples, n_samples))
    sampled_matrix = np.zeros((n_samples, n_samples))

    for n in range(iterations):
        train_idx = rng.choice(n_samples, size=int(0.7 * n_samples),␣
 ↪replace=False)
        X_train = X_clean.iloc[train_idx].to_numpy()

        sc = SpectralClustering(n_clusters=K, affinity='nearest_neighbors',␣
 ↪assign_labels='kmeans', random_state=n)
        cluster_labels = sc.fit_predict(X_train)

        sampled_matrix[np.ix_(train_idx, train_idx)] += 1
        co_members = np.equal.outer(cluster_labels, cluster_labels)
        consensus_matrix[np.ix_(train_idx, train_idx)] += co_members

    consensus_matrix = np.divide(consensus_matrix, sampled_matrix,␣
 ↪where=(sampled_matrix != 0))
    consensus_matrix = np.nan_to_num(consensus_matrix)

    final_labels = SpectralClustering(n_clusters=K,␣
 ↪affinity='nearest_neighbors', assign_labels='kmeans', random_state=42).
 ↪fit_predict(X_clean)
    order_idx = np.argsort(final_labels)
    consensus_matrix = consensus_matrix[order_idx][:, order_idx]

    sns.heatmap(consensus_matrix, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order_idx],␣
 ↪yticklabels=sample_names[order_idx])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
```

```
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_spectral')
plt.show()
```



$K = 4$ is the clear winner!

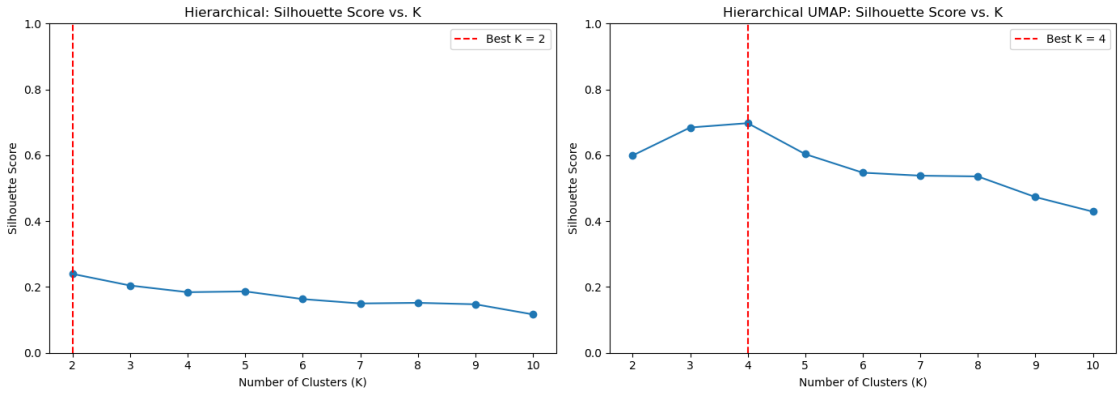
1.4 Hierarchial Clustering

Using the parameters 'ward' and 'euclidean' (because we have seen in notebook 04 that these have the highest accuracy)!

1.4.1 Generalizibility

[19]:
```
# Hierarchical Clustering
print('\033[1m' + 'Hierarchical Clustering without dimension reduction:' +␣
 ↪'\033[0m')
evaluator_hier = SilhouetteEvaluator(X, make_hierarchical(linkage='ward',␣
 ↪metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

# Hierarchical Clustering with UMAP
print('\033[1m' + 'Hierarchical Clustering with dimension reduction (UMAP):' +␣
 ↪'\033[0m')
evaluator_hier_umap = SilhouetteEvaluator(X_umap,␣
 ↪make_hierarchical(linkage='ward', metric='euclidean'), k_range=range(2, 11))
scores, best_k = evaluator_hier_umap.evaluate()
print(f'The scores across K = {scores}')
print(f"Best K: {best_k}")

fig, axs = plt.subplots(1, 2, figsize=(14, 5))

evaluator_hier.plot("Hierarchical", ax=axs[0])
axs[0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[1])
axs[1].set_ylim(y_min, y_max)

