Introduction to Hierarchical Clustering in Information Retrieval

Part 1: Introduction to Hierarchical Clustering in Information Retrieval

In information retrieval (IR), clustering is used to group similar documents together, aiding in data organization and retrieval. There are two primary types of clustering: **flat clustering** and **hierarchical clustering**. Flat clustering produces an unstructured set of clusters, whereas **hierarchical clustering** generates a structured hierarchy of clusters, making it more informative and flexible in certain applications. The most common hierarchical clustering method in IR is **Hierarchical Agglomerative Clustering (HAC)**, where each document starts as its own cluster, and clusters are merged iteratively until a single cluster remains.

1.1 Key Concepts in Hierarchical Clustering

- **Flat vs. Hierarchical Clustering**: Flat clustering (e.g., K-means) is fast and efficient but lacks structure, requires the number of clusters to be specified beforehand, and may yield inconsistent results due to its nondeterministic nature. Hierarchical clustering, on the other hand, builds a tree of clusters without requiring the number of clusters to be prespecified and is usually deterministic.

- **Agglomerative Hierarchical Clustering (HAC)**: In HAC, each document starts as its own cluster, and clusters are merged based on a similarity measure until one large cluster remains. This is a **bottom-up approach**. The merging process is depicted as a **dendrogram**, which illustrates the merging steps and the similarity levels at which clusters are combined.

- **Dendrogram**: A graphical representation of the hierarchical merging of clusters, where each horizontal line represents a merge, and its y-coordinate shows the similarity at which the merge occurred. The structure of a dendrogram can help in visualizing how clusters are formed and in deciding the number of clusters by "cutting" the dendrogram at a desired similarity level.

1.2 Similarity Measures and Merging Criteria

Several algorithms within hierarchical clustering differ in the way they measure similarity between clusters:

- **Single-Link Clustering**: The similarity between two clusters is defined by the most similar pair of points (documents) between them.
- **Complete-Link Clustering**: The similarity is determined by the most dissimilar pair of points between clusters, making it a non-local criterion that results in more compact clusters.
- **Group-Average Clustering**: The similarity between clusters is based on the average of all pairwise similarities between points in the two clusters.
- **Centroid Clustering**: Similarity is computed based on the centroids (the mean points) of the clusters.

Numerical Example of Dendrogram and Similarity in Single-Link Clustering

Let's consider five documents {d1, d2, d3, d4, d5} with the following similarity matrix (cosine similarity):

	d1	**d2**	**d3**	**d4**	**d5**
d1	1.0	0.8	0.3	0.4	0.2
d2	0.8	1.0	0.2	0.5	0.3
d3	0.3	0.2	1.0	0.6	0.7
d4	0.4	0.5	0.6	1.0	0.4
d5	0.2	0.3	0.7	0.4	1.0

Table 1: Cosine similarity matrix between documents.

In **single-link clustering**, the first merge occurs between the two most similar points, which are d1 and d2 (with similarity 0.8). The next step is to merge the remaining clusters based on the most similar remaining points. This process continues until a single cluster remains.

Example Dendrogram Construction (Single-Link)

1. Step 1: Merge (d1, d2) since their similarity is 0.8.
2. Step 2: Merge (d3, d5) with similarity 0.7.
3. Step 3: Merge (d4, (d3, d5)) with similarity 0.6.
4. Step 4: Merge ((d1, d2), (d3, d4, d5)) with the next highest similarity.

This process builds a hierarchical structure that can be visualized through a dendrogram, showing how the clusters form over time based on pairwise similarities.

Conclusion

The subsequent parts will dive deeper into the various types of clustering criteria (single-link, complete-link, group-average, and centroid clustering), optimality conditions, and computational efficiency in hierarchical clustering, along with numerical illustrations and examples.

