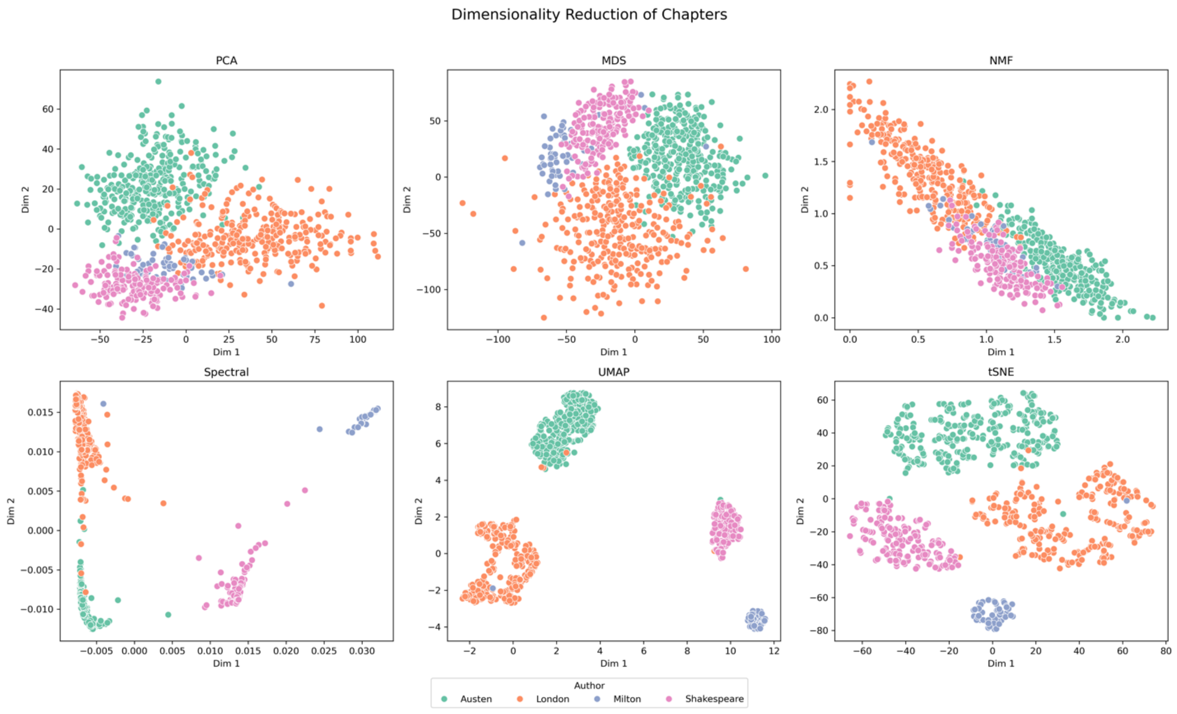
Dimitris Markopoulos
Statistical Machine Learning

Thursday, April 17
Columbia University

<u>Visualizations</u>

■ *observations (book chapters)*

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

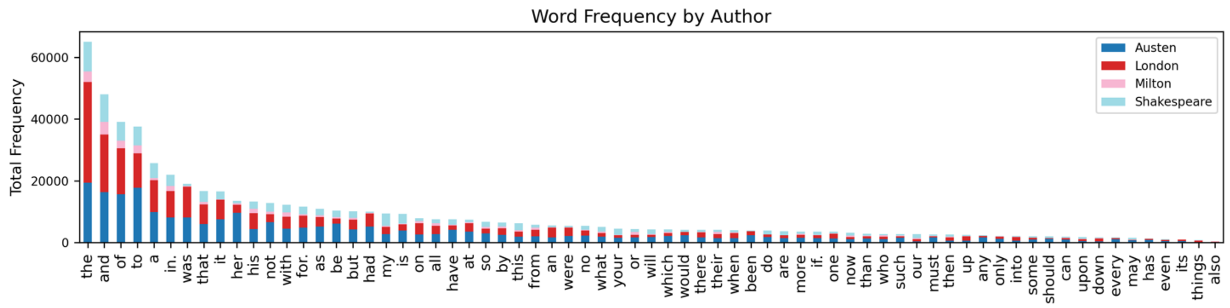


The linear methods perform worse in creating distinguishable clusters (without labels to validate PCA, MDS, and NMF would just look like "blobs" of observations) leading to more overlap and less informative projections; the structure of the data appears to be non-linear, evidenced by the superior performance 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***.