plt.tight_layout()
plt.savefig('Media/viz/05/05_silhouette_score_across_methods.png')
plt.show()
```

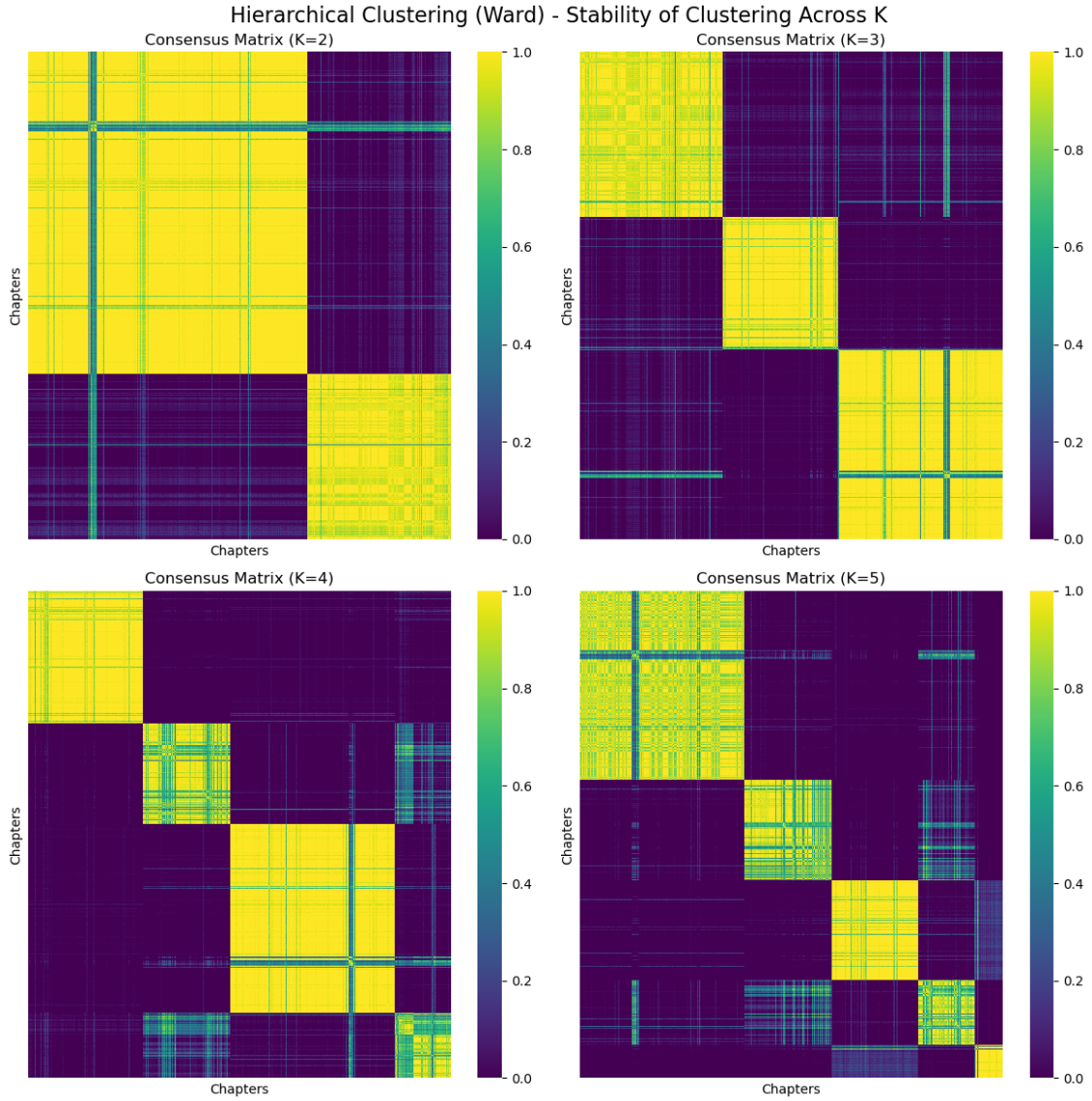
```
Hierarchical Clustering without dimension reduction:
The scores across K = {2: 0.23967574687100793, 3: 0.20437373269232859, 4:
0.18421197778513682, 5: 0.18654800991576861, 6: 0.16319550456880302, 7:
0.14995813568113975, 8: 0.1518500219855489, 9: 0.14745055233611987, 10:
