SEMANTIC STRUCTURING OF DIGITAL DOCUMENTS: KNOWLEDGE
GRAPH GENERATION AND EVALUATION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
Erik Luu
June 2024
COMMITTEE MEMBERSHIP

TITLE: Semantic Structuring of Digital Documents: Knowledge Graph Generation and Evaluation

AUTHOR: Erik Luu

DATE SUBMITTED: June 2024

COMMITTEE CHAIR: Rodrigo Canaan, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Foaad Khosmood, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science
ABSTRACT

Semantic Structuring of Digital Documents: Knowledge Graph Generation and Evaluation

Erik Luu

In the era of total digitization of documents, navigating vast and heterogeneous data landscapes presents significant challenges for effective information retrieval, both for humans and digital agents. Traditional methods of knowledge organization often struggle to keep pace with evolving user demands, resulting in suboptimal outcomes such as information overload and disorganized data. This thesis presents a case study on a pipeline that leverages principles from cognitive science, graph theory, and semantic computing to generate semantically organized knowledge graphs. By evaluating a combination of different models, methodologies, and algorithms, the pipeline aims to enhance the organization and retrieval of digital documents. The proposed approach focuses on representing documents as vector embeddings, clustering similar documents, and constructing a connected and scalable knowledge graph. This graph not only captures semantic relationships between documents but also ensures efficient traversal and exploration. The practical application of the system is demonstrated in the context of digital libraries and academic research, showcasing its potential to improve information management and discovery. The effectiveness of the pipeline is validated through extensive experiments using contemporary open-source tools.

Keywords: Knowledge Organization, Knowledge Graphs, Clustering, Semantic Embeddings, Document Embeddings
ACKNOWLEDGMENTS

I want to express my sincere gratitude to the following individuals and groups who have supported me throughout this journey:

- Amy Luu, Jenyung Luu, and Ella Luu for their unwavering support.
- Rodrigo Canaan, my advisor, for his guidance.
- Aidan Barbieux, my advisor-on-the-side, for his brainchild, continuous support, and inspiration.
- The rest of the LociMaps team: Kaanan Kharwa, Kenny Lau, Ethan Wolfe, Sasank Madineni, Yasser Binbisher, and Trevor Kirkby who are the foundation of this project.
- Carlo Ruggiero, my CS buddy, couldn’t have done it without you.
- My roommates Spencer Thurman, Lucas Lacambra, Mason Elefant, Joaquin Martinez de Pinillos, and Bryce Kueker for being amazing.
- Oscar Gomez, Rory Smail, and Nathan McCutchen, more CS boys.
- The Cal Poly cross-country team for everything.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
</tbody>
</table>

## CHAPTER

1 INTRODUCTION ........................................ 1

2 BACKGROUND .......................................... 3

2.1 Knowledge Graphs ................................. 3

2.2 Information Retrieval .............................. 5

2.2.1 Non-Semantic Search vs. Semantic Search ......... 6

2.2.1.1 Non-Semantic Search ......................... 6

2.2.1.2 Semantic Search ............................. 7

2.2.2 Embedding Architectures ....................... 8

2.2.2.1 Word Embeddings ............................. 8

2.2.2.2 Transformers ............................... 9

2.3 Similarity Scoring ............................... 10

2.3.1 Cosine Similarity ............................ 11

2.3.2 Soft Cosine Similarity ....................... 12

2.3.3 Euclidean Distance ........................... 13

2.3.4 Radial Basis Function (RBF) Kernel: ............ 14

2.4 Clustering ....................................... 15

2.4.1 Evaluation Metrics ............................ 17

2.5 Visualization ..................................... 18

2.6 Related Work ..................................... 19

vi
3.2.4.1 K-Nearest Neighbors (KNN) .......................... 39
3.2.4.2 Minimum Spanning Tree (MST) ......................... 40
3.2.4.3 Threshold-Based Edge Assignment ...................... 41
3.2.4.4 Mutual K-Nearest Neighbors (Mutual KNN) ........... 41
3.2.4.5 Spectral Clustering ................................... 42
3.2.5 Guaranteeing Connectedness: .............................. 42

4 EVALUATION METHODOLOGY ................................. 43
4.1 Embedding and Similarity Scores .............................. 43
4.2 Clustering Evaluation ................................... 44
4.3 Edge Assignment Evaluation ................................. 45
4.4 All Models and Algorithms Tested: .......................... 46

5 RESULTS ......................................................... 49
5.1 Medium Articles (n=2000) .................................. 49
5.1.1 Embedding and Similarity Scores Analysis ............... 49
5.1.2 Clustering Evaluation ................................ 53
5.1.2.1 Homogeneity ...................................... 53
5.1.2.2 Completeness ................................... 55
5.1.2.3 V-measure ....................................... 57
5.1.2.4 Tag Concentration Purity .......................... 58
5.1.3 Edge Assignment Evaluation ............................... 61
5.2 Medium Articles (n=200) .................................. 63
5.2.1 Embedding and Similarity Scores Analysis ............... 63
5.2.2 Clustering Evaluation ................................ 66
5.2.2.1 Homogeneity ...................................... 67
5.2.2.2 Completeness ................................... 68
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Summary of components and their variations used in the pipeline.</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Medium Articles: Minimum Tag Count Filtering</td>
<td>29</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td>65</td>
<td></td>
</tr>
</tbody>
</table>
5.11 Medium Articles (n=200): Top 10 configurations for homogeneity

5.12 Medium Articles (n=200): Top 10 configurations for completeness

5.13 Medium Articles (n=200): Top 10 configurations for v-measure

5.14 Medium Articles (n=200): Top 10 configurations for tag concentration purity

5.15 Medium Articles (n=200): Performance of the Top 10 Configurations Across All Depths

5.16 Medium Articles (n=200): Performance of the Top 10 Configurations Across All Depths

5.17 100 Medium articles with MiniLM embeddings, cosine similarity, K-means (k=10), and KNN (k=3). Each is labeled with its title and cluster assignment

5.18 Cluster nine from figure 5.17

5.19 Cluster seven from figure 5.17

5.20 Cluster two from figure 5.17
The rapid digitization of documents has created a vast and chaotic landscape making it increasingly difficult to organize and retrieve relevant information efficiently. Traditional methods of search and organization often fall short of addressing the complexities of these massive data landscapes, leading to issues such as information overload, irrelevant results, and disorganized data. In this context, innovative approaches are needed to transform the disarray into a structured and navigable repository of knowledge.

