Clustering

Niels Van den Broeck

June 12, 2025

1 Introduction

In this assignment, I created a Jupyter Notebook to walk through the process of testing different clustering setups, like testing text representations, clustering algorithms, distance functions, the amount of clusters, and other hyperparameters. After that, the best scoring combination is applied to cluster ids which are present in the given clusters file. Although the fundamentals are explained in the notebook, I will tell more about the reasoning and discuss the results in this report.

2 Data Exploration and Preprocessing

Since we will cluster reviews based solely on the text itself, meaning no rating, info about the user, or other indications of bias, we will carefully preprocess the text to be able to extract the right meaning of the review. Firstly, we concatenated the summary and the text of the review. After inspecting some reviews, it was clear that a lot of html tags (especially `<br>`) are present in the text. We removed this by simply applying a regular expression `"<.*?>"` and replacing it with `" "`.

Next, we will make the text more abstract by turning everything into lowercase letters, as well as removing all numbers and special characters. We removed stopwords using the NLTK library, as well as lemmatizing the whole text. This way, the clustering algorithm can mainly get critical information out of the important words, instead of either making useless clusters based on stop words, or have no clue what to cluster because for example "improving" and "improvement" are not the same word. An example of the text transformation is shown below.

Original Review

Crunchy & Good Gluten-Free Sandwich Cookies! Having tried a couple of other brands of gluten-free sandwich cookies, these are the best of the bunch. They're crunchy and true to the texture of the other "real" cookies that aren't gluten-free. Some might think that the filling makes them a bit too sweet, but for me that just means I've satisfied my sweet tooth sooner! The chocolate version from Glutino is just as good and has a true "chocolatey" taste - something that isn't there with the other gluten-free brands out there.

Cleaned Review

crunchy good sandwich cookie tried couple brand sandwich cookie best bunch crunchy true texture real cookie might think filling make bit sweet mean satisfied sweet tooth sooner chocolate version glutino good true chocolate taste something brand

Now to better understand and process text data, we apply three methods. Each approach offers a different perspective on how text can be represented and analyzed.

2.1 Bag of Words (BoW)

In the Bag of Words model, each document is represented as a vector of word counts. This approach ignores word order and context, but it's simple and effective for many traditional methods. We convert the cleaned review into numerical vectors where every column corresponds to a word in the vocabulary. Despite its simplicity, BoW can capture basic patterns in word usage.

2.2 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a deep learning model that generates contextual embeddings for text. Instead of representing words independently, BERT takes the entire sentence into account, producing embeddings that reflect meaning and word usage in context. Here, we use te pre-trained BERT model "all-MiniLM-L6-v2" to convert each review into a fixed-size embedding.

2.3 UMAP on BERT Embeddings

Because clustering directly on high-dimensional BERT embeddings was slow and less efficient, we first reduced the dimensionality using UMAP (Uniform Manifold Approximation and Projection). In this case, we reduced the 384-dimensional embeddings to 10 dimensions. This significantly sped up the clustering process while preserving the most important structure in the data. UMAP helps make clustering algorithms like KMeans more scalable and effective on dense embedding representations.

3 Clustering and Evaluation

3.1 Clustering Algorithms

First of all, some clustering algorithms are introduced:

1. **KMeans** – Partitions the data into k clusters by minimizing the variance within each cluster.
2. **Agglomerative** – A hierarchical method that repeatedly merges the two closest clusters until all points are grouped.
3. **GMM (Gaussian Mixture Model)** – Models the data as a mixture of several Gaussian distributions and assigns probabilities of belonging to each cluster.
4. **Spectral Clustering** – Uses the eigenvalues of a similarity matrix to perform dimensionality reduction before clustering.
5. **DBSCAN** – Groups together densely packed points and marks points in low-density regions as outliers.

Each of these algorithms has its own hyperparameters that we can experiment with. The most common is k, the number of clusters. We vary k between 4, 6, 8 and 10, to evaluate the impact on performance. Additionally we vary other parameters and run all possible combinations of parameter values, which leads to an exponential increase in total runs. Despite having limited computational resources, I was able to run a total of 192 hyperparameter combinations across all clustering algorithms in approximately 3 hours. An overview of the different parameters tested is shown below.