0.1167264049290792}
Best K: 2
Hierarchical Clustering with dimension reduction (UMAP):
The scores across K = {2: 0.59943473, 3: 0.68435156, 4: 0.6975769, 5:
0.60349095, 6: 0.5471597, 7: 0.5380937, 8: 0.53579855, 9: 0.47313103, 10:
0.42873582}
Best K: 4
```



1.4.2 Stability

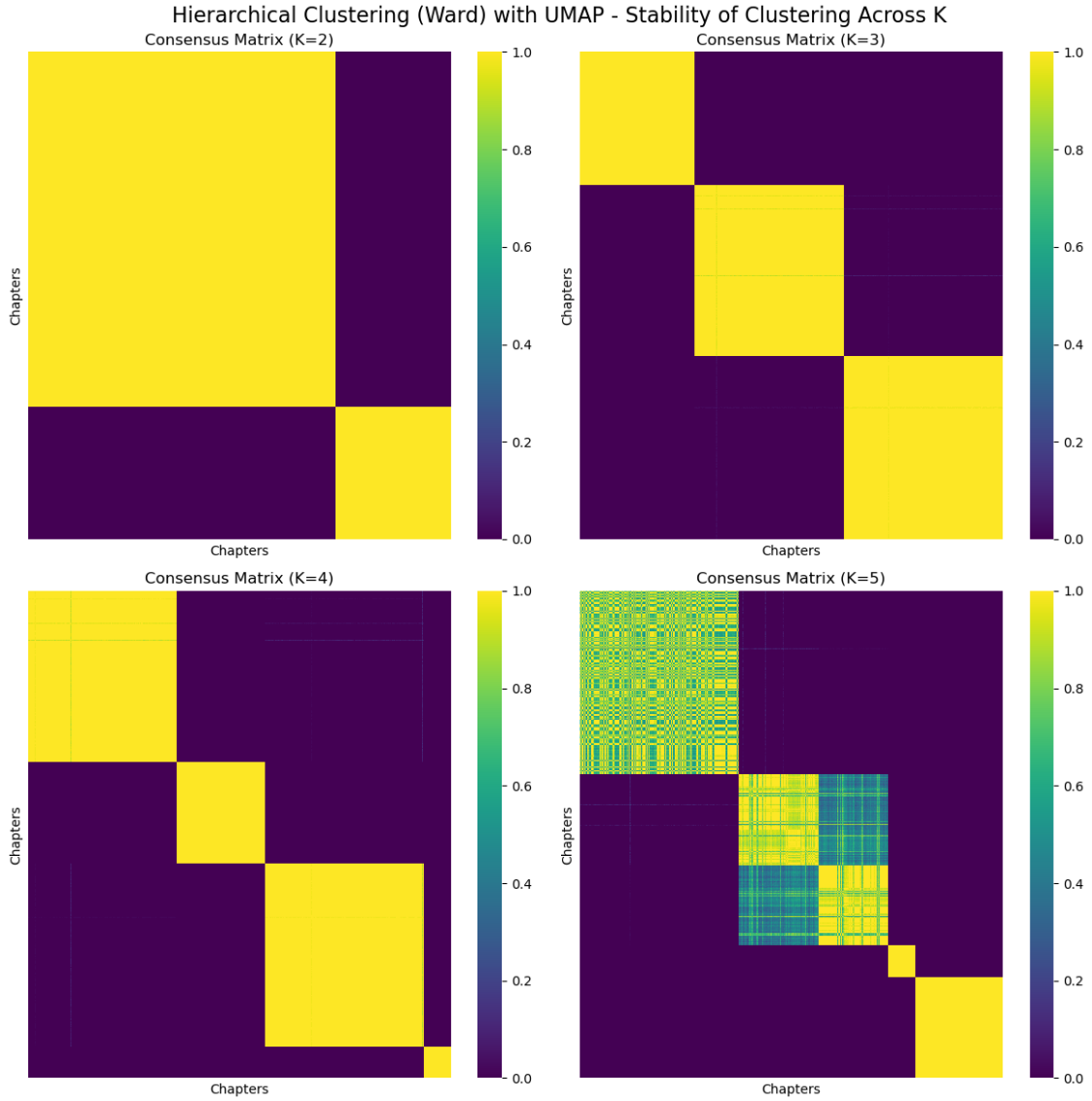
[20]:
```
method = 'Hierarchical Clustering (Ward)'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X)
```