Part 2: Hierarchical Agglomerative Clustering (HAC) Algorithms and Their Variants

In this section, we will explore four specific types of **Hierarchical Agglomerative Clustering (HAC)** algorithms, each of which uses a different similarity measure to decide which clusters to merge. These methods are:

1. **Single-Link Clustering (Single-Linkage Clustering)**
2. **Complete-Link Clustering (Complete-Linkage Clustering)**
3. **Group-Average Clustering**
4. **Centroid Clustering**

2.1 Single-Link Clustering

In **single-link clustering**, the similarity between two clusters is defined as the similarity of their **closest pair** of documents. This method is also referred to as the **minimum spanning tree algorithm** because it connects clusters based on their nearest points.

Numerical Example

Consider the similarity matrix from Part 1 for five documents:

	d1	**d2**	**d3**	**d4**	**d5**
d1	1.0	0.8	0.3	0.4	0.2
d2	0.8	1.0	0.2	0.5	0.3
d3	0.3	0.2	1.0	0.6	0.7
d4	0.4	0.5	0.6	1.0	0.4
d5	0.2	0.3	0.7	0.4	1.0

Table 2: Cosine similarity matrix between documents.

- **Step 1**: The most similar documents are **d1** and **d2** (0.8 similarity), so we merge them into a single cluster.
- **Step 2**: Next, we merge the pair with the highest similarity, which are **d3** and **d5** (0.7 similarity).
- **Step 3**: The next highest similarity is 0.6 between **d4** and the newly formed cluster {d3, d5}.
- **Step 4**: Finally, we merge {d1, d2} with the remaining clusters.

This method tends to create elongated clusters, a phenomenon called **chaining**. Clusters are formed by connecting documents that are close to each other, but the overall structure of the cluster is ignored, which can lead to poorly structured clusters.

Strengths and Weaknesses

- **Strength**: The method is efficient and works well for clusters that form natural chains or sequences.
- **Weakness**: It can result in **chaining** (undesirable elongated clusters), where documents are loosely related but still grouped together.

2.2 Complete-Link Clustering

In **complete-link clustering**, the similarity between two clusters is defined by the similarity of their **most dissimilar pair** of documents. This method ensures that clusters remain compact by focusing on the points farthest apart within two clusters before merging them.

Numerical Example

Using the same similarity matrix, the process would proceed as follows:

- **Step 1**: The closest pair remains **d1** and **d2** (0.8), so we merge them.
- **Step 2**: The next most similar pair is **d3** and **d5** (0.7), so we merge them.
- **Step 3**: The remaining documents (**d4** and the new clusters) are now compared based on their **most distant points**. In this case, **d4** is added to the cluster {d3, d5} because it has the least dissimilarity from both **d3** and **d5**.

This approach tends to create **tight, spherical clusters** and avoids the chaining problem seen in single-link clustering.

Strengths and Weaknesses

- **Strength**: Complete-link clustering avoids chaining and produces compact clusters.
- **Weakness**: It is sensitive to **outliers**; a single outlier can significantly affect the structure of the cluster.

2.3 Group-Average Clustering

In **group-average clustering**, the similarity between two clusters is the **average similarity** between all pairs of documents, one from each cluster. This method strikes a balance between single-link and complete-link clustering by considering all points in the clusters.

Formula for Group-Average Similarity The similarity between two clusters C_1 and C_2 is calculated as:

$$\text{SIM-GA}(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{i \in C_1} \sum_{j \in C_2} \text{sim}(i, j)$$

Where:

- $|C_1|$ and $|C_2|$ are the number of documents in clusters C_1 and C_2, respectively.
- $\text{sim}(i, j)$ is the similarity between document i in C_1 and document j in C_2.

Numerical Example

Let's apply this to the same five-document dataset. Assume we merge **d1** and **d2** in the first step, and now we need to compute the similarity between this new cluster and **d3**:

$$\text{SIM-GA}(\{d1, d2\}, d3) = \frac{1}{2} \left(\text{sim}(d1, d3) + \text{sim}(d2, d3)\right) = \frac{1}{2} \left(0.3 + 0.2\right) = 0.25$$

The pair with the highest group-average similarity would be merged next.