Knowledge Graphs (KGs) offer a promising solution over traditional semantic embedding spaces. While semantic embeddings provide rich representations of individual documents, KGs enhance this by capturing the intricate relationships and connections between different documents. KGs enable a more structured and interconnected view of data. However, they also come with their own set of challenges such as complexity in construction and the need for scalable algorithms.

Despite the potential benefits, there is a noticeable gap in accessible, open-source pipelines specifically designed for generating KGs where entire documents or bodies of text are treated as individual nodes. This gap highlights the need for tools that can seamlessly integrate document-level embeddings with graph-based representations providing a comprehensive solution for the semantic structuring of digital documents.

The motivation for this project stems from the challenges of LociMaps, a novel project aimed at improving knowledge organization and retrieval. It began as a collaborative effort to create a naturalistic information foraging system using principles from
cognitive science and advanced text embeddings to build interactive knowledge landscapes. It aimed to assess the impact of such visualizations on search efficiency, recall, stress levels, and cognitive overload compared to traditional search engines and file systems. Although initial results were inconclusive, the potential of combining human cognitive strengths with AI-driven knowledge representation was evident. It laid the groundwork for further exploration and refinement.

To evaluate the effectiveness of our chosen embedding models and algorithms, we conduct various analyses:

- **Embedding and Similarity Scores**: Comparing different methods for evaluating particular model embeddings and similarity between documents to identify the most effective approach.

- **Clustering**: Assessing the clustering algorithms used to group related documents and evaluating the purity and accuracy of these clusters.

- **Edge Assignment and Graph Construction**: Analyzing different edge assignment methods to construct the most meaningful and efficient knowledge graph.

These analyses help us understand the strengths and limitations of each approach, guiding further development and refinement.
The work of this thesis stems from the realm of information organization and retrieval. This is accomplished through the means of computer science, and more specifically, natural language processing. There are many subtopics and terms relevant to this work. Below is an explanation of some of these terms, methods, and related work in relation to knowledge graphs.

2.1 Knowledge Graphs

A knowledge graph is a data structure representing information in a network format where nodes (entities) are connected by edges (relationships). Each node may represent a real-world entity, such as a person, place, thing, or more abstract concept. Each edge represents a relationship between two entities, which can be labeled or signify a general relation.

Knowledge graphs have evolved from early semantic web projects, which utilized technologies like the Resource Description Framework (RDF) and Web Ontology Language (OWL) to create interconnected data structures. RDF provides a framework for describing resources and their relationships in the form of subject-predicate-object triples, known as semantic triples. OWL extends RDF by adding more vocabulary for describing properties and classes, enabling more complex and nuanced relationships [22].
Knowledge graphs are now integral in various applications, including search engines, graph neural networks, and recommendation systems. In this thesis, each node represents a document written in natural language, and edges indicate semantic similarity or membership in a cluster, weighted by the strength of their relationship.

In our case, each node represents a document written in natural language with arbitrary lengths. Each edge is assigned a weight depending on the strength of the relationship or belonging. Knowledge graphs are designed to integrate, manage, and reason about complex interrelationships within data. This makes them an essential tool for organizing semantic understanding and information retrieval.

An example of a knowledge graph can be seen in figure 2.1.

**Relevant Graph Terms (non-extensive):**

- **Node (Vertex):** An entity in the graph, representing an object such as a person, place, thing, topic, event, etc.

- **Edge (Link):** A connection between two nodes, representing a relationship or interaction between entities.

- **Directed Graph:** A graph where edges have a direction, indicating a one-way relationship between nodes.

- **Undirected Graph:** A graph where edges have no direction, indicating a two-way relationship between nodes.

- **Weighted Graph:** A graph where edges have weights, representing the strength or importance of the relationship.

- **Connected Graph:** A graph where there is a path between every pair of nodes.
Figure 2.1: Example KG: 100 Medium articles generated using sentence transformer 'all-MiniLM-L6-v2', cosine similarity, and K-nearest neighbors (k=2). PCA (n=5) was used to obtain the XY (first two) and RGB (last three) values.

- **Disconnected Graph:** A graph where at least one pair of nodes is not connected by a path.

### 2.2 Information Retrieval

Information retrieval (IR) is the process of obtaining relevant information from a large repository, typically a database or the web, based on a user's query. The field has evolved significantly from traditional keyword-based search methods to more advanced, context-aware systems powered by artificial intelligence (AI).
2.2.1 Non-Semantic Search vs. Semantic Search

The methods used in information retrieval can be broadly categorized into non-semantic and semantic search. Non-semantic search relies on keyword matching whereas semantic search aims to understand the contextual meaning behind the search queries and documents.

2.2.1.1 Non-Semantic Search

Traditional non-semantic search methods primarily rely on keyword-based approaches [34]. These methods involve matching search queries with documents based on the presence of specific keywords. The fundamental technique underlying non-semantic search is the use of inverted indexes mapping keywords to the documents containing them.

Non-semantic search methods have several limitations. One major drawback is the lack of context understanding. These methods treat words as isolated entities and ignore their semantic relationships and the context in which they appear. This often leads to irrelevant search results when different words with similar meanings or contexts are used. Additionally, non-semantic search struggles with synonym handling since it does not recognize that various words can have the same or similar meanings. This limitation reduces the effectiveness of search results, particularly in complex and nuanced queries.

One example is SQL databases [27]. They excel at handling structured data with rigid schemas; they fall short in managing and retrieving information from large, heterogeneous datasets with complex interrelationships.
Despite these limitations, non-semantic search has been foundational in the early
development of search technologies and is still used in many applications today due
to its simplicity and speed.

2.2.1.2 Semantic Search

While the idea is not new, Semantic search aims to improve upon traditional methods
by understanding the meaning behind search queries and the content of documents.
The foundations of semantic search can be traced back to the development of the
Vector Space Model (VSM) in the 1970s [46]. This algebraic model allowed for the
calculation of similarities between documents and queries using cosine similarity.

