UDC 004.912+519.237.8
DOI: 10.20535/SRIT.2308-8893.2026.2.10

SCALABLE TEXT CLUSTERING BASED ON WORD EMBEDDINGS AND NOISE ANALYSIS

D.O. SHUTIAK, G.B. PODKOLZIN, O.A. POKHYLENKO

Abstract. Text data clustering is a key component of unstructured text message analysis. To utilize these methods, text data must be converted into vector representations, i.e., word embeddings must be performed. This paper presents a modification of the HDBSCAN* clustering algorithm using custom distance metrics from the Minkowski family ($L1$, $L2$, $L\infty$) and parameters specifically tailored for clustering unstructured text data. A major contribution is a novel evaluation metric based on the relative point density of identified clusters and surrounding noise formations ("clouds"). Beyond assessing overall clustering quality, this metric highlights problematic dense accumulations within the noise that require additional manual analysis. Experimental evaluation on the "20 Newsgroups" dataset demonstrated that clustering quality is independent of the α parameter but highly sensitive to the distance metric, with $L\infty$ yielding the best results. The nomic-embedding-v1 model significantly outperformed gte-v1.5 in both the silhouette score and the proposed relative density metric.

Keywords: text clustering, word embedding, large language models, machine learning, Python.

INTRODUCTION

In text analysis, clustering holds significant potential precisely due to the embedding algorithms [1] applied to these texts. The vector spaces generated by these language models exhibit high-quality features, thus, employing various distance metrics and clustering models for subsequent classification yields robust spatial relationships between texts with similar or related content. The clusters constructed in this manner can later be used either alongside automatic summarization models to generate labels or categories for incoming texts, or directly to classify texts based on their cluster membership.

The development of large language models has also led to a proliferation of text embedding models, as this represents a core functionality of neural networks for text analysis. Models achieving top results on the MTEB (Massive Text Embedding Benchmark) [2] are typically very high-dimensional, meaning they require high memory usage and tend to be relatively slow. Therefore, this study focuses on comparing models that are significantly faster yet still provide high-quality embedding results. These include models such as: nomic-embedding-v1 [3], gte-v1.5 [4], and stella_en_400M_v5 [5].

This study focuses on and explores the hierarchical clustering algorithm HDBSCAN [6]. Further research is conducted on a modification of the HDBSCAN* clustering method, utilizing a custom metric and other parameters

specifically tailored for clustering unstructured text data. The study also applies a technique for extracting flat clustering based on cluster stability.

OVERALL DESCRIPTION OF THE HDBSCAN* ALGORITHM AND ITS GENERALIZATION

The HDBSCAN clustering method utilized in this study is based on the following approach. Initially, the set of points to be clustered is transformed into a connected graph based on the geometric properties of the points. In the next step, edges corresponding to points that are too far apart are removed. This process is iterative, resulting in a set of connected graph components that represent the expected clustering.

This method is then implemented using a generalization of the HDBSCAN* algorithm. Initially, the space of the original data is transformed according to its density and sparsity. The cluster hierarchy is then condensed based on the minimum cluster size. Stable clusters are selected from the condensed tree. A minimum spanning tree is constructed from the graph, weighted by distance. Clusters are identified based on density to separate noise. For density estimation, distance metrics are used.

We consider a generalized metric:

$$d_k(a,b) = max\{core_k(a), core_k(b), d(a,b)\},$$

where $core_k(a)$ represents the distance to the k-th nearest neighbor (distances from point a are sorted in ascending order); $d(a,b)$ represents the distance between points a and b. Different variations of this distance will be considered as one method of modifying the HDBSCAN* algorithm.

The structure of the algorithm involves finding clusters with higher density. Real-world data often contain outliers and corruption, which will be perceived as noise in the analysis.

The foundation of the algorithm is single-linkage clustering, which can be quite sensitive to noise: a single noisy data point may act as a “bridge” between high-density clusters, connecting them together. The algorithm must be robust to noise. Therefore, we employ the generalized distance $d_k(a,b)$ which sparsifies the noise without affecting the high-density clusters. A caveat here is that increasing k increases the number of points that are interpreted as noise.

The first step of the algorithm involves constructing a graph of the data, where the edge weights represent distances between points according to the chosen metric. The graph is constructed as follows: for each point x we find a radius $r_k(x)$ such that a circle with this radius contains a fixed number of points – k. In the study [7], a value of k = 100 was empirically shown to provide a well-balanced trade-off between cluster coherence and noise sensitivity in short-text clustering, and we adopt this setting in our algorithm. Next, for an increasing positive parameter r we select all points for which $r_k(x) \leq r$. Within this set of points, we connect the points with edges where the distance $\left|\left|x_i - x_j\right|\right| \leq \alpha r$. This process is iterated by increasing r. In this way, we obtain a connected graph from all our points. Our modification of the HDBSCAN* algorithm involves varying the parameter α and considering different distance metrics $d(a,b)$.

In the next step, we consider a threshold value, starting high and gradually lowering it. We discard any edges whose weight exceeds this threshold. By discarding edges in this manner, we divide the graph into connected components.

Thus, we construct a hierarchy of connected components for different threshold levels, where the threshold is the weight of the discarded edges.

Detailed Algorithm Description

A connected component of a graph consists of n^2 edges. To build an efficient algorithm, it's important to optimize and reduce the number of edge removal steps. This requires finding a minimal set of edges such that removing any edge from this set would disconnect components. Additionally, the set must not include any lower-weight edge that could reconnect those components.

Therefore, in the first step of the algorithm, we construct a minimum spanning tree of the graph, weighted by distance, using Prim's algorithm [8], following the original HDBSCAN* formulation [9], in which Prim's algorithm is used to compute the MST of the mutual reachability graph. We build the tree edge by edge, always adding the edge of the smallest weight that connects the current tree to a vertex not yet included in it. In the end, we obtain a single tree – a hierarchy covering all the data points, not yet separated into distinct clusters. Hence, this resulting tree must be further processed to extract the actual clusters. After this, the second step of the algorithm begins.