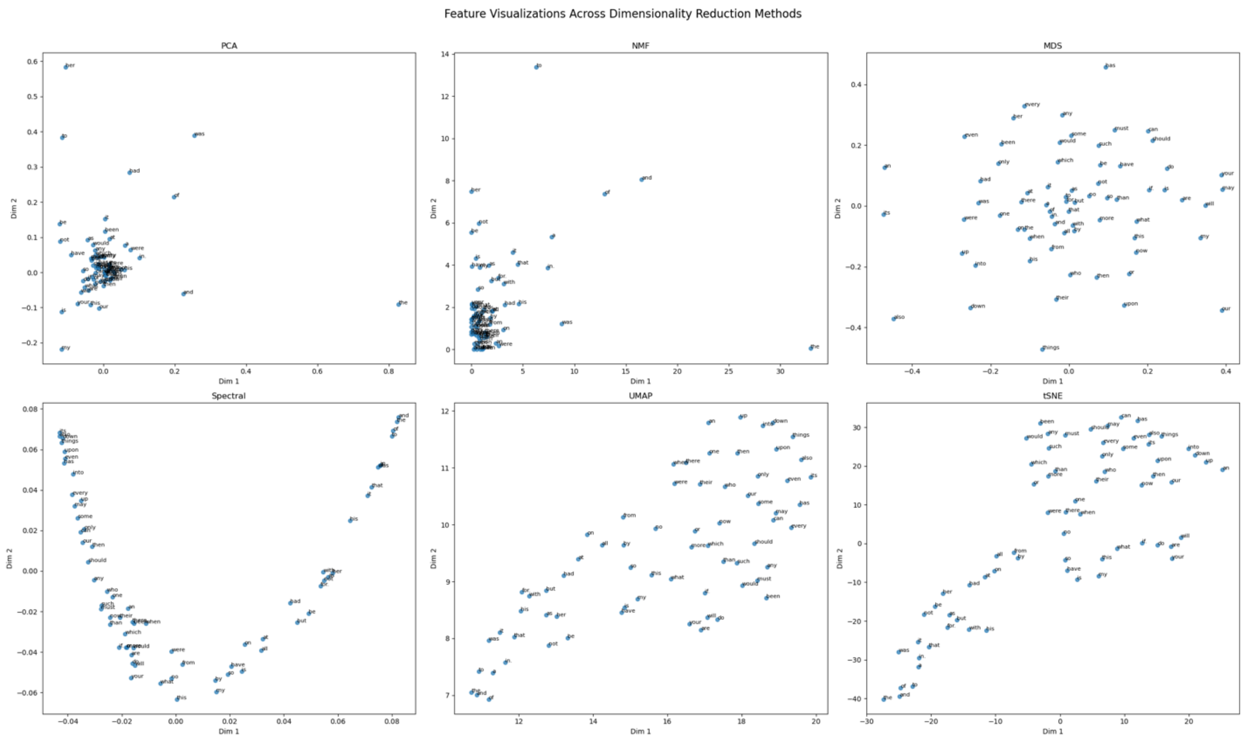
■ *features (words)*

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

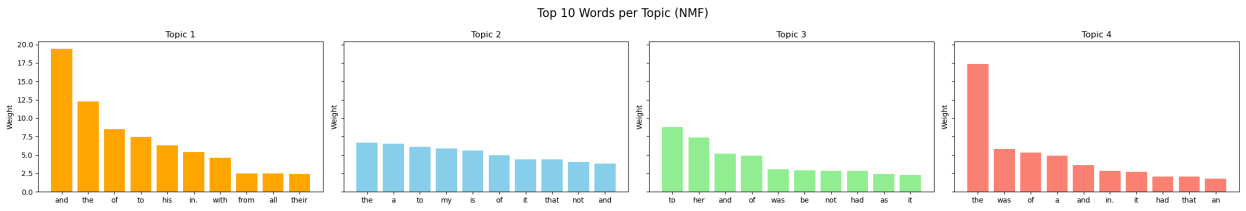


To further analyze patterns among features, I used their co-occurrences across chapters, without knowledge of authorship, to explore whether semantically or stylistically coherent word groups emerge. NMF is particularly well-suited for analyzing features (words) in an unsupervised setting because it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