In VSM, documents are represented as vectors in a multi-dimensional space where
each dimension corresponds to a term (usually a word) in the document and the value
either corresponds to the frequency or weighted frequency of a term across the whole
document.

Fast forward to the 2010s, the introduction of word embeddings, such as Word2Vec
(2013) [40], marked a significant milestone. These models captured semantic relation-
ships between words by embedding them in continuous vector spaces allowing for a
more nuanced understanding of queries and documents. The late 2010s saw the ad-
vent of deep learning models, particularly the transformer architecture. Transfomers
like BERT (2018) [36] significantly advanced the state-of-the-art in NLP. These mod-
els could understand context and semantics at a much deeper level. This architecture
took us into the explosion of AI we’ve seen in the 2020s with technologies such as
ChatGPT [45].
2.2.2 Embedding Architectures

In this project, various embedding models are utilized to convert text data into numerical vectors that can be processed by machine learning algorithms. These models are trained on large datasets to capture the semantic meanings of words and sentences. The effectiveness of these models heavily depends on the quality and quantity of the training data.

Tokens and Tokenization

Tokenization is the process of breaking down text into smaller units called tokens. Tokens can be words, subwords, or characters, depending on the level of granularity required. For example, the sentence “The quick brown fox jumps over the lazy dog” can be tokenized into individual words: [“The”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”]. In natural language processing (NLP), tokenization is a crucial preprocessing step that prepares raw text data for embedding models. Proper tokenization ensures that the model can accurately capture the context and meaning of the text.

2.2.2.1 Word Embeddings

Word embeddings are trained to map words from a vocabulary into dense vectors in a continuous vector space, capturing semantic meanings based on the context in which words appear. See figure 2.2 for a visual example. This training typically involves large text corpora, such as Wikipedia or news articles, and relies on co-occurrence statistics. Models like Word2Vec generate static embeddings, meaning each word has a single representation regardless of its context. The training process involves tokenizing the text, defining a context window, and using methods like Continuous Bag of Words (CBOW) or Skip-Gram (for Word2Vec) to predict words or their contexts.
The resulting embeddings are useful for tasks like word similarity, clustering, and various downstream NLP applications.

2.2.2.2 Transformers

Before transformers, introduced by Vaswani et al. in 2017, Recurrent Neural Networks (RNNs) [48] and their variants were the primary models for sequence tasks in NLP. These models processed input sequentially which made them slow and difficult to parallelize. Attention mechanisms were introduced to address the limitations of RNNs by allowing the model to focus on relevant parts of the input sequence regardless of their distance from the current position. These transformers are limited by their context window.

Embeddings in transformers are created by converting input tokens into dense vector representations, enriched with positional information to capture the order of tokens. These embeddings are refined through multiple layers of self-attention and
feed-forward neural networks, resulting in context-aware embeddings that are crucial for a wide range of NLP tasks, such as text classification, machine translation, and question answering. The effectiveness of transformers is influenced by the size of their context window—the fixed maximum length of input sequences they can process at once. Larger context windows allow the model to capture more context, improving its ability to understand and generate language, though they also require more computational resources. Furthermore, a big portion of the effectiveness can be attributed to the scope and quality of the data used during training.

Similar to word embeddings, transformers are trained on large text corpora using objectives like Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The training process involves tokenizing the text, adding positional encodings, and passing the input through multiple layers of self-attention and feed-forward networks. This results in embeddings that are highly contextual and adaptable to a wide range of NLP tasks. The models are further fine-tuned on specific tasks to optimize their performance in more.

By testing these different embedding architectures and specific models, this thesis aims to identify the most effective approach for enhancing the clustering and knowledge graph generation processes within our knowledge graph generation pipeline. Each method is evaluated to determine its impact on the quality and efficiency of the resulting knowledge graphs.

### 2.3 Similarity Scoring

In this section, we discuss the similarity scoring metrics used in this project: cosine similarity, soft cosine similarity, and Euclidean distance. These metrics are funda-
Figure 2.3: Cosine Similarity; identical angles result in a similarity of one, opposite angles result in zero.

mental in measuring the similarity or distance between vectors, which in this context, represent textual data in the form of embeddings.

2.3.1 Cosine Similarity

Cosine similarity measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. It is computed as the dot product of the vectors divided by the product of their magnitudes, producing a value between 0 and 1, where a higher value indicates greater similarity. It is widely used in text analysis because it effectively measures the orientation, rather than magnitude, of vectors, making it particularly useful for comparing documents of varying lengths. See figure 2.3 for a visual.

Formula:

\[
\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}
\]

Where:
- $A$ and $B$ are the vector representations of the two documents being compared.
- $A \cdot B$ is the dot product of the vectors $A$ and $B$, calculated as $\sum_i A_i B_i$.
- $\|A\|$ and $\|B\|$ are the magnitudes (or Euclidean norms) of the vectors $A$ and $B$.
  The magnitude of a vector $A$ is calculated as $\sqrt{\sum_i A_i^2}$.

### 2.3.2 Soft Cosine Similarity

Soft cosine similarity extends the traditional cosine similarity by considering the similarity between individual features (terms) and produces a value between 0 and 1, like before, a higher value indicates greater similarity. It accounts for the semantic similarity between terms, rather than treating them as independent. By considering term similarities, it provides a more nuanced similarity measure that captures semantic relationships, making it effective in scenarios where vocabulary overlap is limited.

**Formula:**

$$\text{Soft Cosine Similarity} = \frac{\sum_{i,j} S_{ij} A_i B_j}{\sqrt{\sum_{i,j} S_{ij} A_i A_j} \sqrt{\sum_{i,j} S_{ij} B_i B_j}}$$

Where:

- $A$ and $B$ are the vector representations of the two documents being compared. Each element $A_i$ and $B_j$ represents the value of the $i$-th and $j$-th term in the vectors $A$ and $B$, respectively.
- $i$ and $j$ are the indices of terms in the document vectors.
- $S$ is the similarity matrix where $S_{ij}$ represents the similarity between the $i$-th term and the $j$-th term. This matrix helps to capture semantic similarities between different terms.
The numerator, $\sum_{i,j} S_{ij} A_i B_j$, calculates the weighted sum of the products of term values from the two documents, with weights being the similarities between terms. The denominator, $\sqrt{\sum_{i,j} S_{ij} A_i A_j} \sqrt{\sum_{i,j} S_{ij} B_i B_j}$, normalizes the similarity score by the magnitudes of the vectors, adjusted for term similarities.