```
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
 ↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical')
plt.show()
```



[21]:
```
method = 'Hierarchical Clustering (Ward) with UMAP'
rng = np.random.default_rng(42)
iterations = 100
K_values = [2, 3, 4, 5]
n_samples = X_umap.shape[0]
sample_names = X_clean.index.to_numpy()

fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()

for idx, K in enumerate(K_values):
    consensus = np.zeros((n_samples, n_samples))
```

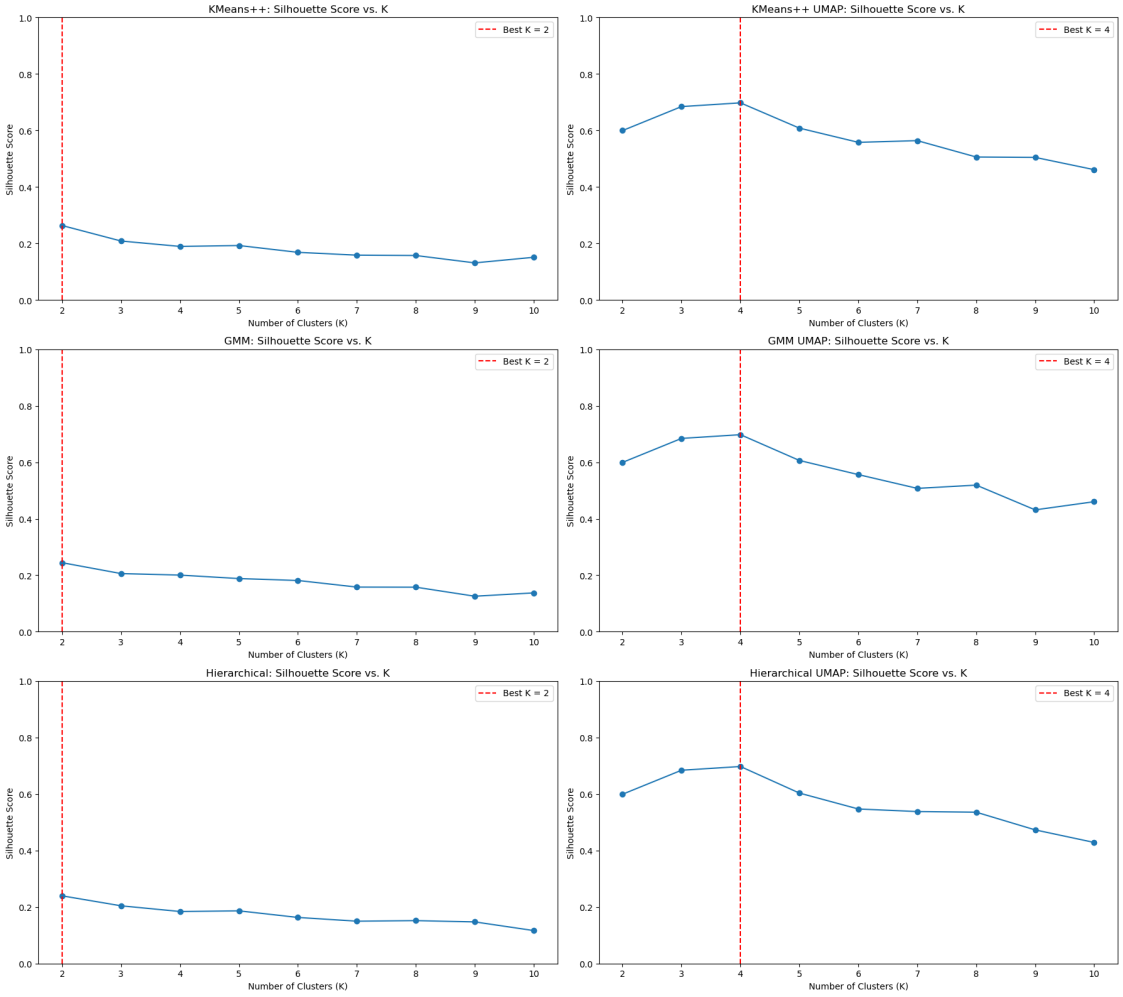
```
    sampled = np.zeros((n_samples, n_samples))