Initially, we define the minimum cluster size – a key parameter in HDBSCAN. With this value specified, we can now traverse the hierarchy and, during each split, check the number of points in the resulting clusters. If one of the resulting parts has fewer points than the minimum cluster size, it is treated as part of the parent cluster (i.e., the cluster before the split in the hierarchy) and is not considered an independent cluster.

Later, we distinguish true clusters from isolated points based on their stability during the splitting process. Clusters that persist through many levels of the hierarchy are considered valid and expected, while short-lived clusters are likely artifacts of the method. If, at a certain step during splitting, we obtain both components that have fewer points than the minimum cluster size, then that cluster is no longer subject to further splitting. Another possible situation is as follows: one of the two components after a split has fewer points than the minimum cluster size, while the second component is further split at a later step into two or more parts, each of which satisfies the minimum size condition. In this case, the smaller first component is considered noise, i.e., points not belonging to any cluster in the sense of density-based clustering [9].

We now formalize the stability of a cluster with respect to splitting. To do this, we need a measure other than distance – one that captures cluster persistence. For this, we will use $\lambda = \frac{1}{distance}$ where *distance* is the weight of the edge in the graph. For a given cluster, we can define two values: λ_{birth} – the lambda value at which the cluster emerges during the split of its parent cluster and λ_{death} – the lambda value at which the cluster itself splits into two parts that are either both smaller than the minimum cluster size, or both larger. For points p belonging to the portion of the cluster which contains fewer points during the split, we define λ_p –

the value at which this point ceases to belong to the cluster (i.e., the point of the split). Since the point can only fall out of the cluster after the cluster has formed and before it has fully split, we have $\lambda_{birth} \leq \lambda_p \leq \lambda_{death}$.

Having these values, we can define a stability score for each cluster:

$$s_{cluster} = \sum_{p \in cluster} (\lambda_p - \lambda_{birth}).$$

Next, an iterative process is carried out to determine stable clusters, starting from the leaf nodes of the tree. For each cluster, its stability is computed and compared to the sum of the stabilities of its child clusters. If the sum of the stabilities of the child clusters is greater than the cluster's own stability, the parent cluster's stability is discarded and replaced with this sum for use in further steps. Conversely, if the cluster's own stability is greater than the sum of its children's stabilities, then this cluster is considered valid, and all its "descendants" cease to be considered as such, even if they were previously.

After completing the iteration up to the root node of the tree, all remaining valid clusters are selected and form the final clustering result. At this point, any data points that are not part of any selected cluster are labeled as noise.

The implementation of this algorithm supports the Minkowski distance family and uses references to standard libraries.

For any natural number m, the L_m distance between n-dimensional vectors $\vec{x}$ and $\vec{y}$ is defined as:

$$L_m(\vec{x}, \vec{y}) = \sqrt[m]{\sum_{i=1}^{n} |x_i - y_i|^m},$$

in the limiting case, the Chebyshev distance L_∞ is defined as:

$$L_\infty(\vec{x}, \vec{y}) = \max_{1 \leq i \leq n} |x_i - y_i|,$$

where x_i and y_i are the i-th components of $\vec{x}$ and $\vec{y}$, respectively.

Below is the code for computing the L_m distance, where m is any natural number:

```
d = 0.0
while i <= n:
    d = d + (fabs(X[i] - Y[i])) ** m
    i = i + 1
d = d ** (1 / m)
```

The code for computing the L_∞ distance:

```
d = 0.0
while i <= n:
    d = max(d, fabs(X_data[i] - Y_data[i]))
    i = i + 1
```

The code for clustering:

```
clf = hdbscan.HDBSCAN(min_cluster_size=100,
                      min_samples=10,
```

```
cluster_selection_epsilon=0.0,
                        metric='euclidean',  # set the
metric value to “l1”, “l2”,
                                             # “infinity”,
the code for which is given above

cluster_selection_method='eom',
                       alpha=1,  # iterate over the values for
alpha:
          # array([0.5  , 0.625, 0.75 , 0.875, 1.   , 1.125, 1.25 , 1.375,
1.5  , 1.625, 1.75 , 1.875, 2.   ])
                        prediction_data=True)
cluster_labels2 = clf.fit(u2)
dblabels2 = cluster_labels2.labels_
n_clusters2_ = len(set(dblabels2)) - (1 if -1 in
dblabels2 else 0)
n_noise2_ = list(dblabels2).count(-1)
```

METRICS FOR EVALUATING THE RESULTS OF THE ALGORITHM

To evaluate the results of the algorithm, we use the following metrics: Silhouette Score Algorithm and our custom metric based on relative cluster density. Let’s consider each in more detail.

Silhouette Score Algorithm

The silhouette score is a metric for evaluating the optimal number of clusters for a specific dataset. It is defined as:

$$S(x_i) = \frac{b(x_i) - a(x_i)}{\max\big(a(x_i), b(x_i)\big)},$$

where $S(x_i)$ – the silhouette score for a specific point x_i; $a(x_i) = \frac{1}{|C_I|-1}\sum_{j\in C_I, i\neq j} d(x_i, x_j)$ – is the average distance from x_i to other data points in the same cluster C_I; $b(x_i) = min_{J\neq I}\left\{\frac{1}{|C_J|}\sum_{j\in C_I} d(x_i, x_j)\right\}$ – is the smallest average distance from x_i to data points in another cluster C_J; $|C_I|$ – is the number of points in cluster C_I; $d(x_i, x_j)$ – is the distance between points x_i and x_j; $a(x_i)$ – reflects the cohesion of x_i with its own cluster; $b(x_i)$ – reflects the separation of x_i from its neighboring cluster; $a(x_i) \geq 0$ and $b(x_i) \geq 0$ are distances, so the denominator is always the larger of the two. From this expression it follows directly that $S(x_i) \in [-1, 1]$. A value of $S(x_i)$ close to 1 (when $a(x_i) \ll b(x_i)$) indicates that the data is properly grouped. Lower values suggest weaker clustering for that point.