While the similarity matrix is typically generated by considering each word and its corresponding Word2Vec vector, we adapted it to work with the transformer embedding dimensions. This approach achieves the desired effect even though the embedding dimensions are not directly interpretable in terms of specific real-world meanings.

2.3.3 Euclidean Distance

Euclidean distance is a measure of the true straight-line distance between two points in a multi-dimensional space. It is computed as the square root of the sum of the squared differences between corresponding elements of the vectors. While Euclidean distance is not usually used in comparing semantic vectors, it is often used in clustering and classification problems due to its straightforward interpretation and simplicity. For instance, in clustering algorithms like K-Means, Euclidean distance helps to define clusters by minimizing the distance within clusters and maximizing the distance between clusters. In classification tasks, it can be used in K-Nearest Neighbors (KNN) algorithms to classify a data point based on the majority class among its closest neighbors. Despite its limited application in direct semantic comparison, Euclidean distance remains a valuable metric in the broader scope of machine learning and data analysis.

Formula:

$$\text{Euclidean Distance} = \sqrt{\sum_i (A_i - B_i)^2}$$
Where:

- $A$ and $B$ are the vector representations of the two documents being compared. Each element $A_i$ and $B_i$ represents the value of the $i$-th term in the vectors $A$ and $B$, respectively.

However, in high-dimensional data, distance metrics like Euclidean distance can become less meaningful due to the “curse of dimensionality,” where the distances between data points tend to become more uniform as the number of dimensions increases. This makes it difficult to distinguish between close and distant points effectively. Similarity metrics, on the other hand, provide a normalized measure that captures the relative closeness of data points, which is particularly useful in high-dimensional spaces. That’s why we use the Radial Basis Function (RBF) Kernel to convert Euclidean distances into similarities.

### 2.3.4 Radial Basis Function (RBF) Kernel:

The RBF kernel, also known as the Gaussian kernel, transforms the distance into a similarity score between 0 and 1, just like cosine and soft cosine similarity above. This is particularly useful for comparing semantic vectors in a way that captures the notion of “closeness” in a high-dimensional space.

**Formula:**

$$
RBF \text{ Similarity} = \exp \left( -\frac{\text{Euclidean Distance}^2}{2\sigma^2} \right)
$$

Where:
• Euclidean Distance is the straight-line distance between the vectors, as previously defined.

• $\sigma$ is a parameter that controls the width of the Gaussian kernel. A smaller $\sigma$ results in a sharper drop-off in similarity with distance, meaning only very close distances will result in high similarity scores.

• $\exp\left(-\frac{\text{Euclidean Distance}^2}{2\sigma^2}\right)$ is the exponential function that transforms the Euclidean distance into a similarity score. The similarity score ranges between 0 and 1, with 1 indicating identical vectors and values closer to 0 indicating dissimilar vectors.

\section{2.4 Clustering}

Clustering is a crucial technique in information retrieval for organizing large datasets by grouping similar items together. This process helps in enhancing the structure and efficiency of knowledge graphs by ensuring that related documents are grouped, which makes information retrieval more intuitive and effective.

Clustering techniques can be broadly categorized based on the approach they use to form clusters. Here are some general types of clustering algorithms we considered in this project:

• **Centroid-based Clustering:** Centroid-based clustering assigns each data point to the nearest cluster center, iteratively refining these centers to minimize within-cluster variance. The most common algorithm in this category is K-Means. Key characteristics include reliance on a predefined number of clusters, minimizing the sum of squared distances between points and their as-
signed cluster centroid, and efficiency for large datasets but assuming clusters are spherical and of similar size.

• **Density-based Clustering:** Density-based clustering algorithms, like DBSCAN, identify clusters as areas of high point density, separated by areas of low density. Points in low-density areas are often considered noise or outliers. Key characteristics include not requiring the number of clusters to be specified in advance, the ability to find clusters of arbitrary shapes and sizes, and the effectiveness at identifying noise and handling clusters of varying densities.

• **Probabilistic Clustering:** Probabilistic clustering methods, such as Gaussian Mixture Models (GMM), assume that the data is generated from a mixture of several Gaussian distributions. Each point is assigned a probability of belonging to each cluster. Key characteristics include providing a soft assignment of points to clusters based on probability, the capability of modeling clusters with different shapes and covariance structures, and the need to specify the number of components.

• **Hierarchical Clustering:** Hierarchical clustering builds a tree-like structure (dendrogram) of clusters either by iteratively merging smaller clusters into larger ones (agglomerative) or by splitting larger clusters into smaller ones (divisive). Key characteristics include not requiring the specification of the number of clusters upfront, providing a multi-level hierarchy of clusters, and potential computational intensity for large datasets.

• **Model-based Clustering:** Model-based clustering, like Birch (Balanced Iterative Reducing and Clustering using Hierarchies), assumes that the data is generated by a mixture of underlying probability distributions. It uses an incremental approach to build a hierarchical clustering model. Key characteristics include suitability for large datasets with incremental data processing, handling
varying cluster sizes and shapes, and efficiency and scalability though it may not perform well with clusters of varying densities.

In this project, we specifically utilized several clustering techniques from the categories mentioned above:

- **K-Means**: A centroid-based clustering algorithm that assigns each data point to the nearest cluster center and iteratively refines these centers [6].

- **DBSCAN**: A density-based clustering algorithm that identifies clusters as areas of high point density and marks points in low-density areas as outliers [6].

- **Gaussian Mixture Models (GMM)**: A probabilistic clustering method that models the data as a mixture of several Gaussian distributions, assigning each point a probability of belonging to each cluster [8].