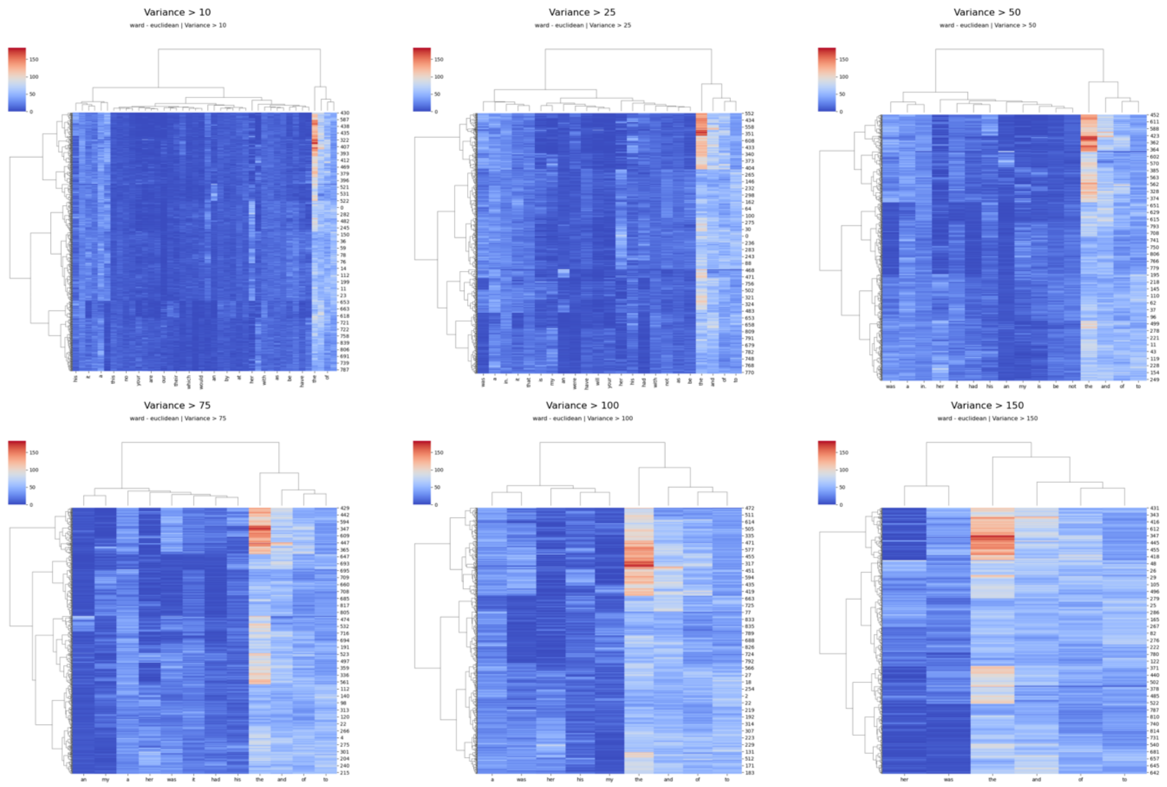


Using this it is far easier to establish semantic themes and distinguish the chapters from another according to which chapters are composed of which topics! The four topics extracted via NMF reveal distinct semantic patterns. **Topic 1** appears to reflect structural scaffolding of narrative prose, with 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.