Strengths and Weaknesses

- **Strength**: It balances local and global structure, making it a good general-purpose algorithm.
- **Weakness**: It is more computationally intensive than single-link or complete-link clustering, as it requires calculating the similarity between all pairs.

2.4 Centroid Clustering

Centroid clustering merges clusters based on the similarity of their centroids, or geometric centers. The centroid of a cluster is the average of all points (documents) within the cluster.

Formula for Centroid Similarity The similarity between clusters C_1 and C_2 based on their centroids μ_1 and μ_2 is:

$$\text{SIM-CENT}(C_1, C_2) = \mu_1 \cdot \mu_2$$

Where:

- μ_1 and μ_2 are the centroids of clusters C_1 and C_2.

Numerical Example

Let's calculate the centroid of a simple cluster {d1, d2}:

- The centroid is computed as the average of the feature vectors of **d1** and **d2**.

If each document is represented by a vector (e.g., word frequencies or embeddings), the centroid is simply the point-wise average of these vectors.

Strengths and Weaknesses

- **Strength**: It often produces well-rounded clusters that are easy to interpret.
- **Weakness**: Centroid clustering is not monotonic, meaning that similarity may increase after merging clusters, leading to **inversions** in the hierarchy.

Summary of the HAC Variants

Each of the four methods offers a different approach to hierarchical clustering:

Method	Merge Criterion	Typical Clusters	Pros	
Single-Link	Closest Points	Elongated Chains	Efficient, captures small connections	
Complete-Link	Farthest Points	Compact Spheres	Compact clusters	
Group-Average	Average Similarity	Balanced	Good balance between extremes	C
Centroid	Centroid Similarity	Balanced	Produces natural cluster centers	Non-

Table 3: Summary of HAC variants.

The next part will discuss **optimality in hierarchical clustering**, **efficiency considerations**, and the **divisive clustering** technique, with examples to explain these advanced concepts.

Part 3: Optimality, Time Complexity, and Divisive Clustering

In this part, we will explore the **optimality** conditions of hierarchical agglomerative clustering (HAC), its **time complexity**, and introduce **divisive clustering**, which is a top-down approach to hierarchical clustering. We will also compare these techniques and examine their advantages and disadvantages.

3.1 Optimality in Hierarchical Clustering

To assess the quality of a hierarchical clustering solution, we define **combination similarity**. This is the similarity between the two clusters that were merged at each step of the clustering process.

Optimality Definition: A hierarchical clustering solution is considered **optimal** if it maximizes the combination similarity at each stage of clustering. More formally, the combination similarity of a cluster ω formed by merging two clusters ω_1 and ω_2 is the similarity between ω_1 and ω_2.

For a set of clusters $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$, the clustering is optimal if no alternative clustering exists with higher combination similarity for the same number of clusters.

Example: In **single-link clustering**, optimality is achieved when the most similar points are always merged. However, **complete-link** and **group-average** clustering are not always optimal because they can suffer from **outliers** or merge decisions that might not maximize global similarity.

Non-monotonicity: **Centroid clustering** is known to be **non-monotonic**. Inversions can occur, where the similarity between clusters increases after a merge. This contradicts the assumption that smaller clusters are more coherent than larger clusters. For example, when merging clusters based on centroids, the resulting cluster might become closer to other clusters than either of the individual clusters was before.

3.2 Time Complexity of Hierarchical Clustering

Hierarchical clustering algorithms vary in their computational efficiency, depending on the similarity measure used.

Time Complexity of HAC: The basic **naive algorithm** for HAC has a time complexity of $\mathbf{O(N^8)}$, where N is the number of documents. This complexity arises from the need to compute all pairwise similarities and update the similarity matrix after each merge step.

Efficient HAC Algorithms: An optimized version of HAC, which uses a **priority queue** to store cluster similarities, reduces the complexity to

$$O(N^2 \log N)$$

In this approach, the most similar clusters are efficiently identified and merged using a data structure that supports fast retrieval of the most similar clusters.