- **Birch**: A model-based clustering algorithm that builds a hierarchical clustering model incrementally, suitable for large datasets with varying cluster sizes and shapes [31].

### 2.4.1 Evaluation Metrics

Evaluating the quality of clustering is critical to understanding the effectiveness of different clustering algorithms and their impact on information retrieval. These are some general metrics:

- **Silhouette Score**: The silhouette score measures how similar an object is to its own cluster compared to other clusters. It provides a way to assess the cohesion within clusters and the separation between clusters [37]. A high silhouette score indicates well-defined clusters.
• **Davies-Bouldin Index**: The Davies-Bouldin index evaluates the average similarity ratio of each cluster with the cluster that is most similar to it. Lower values indicate better clustering, as they signify lower within-cluster scatter and greater separation between clusters [35].

• **Homogeneity**: Homogeneity measures if all of the clusters contain only data points that are members of a single class. Higher homogeneity indicates better clustering performance.

• **Completeness**: Completeness measures if all the data points that are members of a given class are assigned to the same cluster. Higher completeness indicates better clustering performance.

• **V-Measure**: The V-Measure is the harmonic mean of homogeneity and completeness, providing a balanced evaluation of clustering performance. Higher V-Measure values indicate better clustering quality.

\[
V\text{-measure} = 2 \times \frac{\text{homogeneity} \times \text{completeness}}{\text{homogeneity} + \text{completeness}}
\]

By understanding these general clustering techniques, we can better choose and apply the appropriate metrics for organizing and analyzing large datasets in information retrieval and other applications like ours.

### 2.5 Visualization

Because these documents are represented by vectors with hundreds, if not thousands, of dimensions, there are techniques to reduce the dimensions to combine variables or simply make the data more digestible. For example, Principal Component Analysis (PCA) is a popular dimensionality reduction technique used to visualize high-
dimensional data. PCA works by transforming the original high-dimensional data into a new coordinate system where the greatest variance in the data is captured in the first few dimensions, called principal components.

By projecting the high-dimensional vectors of our documents onto these principal components, we can effectively reduce the number of dimensions while preserving as much of the data’s variance as possible. This makes it easier to visualize and interpret the relationships between documents in a lower-dimensional space, typically 2D or 3D.

In our case, PCA was employed to reduce the dimensionality of the document vectors, as we did in 2.1, enabling us to plot and explore the structure and relationships within the knowledge graph. This visualization helps to identify clusters of related documents, outliers, and overall patterns in the data, providing valuable insights during and after development into the underlying structure of the document corpus.

2.6 Related Work

The field of knowledge graph (KG) generation has come a long way over the years. This section looks at some key developments from the early days of the Semantic Web to modern tools like Nomic Atlas and recent research on generating knowledge graphs from text.

2.6.1 Semantic Web

The Semantic Web was an idea put forward by Tim Berners-Lee in 1994 [28]. The goal was to make web data readable by machines using standards like the Resource Description Framework (RDF) and the Web Ontology Language (OWL). These tech-
Technologies help create interconnected data structures—knowledge graphs—where each node represents an entity, and edges represent the relationships between them. The Semantic Web laid the groundwork for many future advancements in data connectivity and semantic data representation.

2.6.2 WordNet

WordNet is a large lexical database of English words [29] grouped into sets of synonyms called synsets. It was developed at Princeton University in the mid-1980s by hand. Unlike a regular thesaurus, WordNet provides richer semantic relationships such as hypernyms, hyponyms, meronyms, and antonyms. It has been an essential resource for natural language processing (NLP) tasks and has influenced many semantic web and knowledge graph projects.

2.6.3 Nomic Atlas

Nomic Atlas is a cutting-edge platform by Nomic AI [18] for handling and visualizing unstructured data like text, images, videos, audio, and embeddings. While it isn’t a knowledge graph, it shares similarities with the project discussed in this thesis. Here’s a rundown of what it can do:

- **Unstructured Data Map**: Nomic Atlas organizes unstructured data into an interactive map that groups similar data points based on their semantic content. This makes it easy to explore and understand large datasets, and the map updates dynamically as new data is added.

- **Embeddings**: The platform uses embeddings—vector representations of data points—to organize the data. These embeddings help in efficiently searching
and filtering the data. If users don’t provide their embeddings, Nomic’s models can generate them. We use Nomic’s embedding model in this thesis.

- **Visualization and Search:** Users can search large datasets, apply metadata filters, and use visualization tools to find insights. You can filter data by timestamps or sentiment scores and run complex search queries.

- **Topic Modeling:** Nomic Atlas automatically creates topic labels from the data, helping users navigate and understand the dataset. These topics can be hierarchical.

- **Generative AI and Model Integration:** Nomic Atlas supports integrating generative AI models and other machine learning models which is great for understanding and visualizing the latent spaces of neural networks during training and evaluation.

Despite these capabilities, Nomic Atlas doesn’t specifically create knowledge graphs. However, its ability to visualize and manage large datasets with embeddings closely relates to the goals of this thesis: focusing on generating knowledge graphs where each document is a node and relationships are based on semantic similarity.

### 2.6.4 Knowledge Graph Generation from Text

A recent paper focuses on creating an end-to-end multi-stage system for generating knowledge graphs from text. The system aims to efficiently extract and represent information in structured graph formats where nodes represent entities extracted from text using pre-trained language models and edges (with labels) represent relationships between entities [39].
• **Methodology:** This paper uses pre-trained language models fine-tuned for entity extraction and seq2seq models like T5 for generating nodes. The process has distinct stages for generating nodes and edges.

• **Differences:** Unlike this paper, my paper evaluates different models and algorithms for generating KGs treating each document as a standalone node. The relationships between nodes are based on similarity and specific criteria making my work more of a comparative study of various models and techniques for knowledge graph generation.

In summary, the evolution from the Semantic Web to modern platforms like Nomic Atlas and recent research papers shows how sophisticated and useful knowledge graphs have become. This thesis builds on these advancements focusing on testing modern open-source technologies to generate document-based knowledge graphs.

### 2.7 Technical Background

This section details the software and tools utilized in this thesis for data processing, mathematical computations, model training, and visualization. The combination of these tools enables the efficient development and evaluation of the knowledge graph generation and clustering pipeline.