We compute the average silhouette score for a cluster as: $S_J = \frac{1}{|C_J|}\sum_{j\in C_I} S(x_i)$.

This average value, S_J, represents how tightly grouped all points are within cluster C_J.

An implementation of this metric is available in standard libraries.

Metric Based on Relative Cluster Density

As a result of the clustering algorithm's execution, in addition to the constructed clusters, noise points remain, which we will treat as a special cluster C_{-1}.

Next, we consider C_k – the cluster closest to some cluster C_j. Let ρ_{kj} be the distance between these two clusters, defined as the smallest distance between any pair of points from the two clusters. We then consider the set of points that are within a distance of $\rho_{kj}/2$ from points in cluster C_j but do not belong to C_j itself. This set represents a portion of the noise cluster C_{-1} and we refer to it as the "cloud" of cluster C_j denoted as $N(C_j)$.

At each step, after constructing the cloud $N(C_j)$ we update the noise cluster C_{-1} by removing the cloud points assigned to $N(C_j)$.

Then we compute the distance matrix between all points in the union of the cluster and its cloud. The first part of the matrix rows corresponds to the cluster points, and the rest to the cloud points. For each point, we find the minimum distance to any other point. We define r_{ij} as half of this minimum distance, and use it as the radius of a sphere covering that point. The corresponding volume is calculated as: $V_{ij} = \frac{\pi^{n/2} \cdot {r_{ij}}^n}{\Gamma\left(\frac{n}{2}+1\right)}$. For example, in an 8-dimensional space, the volume becomes: $V_{ij} = \frac{\pi^4 {r_{ij}}^8}{120}$. We then compute the quasi-volume of a cluster as: $V(C_j) = \sum_i V_{ij}$, as the sum of all found volumes for the points of the cluster. The same approach is used to compute the quasi-volume of the cloud: $V(N(C_j))$.

Rather than solving the highly complex sphere packing problem, we approximate the volume occupied by a cluster or cloud by covering the point set with spheres. This quasi-volume offers a practical and efficient estimation of spatial density.

Now, we define the point density for the cluster and its cloud as: $\nu(C_j) = \frac{|C_j|}{V(C_j)}$ and $\nu(N(C_j)) = \frac{|N(C_j)|}{V(N(C_j))}$. Using these, we define the relative density as: $\mu_j = \frac{\nu(N(C_j))}{\nu(C_j)}$. Under typical conditions, μ_j lies in the range [0, 1]. The average relative density across all clusters is given by: $\mu_{avg} = \sum_j \frac{\mu_j}{n}$. The smaller the value of μ_{avg} the higher the quality of the clustering.

In practice, the analysis of clouds based on relative densities revealed that dense structures can form within the noise itself, causing $\mu > 1$ in some cases. To account for this, we perform a secondary clustering within the noise, using a smaller parameter $k = 3$. This allows us to identify small, dense structures inside the noise – these are treated as microclusters. We then repeat the cloud construction and relative density calculation process for both the main clusters and these newly detected microclusters.

The code for quasi-volume and density computation:

```
def compute_volumes(cluster_points, cloud_points):
  dimensions_count = cluster_points.shape[1]
```

```
  combined_points = np.vstack((cluster_points,
cloud_points))
  dist_matrix = distance_matrix(combined_points,
combined_points)
  np.fill_diagonal(dist_matrix, np.inf)
  num_cluster_points = len(cluster_points)
  radii = np.min(dist_matrix, axis=1) / 2.0
  volume_coeff = (np.pi ** (dimensions_count / 2.0)) /
gamma(dimensions_count / 2.0 + 1)
  cluster_volumes = np.sum(volume_coeff *
(radii[:num_cluster_points] ** dimensions_count))
  cloud_volumes = np.sum(volume_coeff *
(radii[num_cluster_points:] ** dimensions_count))
  return cluster_volumes, cloud_volumes
def compute_density(num_points, volume):
  return num_points / volume if volume > 0 else 0
```

The code for the relative density metric:

```
def compute_relative_density_metric(X, labels):
  unique_labels = np.unique(labels)
  clusters_labels = unique_labels[unique_labels >= 0]
  clusters_count = clusters_labels.shape[0]
  dimensions_count = X.shape[1]
  noise = X[labels == -1]
  clusters = []
  for label in clusters_labels:
    clusters.append(X[labels == label])
  relative_densities = []
  clouds = []
  for cluster in clusters:
    min_distance = float('inf')
    nearest_cluster = None
    for other_cluster in clusters:
      if np.array_equal(other_cluster, cluster):
        continue
      dist = np.min(distance_matrix(cluster,
other_cluster))
      if dist < min_distance:
        min_distance = dist
        nearest_cluster = other_cluster
    cloud_radius = min_distance / 2.0
    cloud_points = np.array([point for point in noise
if np.min(distance_matrix([point], cluster)) <=
cloud_radius])
    if cloud_points.shape[0] > 0:
      noise = np.array([point for point in noise if
point not in cloud_points])
    else:
      cloud_points = cloud_points.reshape(0,
cluster.shape[1])
    clouds.append(cloud_points)
```

```
    V_Cj, V_O_Cj = compute_volumes(cluster,
cloud_points)
    v_Cj = compute_density(len(cluster), V_Cj)
    v_O_Cj = compute_density(len(cloud_points),
V_O_Cj) if V_O_Cj > 0 else 0
    m_j = v_O_Cj / v_Cj
    relative_densities.append(m_j)
  relative_densities = np.array(relative_densities)
  return np.average(relative_densities),
np.max(relative_densities),
np.argmax(relative_densities)
```

The code for detecting microclusters:

```
def select_microclusters(noise, alpha=1.0,
metric='l1'):
  microclusters_clf =
hdbscan.HDBSCAN(min_cluster_size=3, metric=metric,
allow_single_cluster=True,
cluster_selection_method='eom', alpha=alpha,
prediction_data=True)
  microclusters_labels =
microclusters_clf.fit(noise).labels_
  return microclusters_labels
```

During the experiments, the test set "20 Newsgroups" [10] was used with the nomic-embedding-v1 model. Before clustering, the dimensionality of the vectors was reduced to 2 using the UMAP (Uniform Manifold Approximation and Projection) algorithm [11].