    for i in range(iterations):
        idx_sample = rng.choice(n_samples, size=int(0.7 * n_samples),␣
↪replace=False)
        X_sample = X_umap[idx_sample]

        model = AgglomerativeClustering(n_clusters=K, linkage='ward')
        labels = model.fit_predict(X_sample)

        sampled[np.ix_(idx_sample, idx_sample)] += 1
        co_members = np.equal.outer(labels, labels)
        consensus[np.ix_(idx_sample, idx_sample)] += co_members

    consensus = np.divide(consensus, sampled, where=sampled != 0)
    consensus = np.nan_to_num(consensus)

    final_labels = AgglomerativeClustering(n_clusters=K, linkage='ward').
↪fit_predict(X_umap)
    order = np.argsort(final_labels)
    matrix_sorted = consensus[order][:, order]

    sns.heatmap(matrix_sorted, ax=axes[idx], cmap='viridis',
                xticklabels=sample_names[order],␣
↪yticklabels=sample_names[order])
    axes[idx].set_title(f'Consensus Matrix (K={K})')
    axes[idx].set_xticks([])
    axes[idx].set_yticks([])
    axes[idx].set_xlabel('Chapters')
    axes[idx].set_ylabel('Chapters')
    axes[idx].tick_params(axis='x', rotation=90, labelsize=8)
    axes[idx].tick_params(axis='y', labelsize=8)

fig.suptitle(f'{method} - Stability of Clustering Across K', fontsize=16)
plt.tight_layout()
plt.savefig('Media/viz/05/05_consensus_heatmaps_hierarchical_umap')
plt.show()
```



2 Comparison Viz

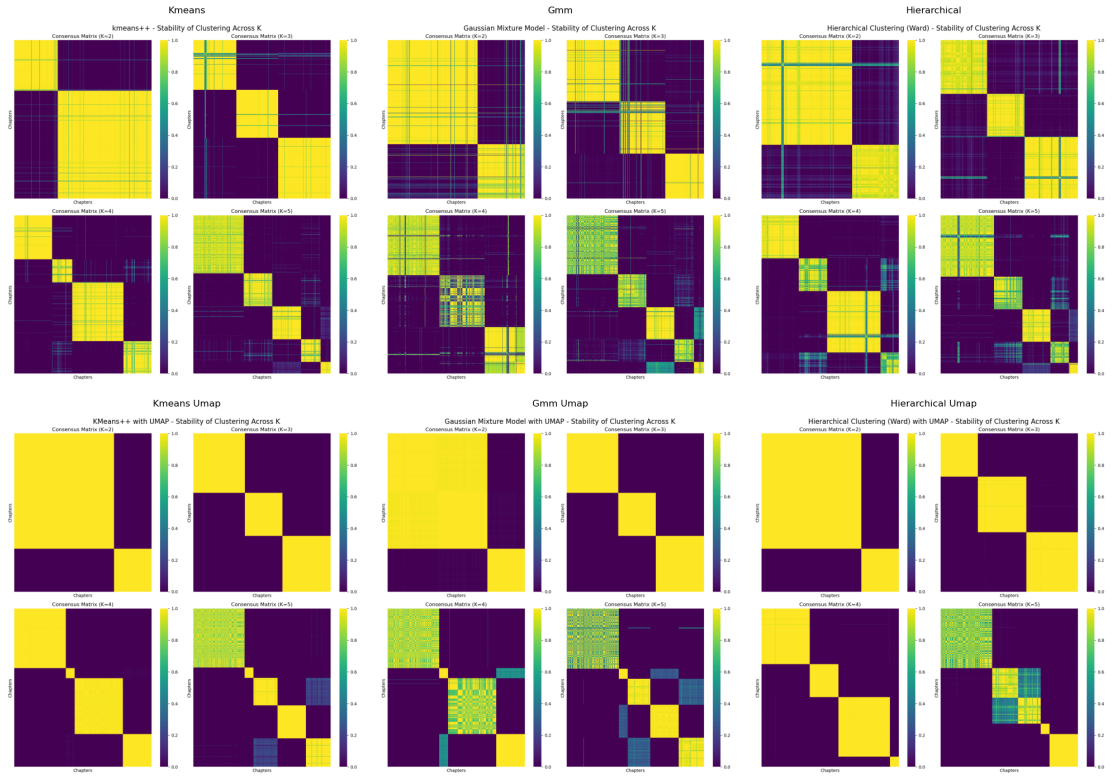
[22]:
```
fig, axs = plt.subplots(3, 2, figsize=(18, 16))