We primarily used Python, a versatile programming language, for its rich ecosystem of libraries and frameworks. Javascript and HTML are used for visualizing these graphs to assist in development. For this, we used the d3.js package [5].
2.7.1 Python Libraries

- **Hugging Face Transformers**: The Hugging Face Transformers library [15] is utilized for natural language processing tasks. It provides pre-trained models and tokenizers that facilitate the creation of embeddings for text data. Fine-tuned models from Google, Nomic, and HuggingFace are used to generate high-quality embeddings that capture the semantic meaning of documents.

- **Scikit-learn**: Scikit-learn [23] is a machine learning library that provides simple and efficient tools for data mining and data analysis. It is used in this thesis for clustering algorithms such as K-Means, DBSCAN, GMM, and Birch. Scikit-learn’s comprehensive suite of clustering techniques allows for thorough evaluation and comparison of different clustering methods.

- **PyTorch**: PyTorch [21] is built for creating and training deep learning models. However, in this thesis, it is primarily used for its implementation of metrics like cosine similarity, soft cosine similarity, and Euclidean distance.

- **NetworkX**: NetworkX [16] is a library for the creation, manipulation, and study of complex networks. In this project, it is used for constructing and analyzing knowledge graphs.

- **Pandas**: Pandas [20] is used for data manipulation and analysis. It provides data structures such as DataFrames that facilitate the handling of structured data. It is particularly useful for cleaning and preprocessing datasets and enabling efficient data exploration and transformation.

- **NumPy**: NumPy [19] is employed for numerical computations, offering support for large multi-dimensional arrays and matrices. NumPy is utilized for a wide
range of mathematical operations and data manipulation tasks, providing the foundational support needed for the efficient processing of numerical data.

- **Gensim**: Gensim [9] is a library used for topic modeling and document similarity analysis. We use it specifically for Word2Vec.

- **Taichi**: Taichi [26] is a high-performance computing library that supports parallel programming and is particularly useful for computationally intensive tasks.

The combination of these software tools and libraries facilitates the development of a robust and efficient pipeline for the construction and evaluation of the semantic structure of the produced graphs.
Our study employs an experimental and exploratory research design to evaluate the proposed methodology for the knowledge graph generation. The objective is to systematically assess the performance of various modular components to build an efficient and optimal pipeline for organizing information-rich environments as knowledge graphs.

The main inputs to our system are diverse textual datasets, text embedding models, clustering algorithms, and relationship assignment criteria. The system’s components are designed in a modular fashion, allowing for flexibility in handling various datasets, embedding methods, and distance metrics. This modularity ensures that the pipeline can be adapted and optimized for different scenarios.

**Inputs and Main Components**

- **Textual Datasets:** We use a variety of datasets, including Medium articles and arXiv papers, to test the system.

- **Embedding Models:** The system utilizes different embedding models: Word2Vec and transformers from Hugging Face.

- **Clustering Algorithms:** Various clustering methods are implemented to group similar documents: K-Means, DBSCAN, GMM, and Birch.

- **Relationship Assignment:** Different criteria for assigning relationships between documents are tested to optimize graph coherence.
Modular Design and Flexibility  The modular design allows for easy replacement and testing of different components. This flexibility is crucial for optimizing the pipeline’s performance across different datasets and tasks. The following table summarizes the variations of components used in the system:

Table 3.1: Summary of components and their variations used in the pipeline.

<table>
<thead>
<tr>
<th>Component</th>
<th>Variations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual Datasets</td>
<td>Medium articles, ArXiv papers</td>
</tr>
<tr>
<td>Embedding Models</td>
<td>Word2Vec, HuggingFace transformers</td>
</tr>
<tr>
<td>Clustering Algorithms</td>
<td>K-Means, DBSCAN, GMM, Birch</td>
</tr>
<tr>
<td>Distance Metrics</td>
<td>Cosine similarity, soft cosine similarity, Euclidean distance</td>
</tr>
<tr>
<td>Relationship Assignment</td>
<td>Various criteria based on embedding distances and clustering results</td>
</tr>
</tbody>
</table>

Evaluation and Analysis  We conduct experiments to evaluate the robustness and generalizability of the different approaches across datasets. The evaluation metrics include embedding distance, graph coherence, and clustering ability at different stages of the pipeline. Additionally, we perform qualitative analysis, visualization, and hypothesis generation to uncover insights and patterns within the generated knowledge graphs.

This structured approach ensures a comprehensive understanding of the pipeline’s performance and its potential for organizing and retrieving information from large, complex datasets.

3.1 Data

Our experiments and pipeline interactions utilize a dataset comprising 192,340 Medium articles [14] and, later, the 1.7 million scientific papers sourced from arXiv [30].
3.1.1 Medium Articles

Each entry in the Medium dataset represents a distinct article encompassing features such as the article title, textual content, associated URL, authors, publication date-time, and a set of tags 3.1.

The data collection process involved scraping articles using Python and the requests library. Beginning with an arbitrary article on Medium, subsequent articles were selected by crawling through the author archive pages, publication archive pages (if available), and tag archives (if available). The HTML pages of the articles were parsed using the newspaper Python library. To ensure consistency, only English articles were retained, accomplished through filtering with the langdetect library.

However, due to the nature of the collection methodology, the distribution of publication dates within the scraped dataset is not uniform. While articles range from 2016 to 2022, there is a notable discrepancy in the number of articles published year-by-year. Notably, there is a significant prevalence of articles published in 2020 as seen in figure 3.2.
3.1.2 ArXiv Papers

As for the ArXiv dataset, it is a mirror of the full database—updated monthly. Because the full set is rather large (1.1TB and growing), this dataset provides only a metadata file in the “JSON” format. Each entry represents a distinct paper each containing an ID (can be used to access the paper), submitter, authors, title, comments (additional info, such as number of pages and figures), journal-ref (information about the journal the paper was published in), DOI (Digital Object Identifier), abstract, categories (tags in the ArXiv system), and versions.