METRIC-BASED EVALUATION OF RESULTS

Results for L_1 distance

Table 1. Results for L_1 distance, without detection of microclusters

α	Number of clusters	Number of noise points	Silhouette	μ	
				μ_{avg}	μ_{max}
0.5	13	488	0.4583341	0.2483888	1.8959304
0.625	13	488	0.4583341	0.2483888	1.8959304
0.75	13	488	0.4583341	0.2483888	1.8959304
0.875	13	488	0.4583341	0.2483888	1.8959304
1.0	13	488	0.4583341	0.2483888	1.8959304
1.125	13	488	0.4583341	0.2483888	1.8959304
1.25	13	488	0.4583341	0.2483888	1.8959304
1.375	13	488	0.4583341	0.2483888	1.8959304
1.5	13	488	0.4583341	0.2483888	1.8959304
1.625	13	488	0.4583341	0.2483888	1.8959304
1.75	13	488	0.4583341	0.2483888	1.8959304
1.875	13	488	0.4583341	0.2483888	1.8959304
2.0	13	488	0.4583341	0.2483888	1.8959304

Table 2. Results for L_1 distance, with detection of microclusters

α	Number of		Number of noise points	Silhouette	μ	
	clusters	micro-			μ_{avg}	μ_{max}
0.5	13	33	81	0.2096269	0.0634202	1.2623597
0.625	13	33	81	0.2096269	0.0634202	1.2623597
0.75	13	33	81	0.2096269	0.0634202	1.2623597
0.875	13	33	81	0.2096269	0.0634202	1.2623597
1.0	13	33	81	0.2096269	0.0634202	1.2623597
1.125	13	33	81	0.2096269	0.0634202	1.2623597
1.25	13	33	81	0.2096269	0.0634202	1.2623597
1.375	13	33	81	0.2096269	0.0634202	1.2623597
1.5	13	33	81	0.2096269	0.0634202	1.2623597
1.625	13	33	81	0.2096269	0.0634202	1.2623597
1.75	13	33	81	0.2096269	0.0634202	1.2623597
1.875	13	33	81	0.2096269	0.0634202	1.2623597
2.0	13	33	81	0.2096269	0.0634202	1.2623597

Results for L_2 distance

Table 3. Results for L_2 distance, without detection of microclusters

α	Number of clusters	Number of noise points	Silhouette	μ	
				μ_{avg}	μ_{max}
0.5	12	398	0.4269198	0.1081406	0.3881788
0.625	12	398	0.4269198	0.1081406	0.3881788
0.75	12	398	0.4269198	0.1081406	0.3881788
0.875	12	398	0.4269198	0.1081406	0.3881788
1.0	12	398	0.4269198	0.1081406	0.3881788
1.125	12	398	0.4269198	0.1081406	0.3881788
1.25	12	398	0.4269198	0.1081406	0.3881788
1.375	12	398	0.4269198	0.1081406	0.3881788
1.5	12	398	0.4269198	0.1081406	0.3881788
1.625	12	398	0.4269198	0.1081406	0.3881788
1.75	12	398	0.4269198	0.1081406	0.3881788
1.875	12	398	0.4269198	0.1081406	0.3881788
2.0	12	398	0.4269198	0.1081406	0.3881788

Table 4. Results for L_2 distance, with detection of microclusters

α	Number of		Number of noise points	Silhouette	μ	
	clusters	micro-			μ_{avg}	μ_{max}
0.5	12	22	64	0.2885321	0.4568041	14.3478839
0.625	12	22	64	0.2885321	0.4568041	14.3478839
0.75	12	22	64	0.2885321	0.4568041	14.3478839
0.875	12	22	64	0.2885321	0.4568041	14.3478839
1.0	12	22	64	0.2885321	0.4568041	14.3478839
1.125	12	22	64	0.2885321	0.4568041	14.3478839
1.25	12	22	64	0.2885321	0.4568041	14.3478839
1.375	12	22	64	0.2885321	0.4568041	14.3478839
1.5	12	22	64	0.2885321	0.4568041	14.3478839
1.625	12	22	64	0.2885321	0.4568041	14.3478839
1.75	12	22	64	0.2885321	0.4568041	14.3478839
1.875	12	22	64	0.2885321	0.4568041	14.3478839
2.0	12	22	64	0.2885321	0.4568041	14.3478839

Results for L_∞ distance

Table 5. Results for L_∞ distance, without detection of microclusters

α	Number of clusters	Number of noise points	Silhouette	μ	
				μ_{avg}	μ_{max}
0.5	10	330	0.5597064	0.0439711	0.1670099
0.625	10	330	0.5597064	0.0439711	0.1670099
0.75	10	330	0.5597064	0.0439711	0.1670099
0.875	10	330	0.5597064	0.0439711	0.1670099
1.0	10	330	0.5597064	0.0439711	0.1670099
1.125	10	330	0.5597064	0.0439711	0.1670099
1.25	10	330	0.5597064	0.0439711	0.1670099
1.375	10	330	0.5597064	0.0439711	0.1670099
1.5	10	330	0.5597064	0.0439711	0.1670099
1.625	10	330	0.5597064	0.0439711	0.1670099
1.75	10	330	0.5597064	0.0439711	0.1670099
1.875	10	330	0.5597064	0.0439711	0.1670099
2.0	10	330	0.5597064	0.0439711	0.1670099