Example: For a dataset of 1,000 documents, the time complexity would be as follows:

- **Naive HAC**: $O(1000) = 1,000,000,000$ operations.
- **Optimized HAC**: $O(1000 \log 1000) \approx 10,000,000$ operations.

Thus, optimized HAC is significantly faster and more feasible for larger datasets.

Complexity by Clustering Method:

- **Single-link**: $O(N)$, the most efficient HAC method due to its reliance on local proximity.
- **Complete-link, Group-average, Centroid**: $O(N \log N)$, these methods require more computation due to the need to assess global structure or average similarity between clusters.

Practical Considerations: Although

$$O(N^2 \log N)$$

complexity is manageable for moderate-sized datasets (e.g., 10,000 documents), HAC becomes impractical for very large datasets (e.g., millions of documents). In such cases, **flat clustering algorithms** like K-means are preferred.

3.3 Divisive Clustering

While HAC is a **bottom-up approach**, starting with singleton clusters and merging them, **divisive clustering** takes a **top-down approach**. In divisive clustering, all documents initially belong to a single large cluster, which is then recursively split into smaller clusters.

Divisive Clustering Process:

1. **Start with all documents** in one cluster.
2. **Split the cluster** using a flat clustering algorithm (e.g., K-means).
3. **Recursively split** each resulting cluster until each document is in its own cluster or until the desired number of clusters is reached.

Divisive clustering is conceptually more complex but can be more efficient than agglomerative clustering for large datasets.

Advantages of Divisive Clustering:

- **Efficiency**: For large datasets, divisive clustering can be faster because it does not require the exhaustive pairwise similarity computations of HAC. Instead, it can utilize **K-means**, which has a linear complexity relative to the number of documents.
- **Global Perspective**: By making global partitioning decisions early in the process, divisive clustering can avoid the local errors that occur in agglomerative clustering, where decisions are made based on pairwise similarities without considering the overall structure.

Example: Assume we have a dataset of 1,000 documents and wish to perform divisive clustering. Using **K-means** as the flat clustering algorithm, the overall complexity would be **O(NK)**, where K is the number of clusters we aim to create. For hierarchical divisive clustering, if we split the dataset into 10 clusters at each step, the complexity could be significantly lower than that of HAC.

3.4 Comparison of Agglomerative and Divisive Clustering

Aspect	**Agglomerative Clustering**	**Divisive Clustering**
Approach	Bottom-up (start with single docs)	Top-down (start with all docs
Efficiency	Less efficient, at least $O(N)$	More efficient, $O(NK)$
Accuracy	Often produces better local clusters	Can produce better global clust
Use of Flat Algorithms	Not required	Requires flat clustering algorith
Implementation Complexity	Easier to implement	More complex, requires hybrid me

Table 4: Comparison of Agglomerative and Divisive Clustering.

3.5 Cutting Dendrograms

A key feature of hierarchical clustering is that it produces a **dendrogram**—a tree that shows the merging history of clusters. In many applications, we do not need the full hierarchy but rather a flat partition of the data. To achieve this, we can **cut the dendrogram** at an appropriate level.

Methods for Cutting the Dendrogram:

1. **Cut at a specific similarity level**: For example, cut at a similarity of 0.4, meaning that any clusters formed with lower similarity are not merged.
2. **Cut based on the largest gap**: Find the largest difference between successive merge similarities and cut the dendrogram there.
3. **Prespecify the number of clusters**: Cut the dendrogram to produce a specific number of clusters, often determined by external requirements.

Example: Consider a dendrogram created for a set of 10 documents. If we wish to group the documents into three clusters, we would cut the dendrogram at the point where three distinct groups are formed.

Exercise: Cutting a Dendrogram Using the following similarity data and dendrogram for 5 documents, cut the dendrogram to form **2 clusters** and **3 clusters**:

- Merge history: {d1, d2} (0.8), {d3, d5} (0.7), {d4, (d3, d5)} (0.6), {(d1, d2), (d3, d4, d5)} (0.4).
- **For 2 clusters**: Cut at similarity 0.4.
- **For 3 clusters**: Cut at similarity 0.6.