3.1.3 Preprocessing

To facilitate analysis, we performed preprocessing on the data, primarily involving the standardization of tags and an overall reduction of the dataset. For instance, variations in the Medium tags such as “COVID-19” and “coronavirus” were unified into a singular tag. Figure 3.3 shows some of the 1000 top most common tags.

ArXiv’s category taxonomy is very closely related to Mediums tags. Each paper has at least one English description such as ’cs.CC’ and is translated into its corresponding
overarching category ‘Computer Science’ and its subcategory ‘Computational Complexity’ for readability.

The initial dataset we utilized was the medium dataset due to its inclusion of complete articles, a feature lacking in ArXiv’s abstract-only data. Obtaining the full text for each ArXiv paper would necessitate additional scraping efforts. To streamline our initial processing and testing phase, we aimed to narrow our focus to articles with tags appearing a certain number of times ($N=[5000, 1000, 500, 100]$). The process resulted in table 3.2. The process finds all the tags occurring greater than $N$ times, removes all articles with no tags that appeared with the minimum count and keeps only the tags with the minimum count. Based on an intuitive sense, we picked to keep the dataset with a 1000 minimum tag count as 98 unique tags seemed like a reasonable number of tags to manage.
The last step in this preprocessing stage was to simplify the tags. As mentioned before, variations in the Medium tags occurred (e.g. “COVID-19”, “coronavirus”). Here is a sample of the remaining tags: [“Mental Health”, “Health”, “Psychology”, “Science”, “Coronavirus”, “Society”, “Books”, “Entrepreneurship”, “Writing”, “Marketing”, “Productivity”, ...]. By hand, we came up with a list of 20 tags we can use to summarize these 98. Here is a sample of the mapping used: [“Fiction”: “Books”, “Creativity”: “Art”, “Design”: “Art”, “Data Visualization”: “Technology”, “Business”: “Business”, “Environment”: “Lifestyle”, “Art”: “Art”, “Humor”: “Entertainment”, “Social Media”: “Technology”, “AI”: “Technology”, ...]. These new simplified assigned tags—which are used during the analysis—are saved in a separate column in the dataset.

This all ensures a more manageable dataset for pipeline development. This approach helps to reduce noise by excluding those with infrequent tags, which may represent outliers or tangential topics. However, it’s important to note that this initial reduction is temporary, intending to incorporate all articles into the analysis once the pipeline is established. This phased approach balances efficiency with the eventual goal of comprehensive analysis across the entire dataset, ensuring manageable testing conditions and robust results. Ultimately, the final pipeline is designed to be agnostic to the data content after preprocessing, allowing it to handle diverse and heterogeneous datasets effectively.

3.2 Pipeline

In this section, we explore the inner workings of our pipeline: a framework engineering to navigate the complexities of the provided data to construct a knowledge graph. From embedding our data to similarity metrics, clustering, edge assignment,
and graph alteration methods, each component plays a vital role in uncovering underlying patterns and structures (see Figure 3.4). Given the experimental nature of this project, each module offers the flexibility to incorporate various methods or algorithms interchangeably as deemed suitable. We will analyze the resulting knowledge graphs to determine the optimal configuration for our modules in three stages.

3.2.1 Embeddings

We begin with the foundational element: embeddings. They enable us to transform raw text data into numerical representations. They capture semantic and syntactic similarities between words, phrases, or documents in a quantifiable manner. By encoding textual information into dense vector spaces, embeddings facilitate nuanced comparisons and analyses, leading to hidden patterns and relationships within the written context. All embedding models chosen produce vectors of floating point numbers of size 1xN (N is inherent to individual model specifications).

We concentrated on two primary embedding techniques: word embeddings like Word2Vec and sentence embedding transformers hosted on HuggingFace. For more details, refer to section 2.
3.2.1.1 Constraints and Batching

Given the constraint of having only 8GB of RAM, it is necessary to process embeddings in batches to avoid memory overflow. By dividing the data into manageable chunks, we ensure that each batch fits within the available memory, allowing us to process large datasets without interruption.

3.2.1.2 Normalizing Chunks Before or After Aggregation

Since the majority of documents are longer than the embedding models’ context windows, aggregation is necessary to obtain one vector per document. Furthermore, normalization is an essential step in the embedding process to ensure that all vectors have a uniform scale, which can significantly impact the performance of downstream tasks such as clustering and similarity calculations. There are two main approaches to normalization: before or after aggregation.

**Normalizing Before Aggregation:** Normalizing chunks before aggregation means that each batch of embeddings is normalized independently before being combined. This approach ensures that the embeddings in each batch are on the same scale, which can help maintain consistency when batches are aggregated. However, this method may introduce slight variations between batches due to the independent normalization process, potentially affecting the overall coherence of the aggregated embeddings.

**Normalizing After Aggregation:** On the other hand, normalizing after aggregation involves combining all batches of embeddings first and then applying normalization to the entire set. This method ensures a uniform scale across all embeddings, leading to greater consistency in the final normalized vectors. It can be particularly beneficial
when dealing with large variations in embedding values across different batches, as the global normalization step adjusts for these discrepancies.

In this project, we opted to normalize the embeddings after aggregation to achieve a more consistent and coherent representation across the entire dataset. This decision was motivated by the need to ensure that the combined embeddings retain their relative scales, enhancing the accuracy of similarity and clustering algorithms applied subsequently.

3.2.1.3 Word Embeddings

Word embeddings represent a foundational approach in natural language processing, providing numerical representations for each word and aggregating them into a single vector. These techniques have been widely studied and utilized in various NLP tasks due to their simplicity and effectiveness.

3.2.1.4 Contextual Word Embeddings

To capture the contextual meaning of words, we leveraged contextual word embeddings such as BERT-based models or other transformer-based architectures like all-MiniLM-L6-v2 and nomic-embed-text-v1.5 found on HuggingFace. These embeddings generate context-sensitive representations for each word based on its surrounding context within a sentence.

For each set of document embeddings, we use one of the three similarity metrics (cosine, soft cosine, euclidean) to obtain a similarity matrix. Refer to section 2.3. Each row of the matrix corresponds to a document and each column corresponds to the document it’s being compared to.
The embedding models utilized in this project are as follows:

- **HuggingFace**: These two models were fine-tuned using the same 1B sentence pairs (including Reddit comments, Wiki Answers, S2ORC Citations pairs, and more) but have different base models.