Table 6. Results for L_∞ distance, with detection of microclusters

α	Number of		Number of noise points	Silhouette	μ	
	clusters	micro-			μ_{avg}	μ_{max}
0.5	10	18	64	0.3375369	0.0391937	0.3494590
0.625	10	18	64	0.3375369	0.0391937	0.3494590
0.75	10	18	64	0.3375369	0.0391937	0.3494590
0.875	10	18	64	0.3375369	0.0391937	0.3494590
1.0	10	18	64	0.3375369	0.0391937	0.3494590
1.125	10	18	64	0.3375369	0.0391937	0.3494590
1.25	10	18	64	0.3375369	0.0391937	0.3494590
1.375	10	18	64	0.3375369	0.0391937	0.3494590
1.5	10	18	64	0.3375369	0.0391937	0.3494590
1.625	10	18	64	0.3375369	0.0391937	0.3494590
1.75	10	18	64	0.3375369	0.0391937	0.3494590
1.875	10	18	64	0.3375369	0.0391937	0.3494590
2.0	10	18	64	0.3375369	0.0391937	0.3494590

The results (Tables 1–6) revealed that for each distance, the number of clusters and noise points does not depend on alpha, but this number changes for different distances.

GRAPHICAL RESULTS

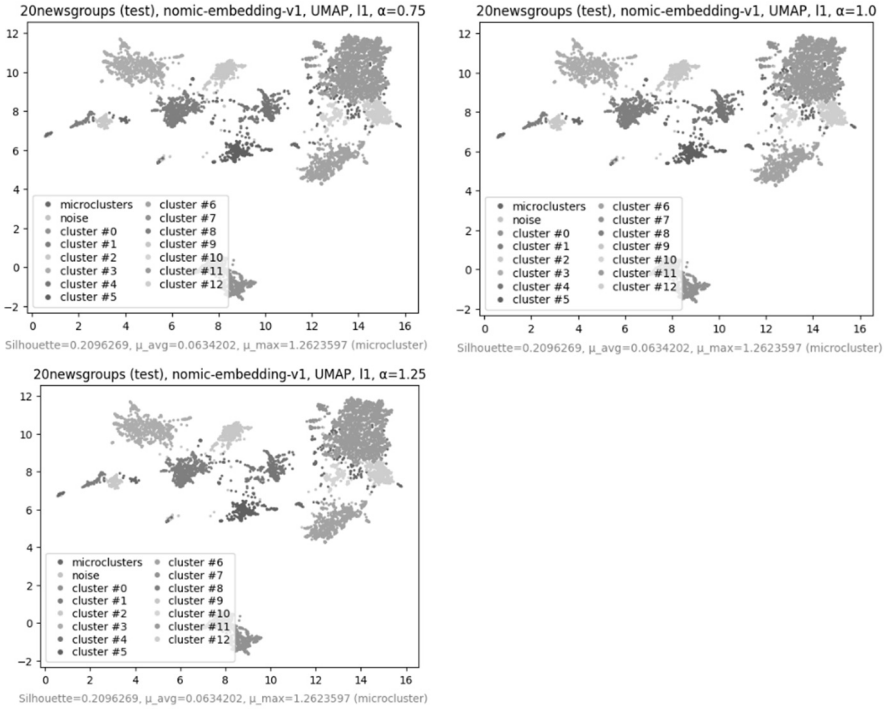

Fig. 1. Clustering with L_1 distance and α = 0.75; = 1.0; = 1.25

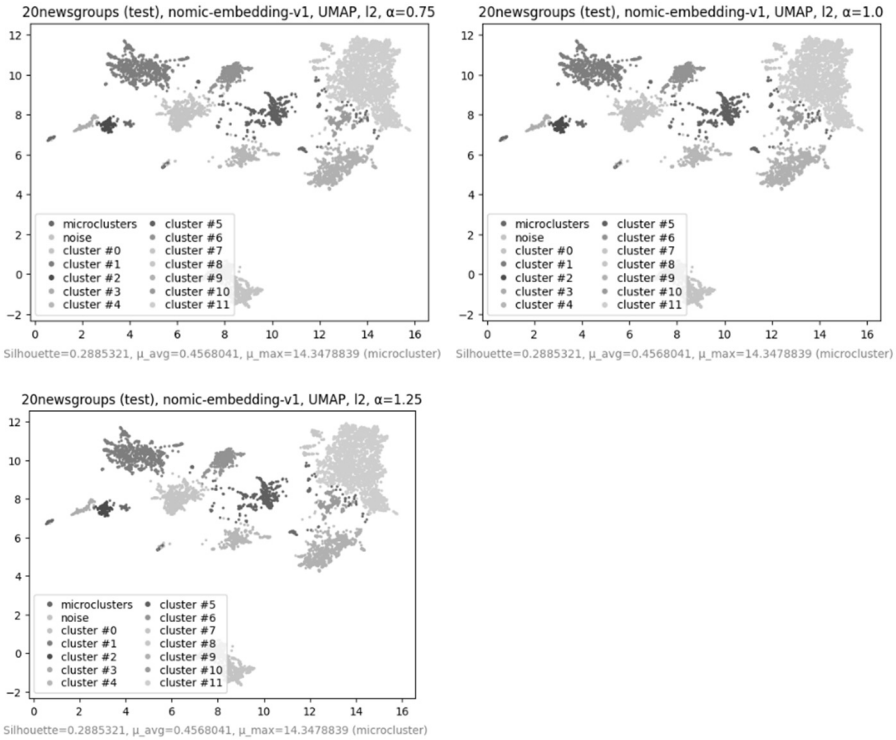

Fig. 2. Clustering with L_2 distance and α = 0.75; = 1.0; = 1.25

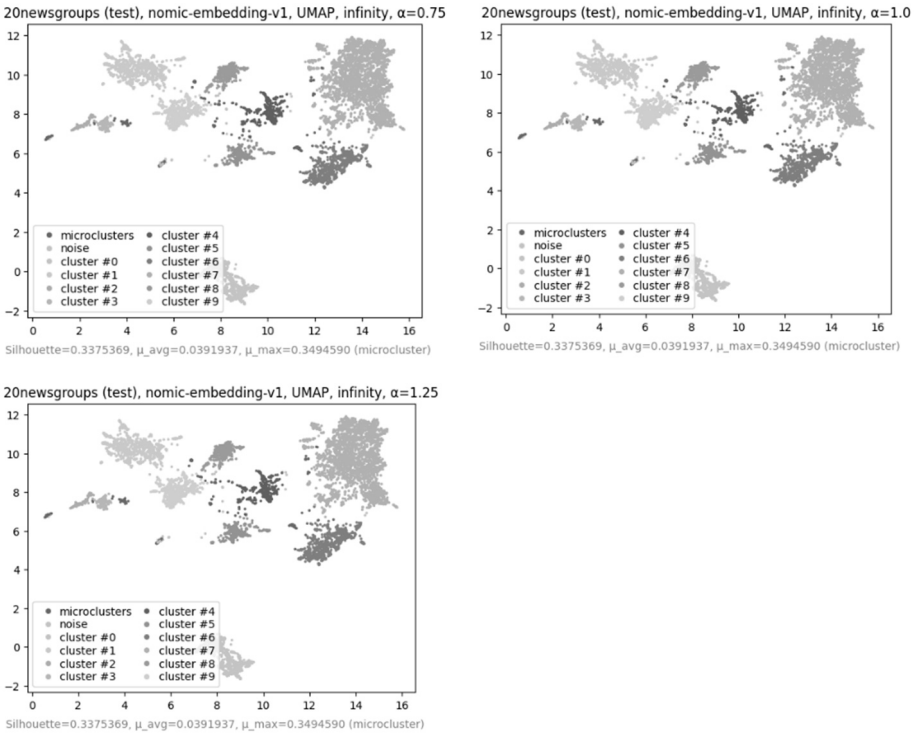

Fig. 3. Clustering with L_∞ distance and $\alpha = 0.75$; = 1.0; = 1.25

The analysis of the graphical results (Figs. 1–3) indicates that for any fixed distance metric, the clustering outcome does not depend on the choice of α: within each figure, the three subplots corresponding to α = 0.75, 1.0, and 1.25 exhibit identical cluster boundaries, cluster counts, and point-to-cluster assignments. At the same time, the clustering outcome is highly sensitive to the choice of distance metric: comparing figures at the same α (e.g., the first subplot of Figs. 1–3) reveals substantial differences in cluster structure.