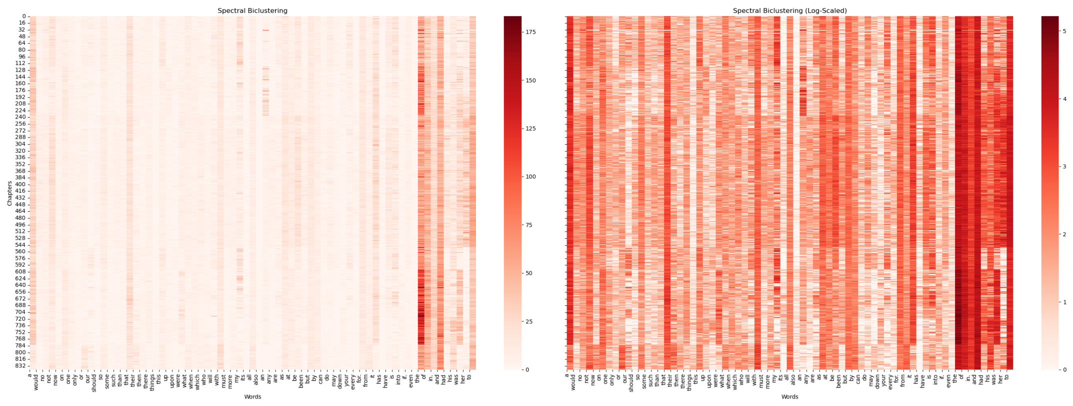
In terms of interpretability, the ***'best/great' interpretation*** is provided by ***NMF***. Now this leads to **1Ac** where we tie together these trends in words to the features which provides further value to interpretation!

■ ***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!



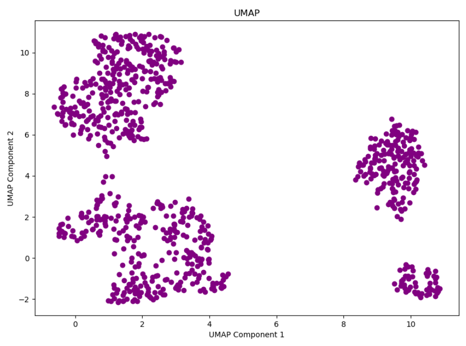
In other words, the features that have high contract among chapters allows us to interpret potential themes/topics among the chapters! I have also produced a heatmap using the Spectral Bi-clustering method which provided a decent visualization. However, clearly Hierarchical Bi-clustering outperforms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

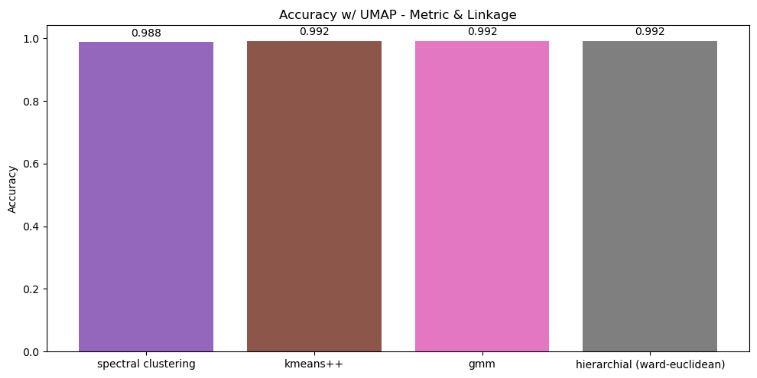


In terms of interpretability, the ***'best/great' interpretation*** is provided by ***Hierarchical Bi-clustering'***.