Summary of Part 3

- **Optimality**: Single-link clustering is considered optimal for certain similarity-based objectives, while complete-link, group-average, and centroid clustering may suffer from outliers or inversions.
- **Time Complexity**: Single-link clustering is the most computationally efficient with
$$O(N^2 \log N)$$
complexity, but most HAC algorithms have
$$O(N^2 \log N)$$
complexity. Divisive clustering can be more efficient for large datasets.
- **Divisive Clustering**: A top-down approach that can generate more accurate clusters in certain cases, especially when combined with flat clustering algorithms like K-means.

In the final part, we will explore **cluster labeling**, **implementation issues**, and the practical applications of hierarchical clustering in real-world IR systems. We will also discuss how to present clusters to users and how to handle large-scale clustering problems.

Part 4: Cluster Labeling, Implementation, and Applications in Information Retrieval

In the final part of this discussion, we will cover **cluster labeling**—a crucial step in making clustering results interpretable for users. We will also discuss the practical **implementation issues** related to hierarchical clustering and look at some real-world **applications** in Information Retrieval (IR).

4.1 Cluster Labeling

Once clusters are formed, they need to be labeled so that users can understand the contents of each cluster. This is particularly important in **user-facing applications**, such as search engines or document analysis systems, where users interact with clusters and need to identify their relevance quickly.

Types of Cluster Labeling:

1. **Differential Cluster Labeling**: In this method, terms or features that are particularly distinct in a cluster (as compared to other clusters) are selected. Several techniques can be used for differential labeling:

   - **Mutual Information (MI)**: This measures how much knowing a term in a cluster reduces uncertainty about that cluster's content. Terms with high mutual information are good descriptors of the cluster.
   - **Chi-Squared Test**: This statistical test identifies terms that are overrepresented in one cluster compared to others.

   Both methods help to highlight the **distinctive characteristics** of a cluster, ensuring that the labels reflect the unique aspects of the clustered documents.

2. **Cluster-Internal Labeling**: This method focuses only on the contents of a cluster itself, without comparing it to others. Techniques include:

   - **Most Frequent Terms**: The most common terms within a cluster are selected as labels.
   - **Title of the Closest Document to the Cluster Centroid**: In document clustering, the title of the document that is closest to the centroid (the "average" document) can be used as the cluster label.

Example of Cluster Labeling: Let's assume a cluster of news articles is formed around the topic of **oil production**. The differential labeling method using mutual information might identify terms such as "crude oil," "barrel," and "OPEC" as distinguishing features of the cluster. Alternatively, the most frequent terms within the cluster might be selected, but this approach might introduce common terms like "year" or "company," which are not particularly informative about the cluster's topic.

Comparison:

- **Differential labeling** is more informative when clusters are meant to represent distinct categories.
- **Cluster-internal labeling** is useful when we are only interested in summarizing a single cluster, especially if it contains diverse documents.

4.2 Implementation Issues

Implementing hierarchical clustering for large-scale document sets poses several challenges. Despite the conceptual simplicity of HAC, there are **performance bottlenecks** and **scalability issues** that need to be addressed.

4.2.1 Computational Complexity

As discussed earlier, the time complexity of hierarchical clustering methods is typically $\mathbf{O(N^2 \log N)}$, which can be prohibitive for very large datasets. The computational cost arises from two main factors:

1. **Similarity Computation**: Computing the pairwise similarity between all documents is computationally expensive, especially for large document collections with high-dimensional feature vectors (e.g., term frequency-inverse document frequency or word embeddings).
2. **Cluster Merging**: The process of repeatedly updating the similarity matrix after each merge step is time-consuming.