Overview of Clustering Parameters

KMeans

Parameter	*Values*
k (number of clusters)	4, 6, 8, 10

GMM (Gaussian Mixture Model)

Parameter	*Values*
k (number of clusters)	4, 6, 8, 10

Agglomerative Clustering

Parameter	*Values*
k (number of clusters)	4, 6, 8, 10
linkage	complete, average, single
metric	euclidean, manhattan, cosine

Spectral Clustering

Parameter	*Values*
k (number of clusters)	4, 6, 8, 10
affinity	rbf, nearest_neighbors

DBSCAN

Parameter	*Values*
eps_value	0.3, 0.5, 0.7
min_samples	3, 8
metric	euclidean, manhattan

3.2 Evaluation: Silhouette score

To evaluate certain configurations, we used the silhouette score. The silhouette score is a metric that measures how well each data point fits within its assigned cluster compared to other clusters. It returns a value between -1 and 1. A value close to 1 means that all points are well-matched to its cluster and poorly matched to neighboring clusters (ideal). A value closer to -1 suggests that the points may have been assigned to the wrong clusters.

4 Results

The results discussed in this report are present in Table 1. Note that this is only a fraction of all tested combinations. The full results are stored in the csv called "performance_results.csv".

The first thing we can take a look at is which combination of preprocessing, clustering algorithm, and hyperparameters perform the best. When using Bag of Words and agglomerative clustering, the silhouette score sits around 0.70. If we apply average linkage with the euclidean distance as distance metric and 4 clusters, an astonishing score of 0.768 is achieved. This is the highest score of all experiments. We can see that the agglomerative clustering algorithm performs better for all preprocessing methods with . On the other hand, dbscan performs remarkably worse in all cases while kmeans and gmm seem to perform somewhat in between. If we look at the different methods of preprocessing, BERT has by far the worst results. We assume that this is the result of the too high-dimensional embeddings. Lowering the embeddings using UMAP enormously improves the performance, going from 0.103 with BERT and agglomerative in the best case, to 0.516 using UMAP and agglomerative. For other clustering methods like kmeans and gmm, UMAP has some promising results, which was not the case with Bag of Words.

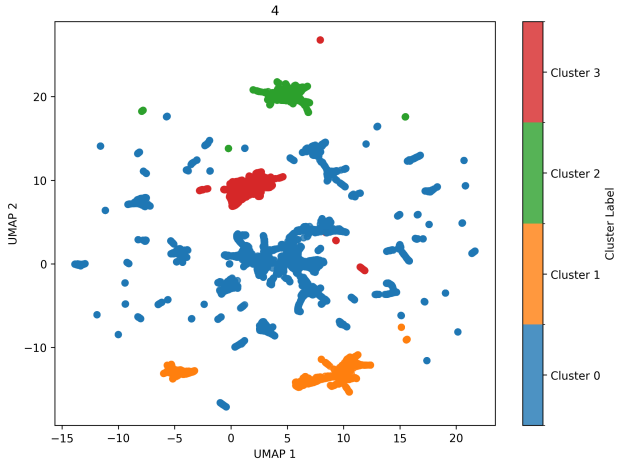

Figure 1: Agglomerative clustering using UMAP, k=4, linkage="average", metric="manhattan"

If we look at the common words/topics/patterns in the clusters, we can see that words like "Good", "Love", and "Great" are clustered together. In other clusters there are more topic like similarities like "Coffee", "Cream, and "Water".

Lastly, we will use the best scoring model to apply clustering on the dataset and fill in the cluster.csv file with our predicted clusters.

Table 1: Clustering results

data_name	model_name	parameters	score
BoW	kmeans	'k': 4	0.165000
BoW	kmeans	'k': 6	0.160000
BoW	kmeans	'k': 8	0.158000
BoW	kmeans	'k': 10	0.160000
BoW	gmm	'k': 4	0.245000
BoW	gmm	'k': 6	0.113000
BoW	gmm	'k': 8	0.096000
BoW	gmm	'k': 10	0.104000
BoW	**agglomerative**	**'k': 4, 'linkage': 'average', 'metric': 'euclidean'**	**0.768000**
BoW	agglomerative	'k': 4, 'linkage': 'complete', 'metric': 'manhattan'	0.755000
BoW	agglomerative	'k': 4, 'linkage': 'average', 'metric': 'manhattan'	0.693000
BoW	agglomerative	'k': 4, 'linkage': 'single', 'metric': 'euclidean'	0.765000
BoW	agglomerative	'k': 6, 'linkage': 'complete', 'metric': 'euclidean'	0.125000
BoW	agglomerative	'k': 6, 'linkage': 'single', 'metric': 'euclidean'	0.738000
BoW	agglomerative	'k': 8, 'linkage': 'complete', 'metric': 'euclidean'	0.112000
BoW	agglomerative	'k': 8, 'linkage': 'average', 'metric': 'euclidean'	0.737000
BoW	agglomerative	'k': 10, 'linkage': 'complete', 'metric': 'manhattan'	0.641000
BoW	agglomerative	'k': 10, 'linkage': 'average', 'metric': 'euclidean'	0.720000
BoW	agglomerative	'k': 10, 'linkage': 'single', 'metric': 'euclidean'	0.718000
BoW	agglomerative	'k': 10, 'linkage': 'single', 'metric': 'manhattan'	0.693000
BoW	dbscan	'eps_value': 0.3, 'min_samples': 3, 'metric': 'euclidean'	-0.222000
BERT	kmeans	'k': 4	0.051000
BERT	kmeans	'k': 6	0.049000
BERT	kmeans	'k': 8	0.048000
BERT	kmeans	'k': 10	0.046000
BERT	gmm	'k': 4	0.050000