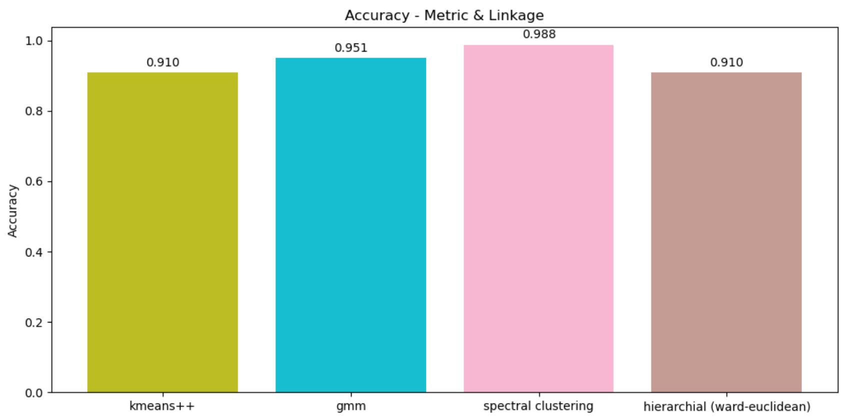
Clustering Methods

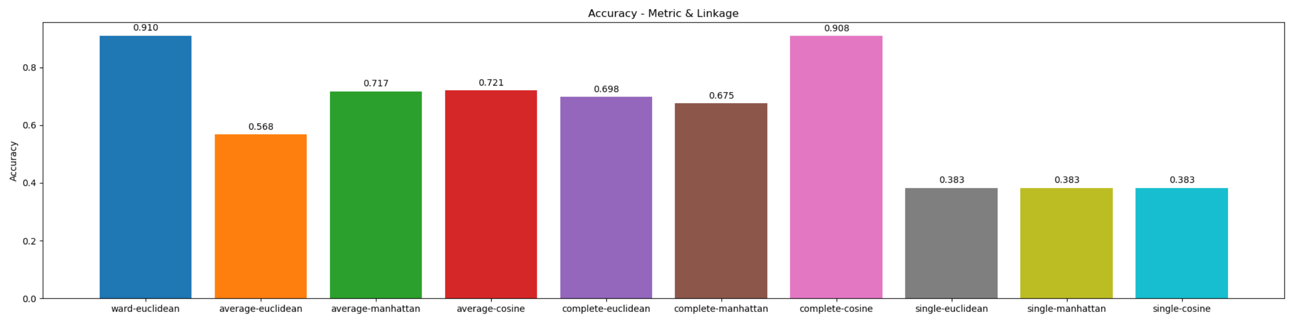
Since UMAP proved to be the superior dimension reduction technique for this dataset, I compared the clustering accuracy across methods on the UMAP transformed data (except for spectral clustering as this already has a built-in dimension reduction – spectral embedding). The results were not surprisingly 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.