4.2.2 Optimizing Performance

To overcome these challenges, several optimizations can be applied:

- **Inverted Index**: In information retrieval, inverted indexes (which store occurrences of terms in documents) can be used to speed up similarity computations by focusing only on relevant terms.
- **Sparse Representations**: For large document collections, it's often useful to work with **sparse vectors**, where only non-zero elements are stored. This reduces the complexity of similarity computations.
- **Truncating Centroids**: In methods like **centroid clustering**, centroids (which represent clusters) can become dense as clusters grow. By truncating centroids (i.e., keeping only the most relevant terms), we can reduce the computational burden.
- **Medoids instead of Centroids**: Using **medoids** (actual documents that best represent a cluster) instead of **centroids** (average points) can improve both efficiency and interpretability. Medoids are sparse and directly correspond to documents in the dataset.

4.2.3 Buckshot Algorithm for Large Datasets

For very large datasets (millions of documents), **HAC** becomes impractical. The **Buckshot algorithm** combines the **determinism** of HAC with the **efficiency** of flat clustering methods (like K-means). The process is as follows:

1. A random sample of size $\sqrt{N}$ is selected.

2. HAC is performed on this sample to generate high-quality seeds.

3. The remaining documents are clustered using K-means, initialized with the seeds obtained from HAC.

This hybrid approach provides the benefits of both methods—**hierarchical structure** and **scalability**.

4.3 Applications in Information Retrieval

Hierarchical clustering has a wide range of applications in information retrieval and related fields, including:

4.3.1 Document Organization

In digital libraries, news archives, and content repositories, hierarchical clustering helps organize documents into meaningful categories. Users can explore documents by navigating through the hierarchy of clusters, starting with broad topics and drilling down to more specific subtopics.

Example: Consider an online news archive where articles are clustered hierarchically based on topics. The top level might contain clusters like "Politics," "Economics," and "Sports." Each of these can be further divided into subtopics (e.g., "Elections," "Trade Policy" under "Politics").

4.3.2 Search Result Clustering

Search engines can use hierarchical clustering to group search results into clusters. Instead of presenting a long list of documents, users are shown clusters that represent different aspects of the query.

Example: For the query "machine learning," search results could be clustered into categories such as "Algorithms," "Applications in Healthcare," "Deep Learning," etc. Users can then choose the most relevant cluster to explore further.

4.3.3 First Story Detection and Event Tracking

Hierarchical clustering is used in **first story detection** and **event tracking**, where the goal is to identify the first occurrence of a news event and track its development over time.

Example: In news feeds, hierarchical clustering can group together articles reporting on the same event (e.g., a natural disaster). As new articles arrive, they are either added to existing clusters or used to form new clusters if they report a novel event.

4.3.4 Duplicate Detection

Single-link clustering can be used in **duplicate detection** tasks, such as identifying duplicate or near-duplicate documents on the web. By clustering similar documents together, search engines can avoid showing users multiple copies of the same document.

Example: In web crawling, single-link clustering helps identify pages that are duplicates or have near-duplicate content, ensuring that only one version is indexed.

4.4 Practical Considerations for Large-Scale Clustering

For real-world applications, especially on large-scale datasets, some considerations include:

1. **Scalability**: Flat clustering methods like K-means are more suitable for very large datasets. Hierarchical clustering is generally used when the dataset size is moderate, or when we need interpretable cluster hierarchies.
2. **Interpretability**: Hierarchical clustering is favored in applications where interpretability is important, such as in content analysis or taxonomy creation.
3. **Computational Resources**: The choice of clustering algorithm must balance computational resources (time, memory) and the quality of clusters.

Conclusion

In this series, we explored hierarchical clustering, focusing on **Hierarchical Agglomerative Clustering (HAC)**, its variants (single-link, complete-link, group-average, and centroid), and **divisive clustering**. We also discussed practical aspects of clustering, including **optimality**, **time complexity**, and **labeling**, which help make clustering results useful and interpretable in real-world information retrieval applications.

Hierarchical clustering remains a powerful tool in organizing and interpreting large datasets, especially when combined with efficient algorithms and proper labeling techniques. By addressing scalability and performance challenges, hierarchical clustering can play a vital role in document management, search engines, and content analysis systems.