RESULTS OF METRIC-BASED COMPARISON

The Silhouette Score metric evaluates clustering results as follows: a value close to 1 indicates that the data points are well-clustered, a lower score suggests poorer clustering performance. The score ranges from -1 to 1.

From the results Tables 1–6, we can observe that for each distance metric, changing alpha has no effect on the outcome. At the same time, the best results are achieved using the L_∞ distance, which is evident from the following: the silhouette score is the highest for L_∞, and the relative density μ is the lowest, indicating high-quality clustering.

Since the clustering quality does not depend on the choice of α, it is sufficient to set α to a constant value (e.g., α = 1) for further use of the algorithm.

Table 7. Metrics using nomic-embedding-v1 without microcluster detection for $\alpha = 1$

Distance	Silhouette	μ_{avg}	μ_{max}
L_1	0.4583340883255005	0.24838879949347553	1.895930397872204
L_2	0.42691975831985474	0.1081405606418731	0.388178799611063
L_∞	**0.5597063899040222**	**0.04397112285035522**	**0.16700987081107002**

T a b l e 8. Metrics using nomic-embedding-v1 with microcluster detection for $\alpha = 1$

Distance	Silhouette	μ_{avg}	μ_{max}
L_1	0.20962685346603394	0.0634201975689054	1.26235968775993
L_2	0.2885320782661438	0.45680405387564943	14.347883866030253
L_∞	**0.33753687143325806**	**0.0391937481804427**	**0.3494589734305932**

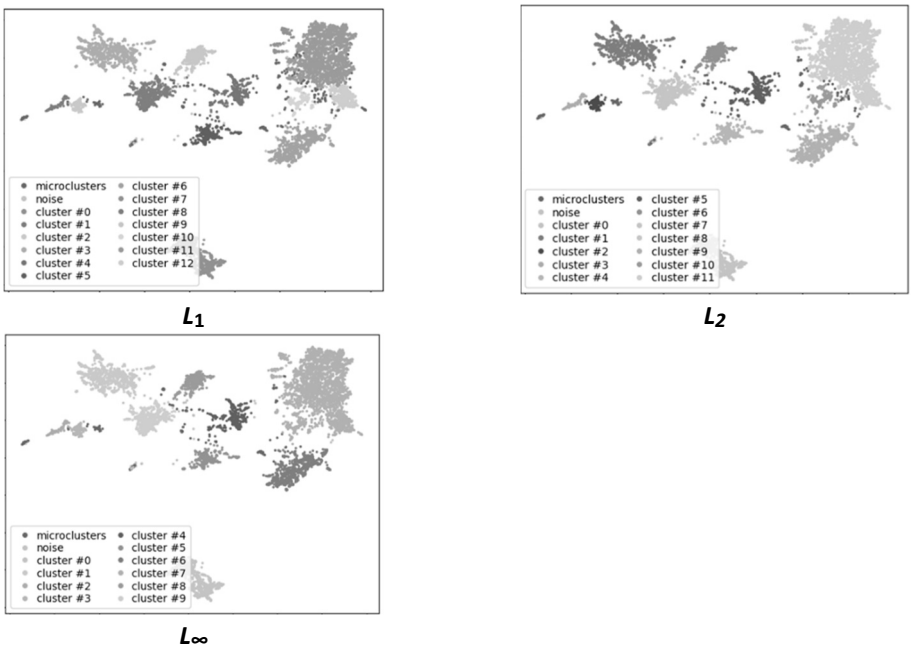

Fig. 4. Clustering using nomic-embedding-v1 with L_1, L_2, and L_∞ distances

From the Tables 7, 8 we can see that the best result is achieved for the L_∞ distance.