evaluator_kmeans.plot("KMeans++", ax=axs[0,0])
axs[0,0].set_ylim(y_min, y_max)
evaluator_kmeans_umap.plot("KMeans++ UMAP", ax=axs[0,1])
axs[0,1].set_ylim(y_min, y_max)

evaluator_gmm.plot("GMM", ax=axs[1,0])
axs[1,0].set_ylim(y_min, y_max)
```

```
evaluator_gmm_umap.plot("GMM UMAP", ax=axs[1,1])
axs[1,1].set_ylim(y_min, y_max)

evaluator_hier.plot("Hierarchical", ax=axs[2,0])
axs[2,0].set_ylim(y_min, y_max)

evaluator_hier_umap.plot("Hierarchical UMAP", ax=axs[2,1])
axs[2,1].set_ylim(y_min, y_max)

plt.savefig('Media/viz/05/05_silhouette_score_across_methods')
plt.tight_layout()
```



[26]:
```
heatmap_paths = [
    "Media/viz/05/05_consensus_heatmaps_kmeans.png",
    "Media/viz/05/05_consensus_heatmaps_gmm.png",
```

```
    "Media/viz/05/05_consensus_heatmaps_hierarchical.png",
    "Media/viz/05/05_consensus_heatmaps_kmeans_umap.png",
    "Media/viz/05/05_consensus_heatmaps_gmm_umap.png",
    "Media/viz/05/05_consensus_heatmaps_hierarchical_umap.png"
]

heatmap_images = [Image.open(path) for path in heatmap_paths]

fig, axes = plt.subplots(2, 3, figsize=(20, 14), constrained_layout=True)
axes = axes.flatten()
for ax, img, path in zip(axes, heatmap_images, heatmap_paths):
    ax.imshow(img)
    ax.set_title(
        os.path.basename(path)
          .replace("05_consensus_heatmaps_", "")
          .replace(".png", "")
          .replace("_", " ")
          .title(),
        fontsize=12
    )
    ax.axis('off')
plt.savefig("Media/viz/05/05_all_consensus_heatmaps_grid.png")
plt.show()
```



06_em_algorithm_fit_gmm

April 17, 2025

Load necessary libraries.

[2]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

### ML packages
from scipy.stats import multivariate_normal
from sklearn.mixture import GaussianMixture
import umap
from sklearn.metrics import adjusted_rand_score
from scipy.spatial.distance import cdist

### Msc
import warnings

### OOP Code
from ml_utils import GaussianMixtureEM
```

Load dataset.

[3]:
```
df = pd.read_csv('00_authors.csv').rename(columns={'Unnamed: 0': 'Author'}).
 ↪drop(columns='BookID')
authors = df['Author'].values  # n_samples-length array
X = df.drop(columns=['Author'])

# UMAP dimensionality reduction
warnings.filterwarnings("ignore", category=UserWarning, module="umap")  #␣
 ↪suppress joblib warning
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3, random_state=42)
X_umap = pd.DataFrame(umap_model.fit_transform(X))
```

EM Algorithm for Multivariate Gaussian Mixture Model

Let the density of $x \in \mathbb{R}^d$ be: $p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x \mid \mu_k, \Sigma_k)$

where: - π_k are the mixing proportions (with $\sum_k \pi_k = 1$), - $\mu_k \in \mathbb{R}^d$ is the mean of component k, - $\Sigma_k \in \mathbb{R}^{d \times d}$ is the covariance matrix.