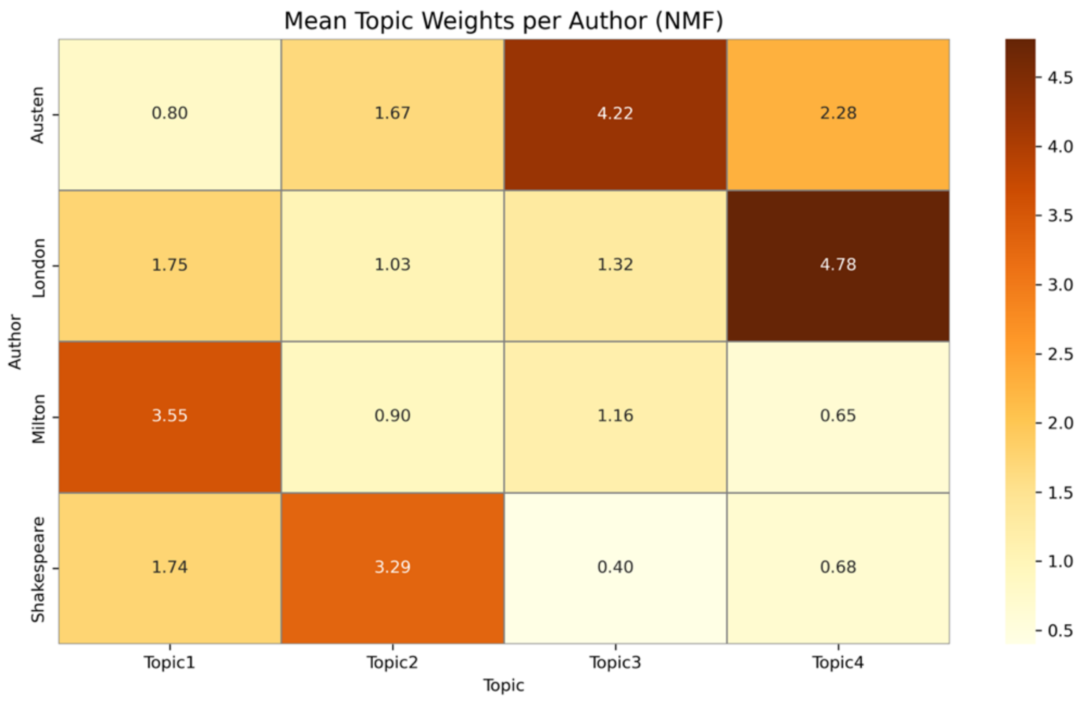




Without UMAP, performance dropped slightly yet still maintained strong results (of course spectral embedding yielded the same results). Among the metric and linkage parameters, Ward linkage with Euclidean distance yielded the highest accuracy, rivaling the performance of UMAP-based methods, while others 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'}
```

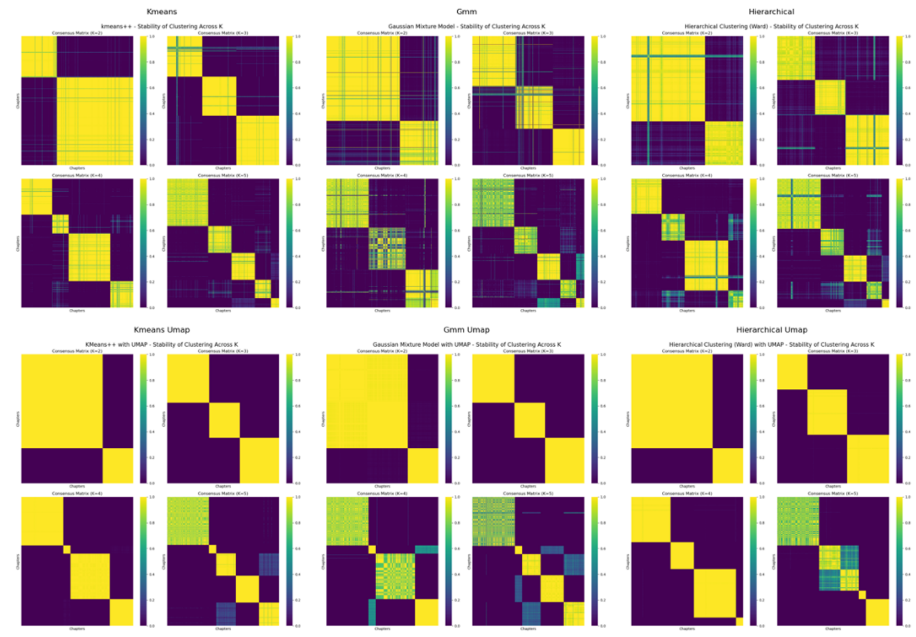
This representation suggests distinct stylistic or grammatical tendencies across authors. To evaluate how informative these patterns are, I split the data into **80% training** and **20% testing**, learned a mapping from dominant topic to author based on training data, and predicted authors on the test 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']
}
```