For comparison, let's present the silhouette score value calculated on the initially labeled clusters:

$$S(C) = 0.21342362.$$

As we can observe, this result is worse than the result produced by the clustering method. This can be explained by the fact that the data is strongly mixed, making it very difficult to formally separate into distinct clusters. The initially labeled clusters are shown in Fig. 5.

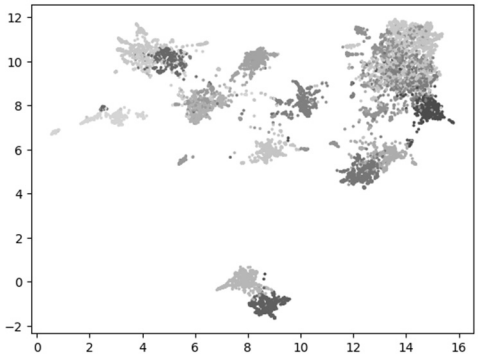

Fig. 5. Initially labeled clusters from the "20 Newsgroups" test set

Results Using the gte-v1.5 Text Embedding Model

In this set of experiments, the same test dataset was used, but the gte-v1.5 model was applied for text vectorization. As in the first experiment, dimensionality reduction to 2 dimensions was performed using UMAP prior to clustering.

Table 9. Metrics using gte-v1.5 without microcluster detection for $\alpha = 1$

Distance	Silhouette	μ_{avg}	μ_{max}
L_1	**0.4408782422542572**	0.4250197224211738	3.1405633170589624
L_2	0.39475032687187195	**0.24075141157025212**	**1.2032307329322984**
L_∞	0.4177089333534241	0.3099177220964534	1.4140965575553588

Table 10. Metrics using gte-v1.5 with microcluster detection for $\alpha = 1$

Distance	Silhouette	μ_{avg}	μ_{max}
L_1	0.10931075364351273	0.30896811431991195	13.283245660699313
L_2	0.0682852640748024	0.45744008384031115	26.70187725552937
L_∞	**0.12976054847240448**	**0.2800077887649127**	**9.137584489720576**

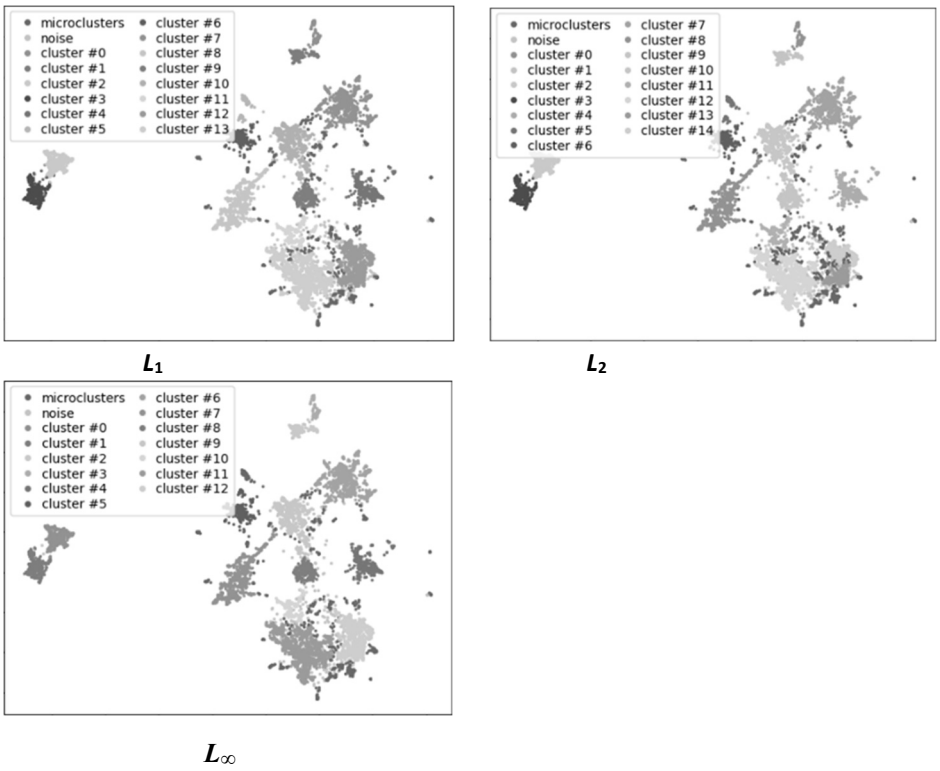

Fig. 6. Clustering using gte-v1.5 with L_1, L_2, and L_∞ distances

The obtained metric values differ significantly from those achieved using the nomic-embedding-v1 model. With gte-v1.5, the μ values are higher, and the silhouette scores are lower compared to the results obtained with the first model. See Tables 9, 10 and Fig. 6 for detailed metrics and clustering results across different distance metrics.

CONCLUSION

In the first stage of the algorithm, clustering was performed using a modification of the HDBSCAN* with a minimum cluster size parameter set to $k = 100$, following prior empirical recommendations. This produced an initial set of clusters. However, analysis based on the proposed relative density metric μ revealed that several high-density point formations remained among the unclustered (noise) data.