E-step

$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i|\mu_j,\Sigma_j)}$ where $\mathcal{N}(x \mid \mu, \Sigma)$ is the multivariate Gaussian density: $\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$

M-step

Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

Update mean: $\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i$

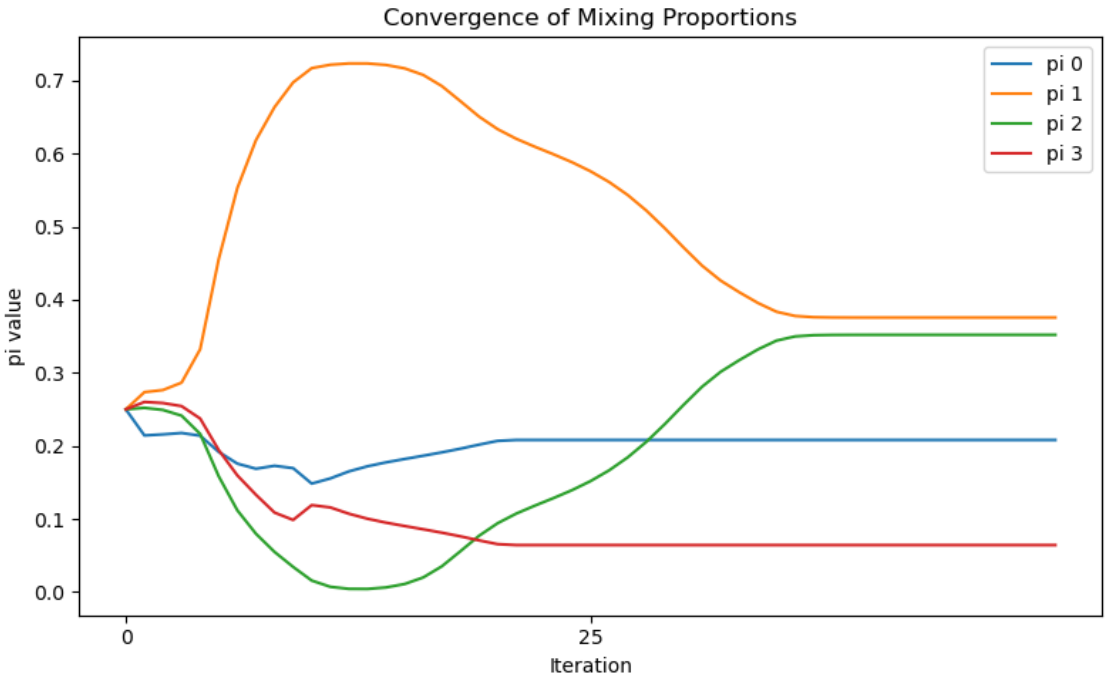
Update covariance: $\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T$

Update mixing weights: $\pi_k = \frac{N_k}{N}$

Repeat E-step and M-step until convergence (e.g., change in log-likelihood is below a threshold).

[4]:
```
# code in -> ml_utils.py
# code also pasted at appendix
model = GaussianMixtureEM(K=4, num_iterations=50, allow_singular=False)
results = model.fit_fast(X_umap)
```

[5]:
```
pis_dict = results['pis_dict']
iterations = [key for key in pis_dict if key.startswith('Iteration_')]
iterations_sorted = sorted(iterations, key=lambda x: int(x.split('_')[1]))

pi_matrix = np.array([pis_dict['Initial'][0]] + [pis_dict[it][0] for it in␣
 ↪iterations_sorted])

plt.figure(figsize=(8, 5))
for k in range(pi_matrix.shape[1]):
    plt.plot(range(len(pi_matrix)), pi_matrix[:, k], label=f'pi {k}')

plt.title('Convergence of Mixing Proportions')
plt.xlabel('Iteration')
plt.ylabel('pi value')

steps = 25
plt.xticks(range(0, len(iterations_sorted), steps))

plt.legend()
plt.tight_layout()
plt.savefig('Media/viz/06/06_em_convergence_plot')
plt.show()
```



1 Compare

Now to compare my manually EM function to the built in Sklearn package!