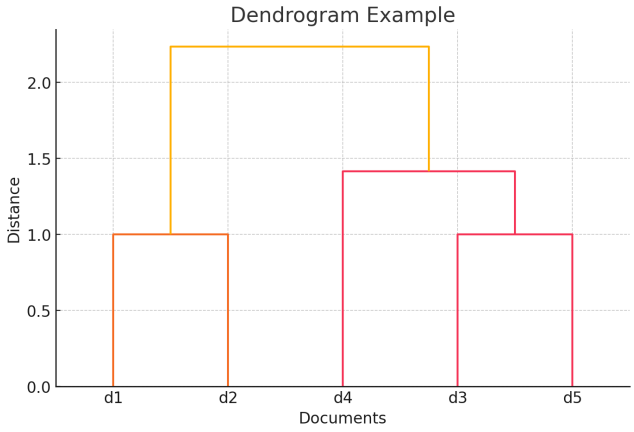

Figure 1: Dendrogram for Single-Link Hierarchical Clustering

Numerical Question: Single-Link Hierarchical Clustering

Given five documents represented as points in a 2-dimensional space:

Documents: $d1 = (1, 1)$, $d2 = (2, 1)$, $d3 = (4, 3)$, $d4 = (5, 4)$, $d5 = (3, 3)$

Apply **Single-Link Hierarchical Clustering** to these points. Construct the dendrogram based on the single-link clustering method and determine the order in which the documents will be merged.

Steps:

1. Calculate the Euclidean distance between all pairs of documents.
2. Use the **Single-Linkage** method to merge the two clusters with the smallest distance.
3. Continue merging until all documents are in a single cluster.

Explanation:

The following steps outline the process of applying single-link clustering:

- First, calculate the Euclidean distance between the document points $d1, d2, d3, d4, d5$.
- The first merge happens between $d1$ and $d2$, as they are the closest documents based on their Euclidean distance.

- Next, $d3$ and $d5$ are merged due to their proximity.
- The remaining documents are successively merged based on the smallest distances between clusters.

This hierarchical process creates a **dendrogram** where each horizontal line represents a merge. The y-axis denotes the distance (or dissimilarity) at which the merges occur.

Euclidean Distance Calculation

The Euclidean distance d between two points (x_1, y_1) and (x_2, y_2) is calculated as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- $d(d1, d2) = \sqrt{(2-1)^2 + (1-1)^2} = 1$
- $d(d1, d3) = \sqrt{(4-1)^2 + (3-1)^2} = \sqrt{9+4} = \sqrt{13} \approx 3.61$
- $d(d1, d4) = \sqrt{(5-1)^2 + (4-1)^2} = \sqrt{16+9} = \sqrt{25} = 5$
- $d(d1, d5) = \sqrt{(3-1)^2 + (3-1)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$
- $d(d2, d3) = \sqrt{(4-2)^2 + (3-1)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$
- $d(d2, d4) = \sqrt{(5-2)^2 + (4-1)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$
- $d(d2, d5) = \sqrt{(3-2)^2 + (3-1)^2} = \sqrt{1+4} = \sqrt{5} \approx 2.24$
- $d(d3, d4) = \sqrt{(5-4)^2 + (4-3)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$
- $d(d3, d5) = \sqrt{(3-4)^2 + (3-3)^2} = \sqrt{1+0} = 1$
- $d(d4, d5) = \sqrt{(5-3)^2 + (4-3)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$

Dendrogram Construction (Single-Link)

The following steps represent the order in which documents are merged:

1. Merge $d1$ and $d2$ since their distance is 1.
2. Merge $d3$ and $d5$ since their distance is 1.
3. Merge $d3, d5$ with $d4$, as $d3$ and $d4$ have a distance of 1.41.
4. Merge $d1, d2$ with the remaining cluster, as their closest point is at a distance of 2.24.

This process creates a complete dendrogram showing how the clusters are formed based on the smallest pairwise distances.