To address this, a second clustering pass was applied specifically to the noise using a smaller minimum cluster size parameter set to $k = 3$. This step enabled the identification of compact, dense formations that were not captured in the first stage – referred to in this study as microclusters. While the choice of k in this secondary step is heuristic and data-dependent, this two-stage approach allowed for a more nuanced interpretation of the dataset.

Nevertheless, even after this refinement, some microclusters exhibited high relative density values μ, with μ_{max} occasionally exceeding 1, which also affected the average relative density μ_{avg}. A μ value greater than 1 may indicate underlying issues in the data structure, suggesting that certain regions require manual inspection. These cases fall outside the capacity of the formal algorithm to resolve.

Furthermore, experimental results showed that the gte-v1.5 embedding model performed noticeably worse than nomic-embedding-v1, both in terms of the silhouette score and the relative density metric. This demonstrates the importance of selecting embedding models that preserve meaningful distance relationships for downstream clustering tasks.

Overall, the proposed relative density metric μ proved to be a practical and informative tool for evaluating clustering quality. It complements traditional metrics by capturing local density consistency and highlighting potentially problematic regions in the data, making it a valuable complement to conventional evaluation methods such as the silhouette score.

REFERENCES

1. A. Petukhova, J.P. Matos-Carvalho, N. Fachada, “Text Clustering with LLM Embeddings,” *arXiv preprint*, 2024. doi: https://doi.org/10.48550/arXiv.2403.15112
2. N. Muennighoff, N. Tazi, L. Magne, N. Reimers, “MTEB: Massive Text Embedding Benchmark,” *arXiv preprint*, 2022. doi: https://doi.org/10.48550/arXiv.2210.07316
3. Z. Nussbaum, J.X. Morris, B. Duderstadt, A. Mulyar, “Nomic Embed: Training a Reproducible Long Context Text Embedder,” *arXiv preprint*, 2024. doi: https://doi.org/10.48550/arXiv.2402.01613
4. L. Zehan, Z. Xin, Z. Yanzhao, L. Dingkun, X. Pengjun, Z. Meishan, “Towards General Text Embeddings with Multi-stage Contrastive Learning,” *arXiv preprint*, 2023. doi: https://doi.org/10.48550/arXiv.2308.03281
5. D. Zhang, J. Li, Z. Zeng, and F. Wang, “Jasper and Stella: distillation of SOTA embedding models,” *arXiv preprint*, 2025. doi: https://doi.org/10.48550/arXiv.2412.19048
6. C. Malzer, M. Baum, “A Hybrid Approach to Hierarchical Density-based Cluster Selection,” *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Karlsruhe, Germany, 2020*, pp. 223–228. doi: https://doi.org/10.1109/MFI49285.2020.9235263
7. Y. Feng, Z. Chen, Y. Zhang, W. Huang, X. Zhang, S. He, “BERTopic_Teen: A multi-module optimization approach for short text topic modeling in adolescent health,”

Frontiers in Public Health, vol. 13, p. 1608241, Aug. 2025. doi: https://doi.org/10.3389/fpubh.2025.1608241
8. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*; third edition. MIT Press, 2009, pp. 631–638.
9. R.J.G.B. Campello, P. Kröger, J. Sander, A. Zimek, "Density-based clustering", *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 2, p. e1343, 2020. doi: https://doi.org/10.1002/widm.1343
10. T. Mitchell, "Twenty Newsgroups," *UCI Machine Learning Repository*. doi: https://doi.org/10.24432/C5C323
11. L. McInnes, J. Healy, J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint*, 2020. doi: https://doi.org/10.48550/arXiv.1802.03426
12. R. Saha, "Influence of various text embeddings on clustering performance in NLP," *arXiv preprint*, 2023. doi: https://doi.org/10.48550/arXiv.2305.03144

Received 16.05.2025

INFORMATION ON THE ARTICLE

Dmytro O. Shutiak, ORCID: 0009-0008-6480-3706, World Data Center for Geoinformatics and Sustainable Development of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: dima.shutyak@gmail.com

Gleb B. Podkolzin, ORCID: 0000-0002-7120-2772, Educational and Research Institute for Applied System Analysis of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: podkolzin.gleb@lll.kpi.ua

Oleksandr A. Pokhylenko, ORCID: 0000-0002-1562-2051, Educational and Research Institute for Applied System Analysis of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: o.pokhylenko@kpi.ua

МАСШТАБОВАНА КЛАСТЕРИЗАЦІЯ ТЕКСТОВИХ ДАНИХ НА ОСНОВІ ВКЛАДЕННЯ СЛІВ ТА АНАЛІЗ ШУМУ / Д.О. Шутяк, Г.Б. Подколзін, О.А. Похиленко

Анотація. Кластеризація текстових даних є ключовою частиною аналізу неструктурованих текстових повідомлень. Для використання цих методів текстові дані потрібно перетворити у векторні подання, тобто провести вкладення слів. Подано модифікацію алгоритму кластеризації HDBSCAN* з використанням власних метрик відстані з родини Мінковського ($L1$, $L2$, $L\infty$) та параметрів, спеціально адаптованих для кластеризації неструктурованих текстових даних. Ключовим здобутком є нова метрика оцінювання, що ґрунтується на відносній щільності знайдених кластерів та навколишніх шумових утворень («хмар»). Окрім оцінювання загальної якості кластеризації, метрика вказує на проблемні щільні скупчення елементів у шумі, які потребують додаткового ручного аналізу. Експериментальне оцінювання на наборі даних «20 Newsgroups» показало, що якість кластеризації не залежить від параметра α, але є дуже чутливою до вибору метрики відстані, причому найкращі результати забезпечує $L\infty$. Модель nomic-embedding-v1 значно перевершила gte-v1.5 як за силуетним коефіцієнтом, так і за запропонованою метрикою відносної щільності.

Ключові слова: текстова кластеризація, вкладення слів, великі мовні моделі, машинне навчання, Python.