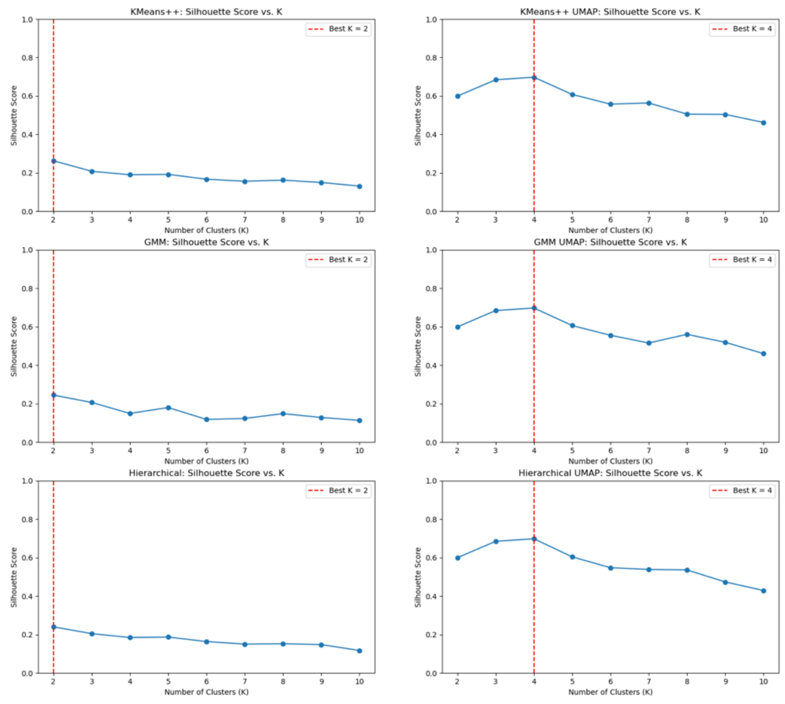
The words that are significant within topics and topics significant across authors provide predictive capabilities!

Validation

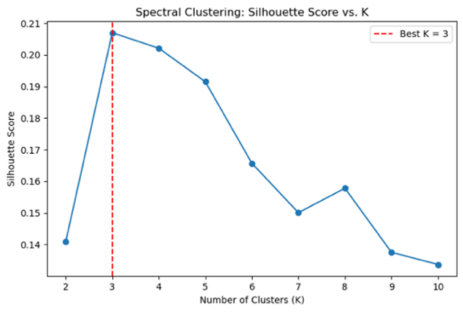
To validate our hyperparameter K without author labels, I used generalizability and stability as metrics.

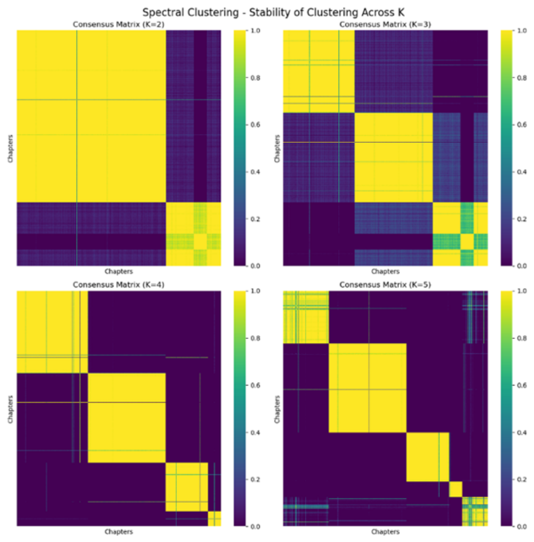


To gain insights to the generalizability, I calculated the silhouette scores across all methods (with and without the dimension reduction UMAP). From the visualization (right) you can see that before applying UMAP the silhouette score was very low across all K indicating but after applying UMAP, the 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.