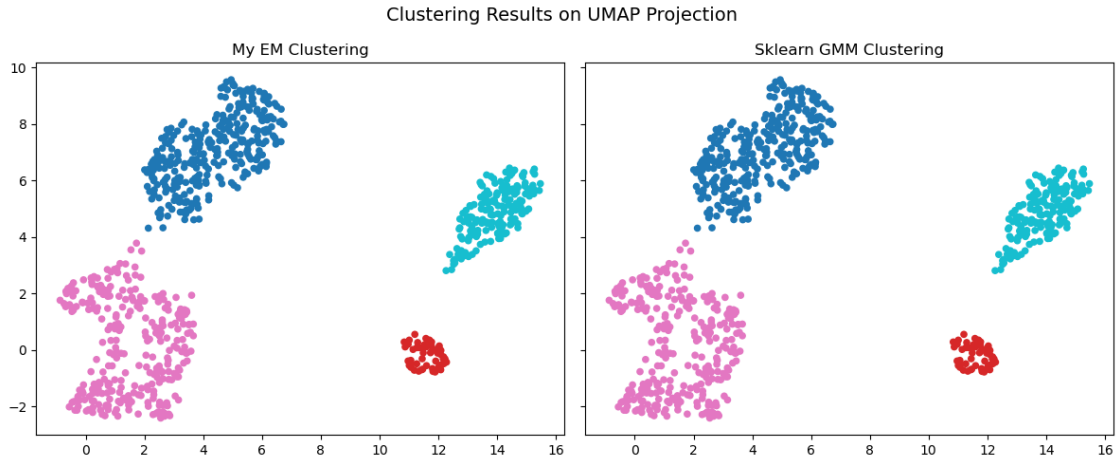
[6]:
```
sk_model = GaussianMixture(n_components=4, random_state=42)
sk_model.fit(X_umap)

# Your final results
my_pi = results['pis_dict'][f'Iteration_{model.num_iterations - 1}'][0]
my_mu = np.array(results['means'])
my_cov = np.array(results['cov'])

# Sklearn results
sk_pi = sk_model.weights_
sk_mu = sk_model.means_
sk_cov = sk_model.covariances_
```

[7]:
```
dist_matrix = cdist(my_mu, sk_mu)
matches = np.argmin(dist_matrix, axis=1)

print("Matching components based on mean proximity:")
for i, j in enumerate(matches):
    print(f"Component {i} → Sklearn {j} | Distance: {dist_matrix[i, j]:.4f}")
```

```python
print("\nMixing weights comparison:")
for i, j in enumerate(matches):
    print(f"Component {i}: mine={my_pi[i]:.4f} | sklearn={sk_pi[j]:.4f}")

print("\nMean vector L2 distances:")
for i, j in enumerate(matches):
    delta = np.linalg.norm(my_mu[i] - sk_mu[j])
    print(f"Component {i}: || _mine -  _sklearn|| = {delta:.4f}")

my_labels = np.argmax(results['gamma'], axis=1)
sk_labels = sk_model.predict(X_umap)
ari = adjusted_rand_score(my_labels, sk_labels)

print(f"\nAdjusted Rand Index: {ari:.4f}")

remapped_labels = np.zeros_like(my_labels)
for i, j in enumerate(matches):
    remapped_labels[my_labels == i] = j

X_umap_np = X_umap.to_numpy() if isinstance(X_umap, pd.DataFrame) else X_umap

fig, axs = plt.subplots(1, 2, figsize=(12, 5), sharex=True, sharey=True)

axs[0].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=remapped_labels,␣
 ↪cmap='tab10', s=20)
axs[0].set_title("My EM Clustering")

axs[1].scatter(X_umap_np[:, 0], X_umap_np[:, 1], c=sk_labels, cmap='tab10',␣
 ↪s=20)
axs[1].set_title("Sklearn GMM Clustering")

plt.suptitle("Clustering Results on UMAP Projection", fontsize=14)
plt.tight_layout()
plt.savefig('Media/viz/06/06_label_2dim_comparison_viz')
plt.show()
```

```
Matching components based on mean proximity:
Component 0 → Sklearn 3 | Distance: 0.0000
Component 1 → Sklearn 0 | Distance: 0.0000
Component 2 → Sklearn 2 | Distance: 0.0001
Component 3 → Sklearn 1 | Distance: 0.0000

Mixing weights comparison:
Component 0: mine=0.2081 | sklearn=0.2081
Component 1: mine=0.3757 | sklearn=0.3757
Component 2: mine=0.3520 | sklearn=0.3520
Component 3: mine=0.0642 | sklearn=0.0642
```

```
Mean vector L2 distances:
Component 0: || _mine -  _sklearn|| = 0.0000
Component 1: || _mine -  _sklearn|| = 0.0000
Component 2: || _mine -  _sklearn|| = 0.0001
Component 3: || _mine -  _sklearn|| = 0.0000

Adjusted Rand Index: 1.0000
```



Perfect matchup as sklearns package as we make iterations of our EM algorithm sufficiently large!