BERT	gmm	'k': 6	0.050000
BERT	gmm	'k': 8	0.049000
BERT	gmm	'k': 10	0.045000
BERT	agglomerative	'k': 4, 'linkage': 'average', 'metric': 'cosine'	0.103000
BERT	agglomerative	'k': 4, 'linkage': 'single', 'metric': 'euclidean'	0.088000
BERT	agglomerative	'k': 4, 'linkage': 'single', 'metric': 'manhattan'	0.088000
BERT	agglomerative	'k': 10, 'linkage': 'single', 'metric': 'euclidean'	0.065000
BERT	agglomerative	'k': 10, 'linkage': 'single', 'metric': 'manhattan'	0.066000
BERT	agglomerative	'k': 10, 'linkage': 'single', 'metric': 'cosine'	0.065000
BERT	spectral	'k': 4, 'affinity': 'rbf'	0.047000
BERT	spectral	'k': 4, 'affinity': 'nearest_neighbors'	0.047000
BERT	spectral	'k': 10, 'affinity': 'rbf'	0.042000
BERT	spectral	'k': 10, 'affinity': 'nearest_neighbors'	0.025000
BERT	dbscan	'eps_value': 0.3, 'min_samples': 3, 'metric': 'euclidean'	-0.113000
BERT	dbscan	'eps_value': 0.3, 'min_samples': 3, 'metric': 'manhattan'	-0.104000
BERT	dbscan	'eps_value': 0.7, 'min_samples': 8, 'metric': 'euclidean'	-0.130000
UMAP	kmeans	'k': 4	0.372000
UMAP	kmeans	'k': 6	0.434000
UMAP	kmeans	'k': 8	0.455000
UMAP	kmeans	'k': 10	0.405000
UMAP	gmm	'k': 4	0.509000
UMAP	gmm	'k': 6	0.429000
UMAP	gmm	'k': 8	0.465000
UMAP	gmm	'k': 10	0.424000
UMAP	agglomerative	'k': 4, 'linkage': 'complete', 'metric': 'cosine'	0.516000
UMAP	agglomerative	'k': 4, 'linkage': 'average', 'metric': 'euclidean'	0.500000
UMAP	agglomerative	'k': 4, 'linkage': 'average', 'metric': 'manhattan'	0.521000
UMAP	agglomerative	'k': 4, 'linkage': 'average', 'metric': 'cosine'	0.523000
UMAP	agglomerative	'k': 6, 'linkage': 'complete', 'metric': 'euclidean'	0.433000
UMAP	agglomerative	'k': 6, 'linkage': 'complete', 'metric': 'manhattan'	0.423000
UMAP	agglomerative	'k': 6, 'linkage': 'complete', 'metric': 'cosine'	0.414000
UMAP	agglomerative	'k': 10, 'linkage': 'average', 'metric': 'euclidean'	0.445000
UMAP	agglomerative	'k': 10, 'linkage': 'average', 'metric': 'manhattan'	0.413000
UMAP	agglomerative	'k': 10, 'linkage': 'average', 'metric': 'cosine'	0.443000
UMAP	spectral	'k': 6, 'affinity': 'rbf'	0.415000
UMAP	spectral	'k': 6, 'affinity': 'nearest_neighbors'	0.028000
UMAP	spectral	'k': 8, 'affinity': 'rbf'	0.430000
UMAP	spectral	'k': 8, 'affinity': 'nearest_neighbors'	-0.040000
UMAP	spectral	'k': 10, 'affinity': 'rbf'	0.422000
UMAP	spectral	'k': 10, 'affinity': 'nearest_neighbors'	0.029000
UMAP	dbscan	'eps_value': 0.3, 'min_samples': 8, 'metric': 'euclidean'	0.061000
UMAP	dbscan	'eps_value': 0.3, 'min_samples': 8, 'metric': 'manhattan'	-0.198000
UMAP	dbscan	'eps_value': 0.7, 'min_samples': 3, 'metric': 'euclidean'	0.196000
UMAP	dbscan	'eps_value': 0.7, 'min_samples': 3, 'metric': 'manhattan'	-0.166000