<u>EM Algorithm</u>

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

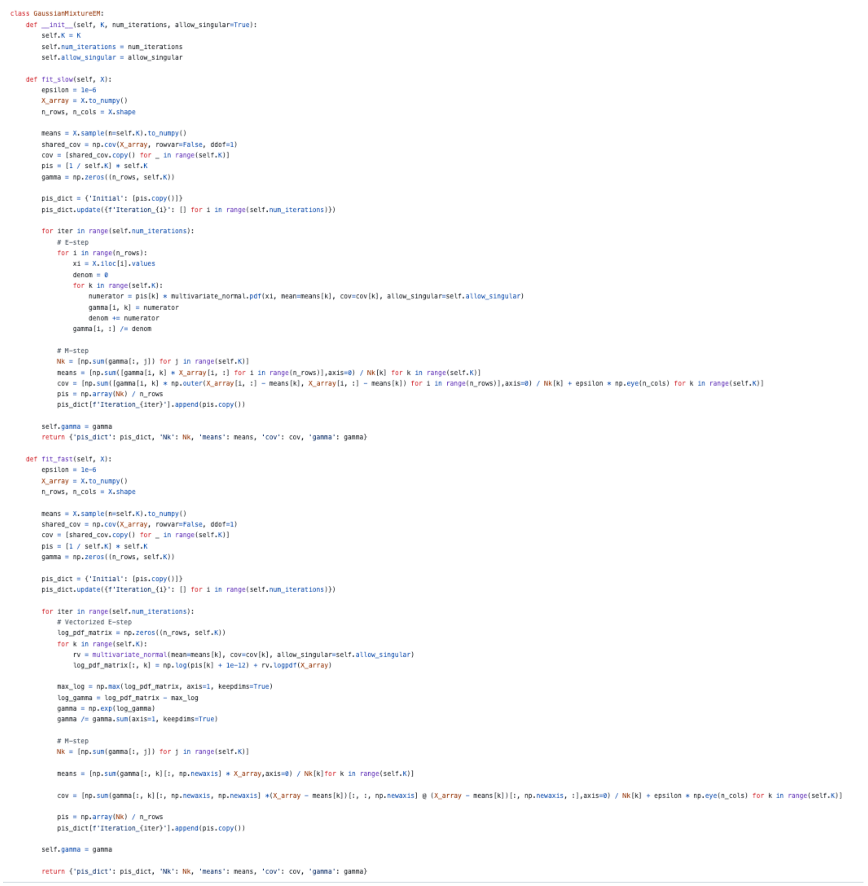
Effective number of points (soft cluster count): $N_k = \sum_{i=1}^{N} \gamma_{ik}$

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: $\pi_k = \frac{N_k}{N}$

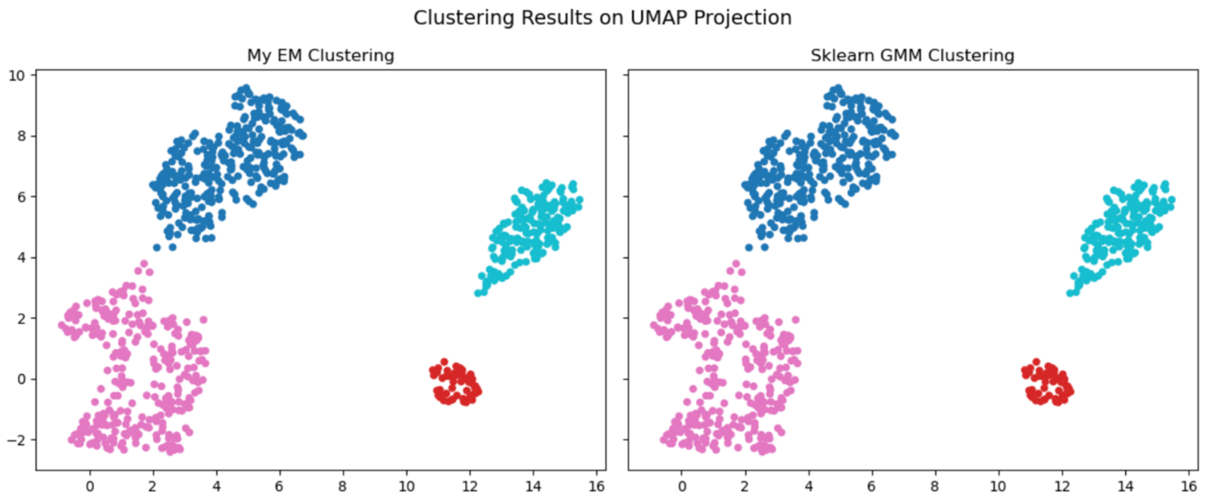
Running this on the authors algorithm on the 00_authors.csv dataset with the following code (fit_slow uses for loops and then I vectorized it using numpy for speed into fit_fast). See code here (alternatively I have attached a screenshot below).



```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!