  - **all-MiniLM-L6-v2**: This model uses the pretrained nreimers/MiniLM-L6-H384-uncased model. [24].
  
  - **all-mpnet-base-v2**: This model uses the pretrained microsoft/mpnet-base model [25].

- **Nomic**:

  - **nomic-embed-text-v1.5**: Trained on a large, heterogeneous dataset of English text, this model is designed to generate embeddings that effectively capture the nuances in textual data [17].

- **Allen Institute**:

  - **allenai/specter**: Specifically trained on scientific papers, this model aims to create embeddings that are particularly useful for academic and research-focused documents [2].

- **Google**:

  - **google-bert/bert-base-uncased**: This is a widely used model trained on the BooksCorpus and English Wikipedia. It provides powerful embeddings by capturing a wide range of linguistic features from large and diverse text corpora [10].

  - **Word2Vec (hosted by Gensim)**: Pre-trained on the Google News dataset, this model captures the semantic relationships between words through continuous bag-of-words (CBOW) and skip-gram approaches [24].
3.2.2 Similarity Scoring

We used various similarity scoring metrics to quantify the similarity or dissimilarity between the floating point vectors representing entire documents.

**Cosine Similarity:** It calculates the cosine of the angle between two vectors, producing a value between 0 and 1, where a higher value indicates greater similarity. Cosine similarity is preferred when considering the semantic similarity between vectors, as it is invariant to the magnitude of the vectors and focuses solely on the direction. This property makes cosine similarity particularly suitable for comparing vectors generated by transformer models, such as BERT-based models, where the emphasis is on capturing semantic relationships between words or sentences.

**Soft Cosine Similarity:** Soft cosine similarity extends the traditional cosine similarity by considering the similarity between features, not just their presence. This metric is useful in scenarios where features (e.g., words or vector dimensions in our case) are not completely independent. By incorporating a feature similarity matrix, soft cosine similarity provides a more nuanced measure of similarity, capturing semantic relationships more effectively than standard cosine similarity.

**Euclidean Distance/Similarity:** It provides a measure of dissimilarity between vectors, with smaller distances indicating greater similarity. Euclidean distance is commonly used in clustering and classification tasks, where the goal is to partition data points into groups based on their proximity in feature space. While Euclidean distance considers both the direction and magnitude of vectors, it may not always be the most appropriate choice for comparing vectors in high-dimensional spaces, as it can be sensitive to differences in magnitude. However, because the rest of the pipeline requires a similarity value, we normalize and invert the distances to between 0 and 1.
where a higher value indicates greater similarity. We use the Gaussian (RBF) kernel, which maps distances to the range \([0, 1]\).

For more information about these formulas, refer to section 2.3.

3.2.3 Clustering Methods

Clustering is a fundamental technique in machine learning and data analysis. In this thesis, we explore several clustering algorithms that assign each element a specific cluster-ID. These IDs provide insights into the structure and relationships within the data and are used in later processes; they are as follows:

3.2.3.1 K-Means

K-Means Clustering is a widely-used centroid-based algorithm that partitions a dataset into \(K\) clusters, where \(K\) is a predefined number. The algorithm iteratively assigns each data point to the nearest centroid and then updates the centroids based on the mean of the assigned points. This process repeats until the centroids stabilize.

K-Means was chosen for its simplicity and efficiency, which are crucial when dealing with large datasets of document embeddings. However, while its ability to quickly partition data into a specified number of clusters it is not perfectly suitable for scenarios where the expected number of categories is not known in advance.

Algorithm: K-Means
Input: data points \(X\), number of clusters \(K\)
Output: cluster assignments

1. Initialize \(K\) centroids randomly
2. Repeat until convergence:
   a. Assign each point to the nearest centroid
   b. Update centroids as the mean of assigned points
3. Return cluster assignments

3.2.3.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based clustering algorithm that groups together points that are closely packed and marks points that lie alone in low-density regions as outliers. It requires two parameters: epsilon, which defines the radius of the neighborhood around a point, and the minimum number of samples to form a dense region.

DBSCAN was selected for its ability to find arbitrarily shaped clusters and handle noise. This is particularly useful in document clustering where documents can vary significantly in length and content—like our data—leading to clusters of different densities. DBSCAN’s robustness to outliers ensures that noisy documents do not distort the cluster structure, making it ideal for identifying core topics within a heterogeneous document set.

Algorithm: DBSCAN
Input: data points X, epsilon ε, min samples MinPts
Output: cluster assignments

1. For each point:
   a. If the point is not visited:
      i. Mark as visited
      ii. Find neighboring points within ε
      iii. If neighbors >= MinPts, form a cluster
          - Expand cluster with density-reachable points

2. Return cluster assignments

3.2.3.3 Gaussian Mixture Models (GMM)

Gaussian Mixture Models assume that the data is generated from a mixture of several Gaussian distributions, each representing a cluster. The algorithm uses the
Expectation-Maximization (EM) method to iteratively estimate the parameters of the Gaussian distributions and the probability that each data point belongs to each distribution.

GMM was chosen for its flexibility in modeling clusters of different shapes and sizes and its probabilistic nature. This is advantageous in scenarios where documents may belong to multiple topics allowing for a more nuanced understanding of document similarities and the relationships between clusters.

**Algorithm: Gaussian Mixture Models (GMM)**

**Input:** data points $X$, number of components $K$

**Output:** cluster assignments

1. Initialize parameters for $K$ Gaussian distributions
2. Repeat until convergence:
   a. E-step: Calculate probability of each point belonging to each Gaussian
   b. M-step: Update parameters to maximize likelihood
3. Assign points to the Gaussian with the highest probability
4. Return cluster assignments

### 3.2.3.4 Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

BIRCH is designed for large datasets and incrementally builds a clustering feature (CF) tree. It reduces the data into compact summaries and then clusters these summaries.

BIRCH was selected for its efficiency and scalability, making it suitable for very large document datasets. Its ability to handle noise and find clusters of varying shapes and sizes ensures robust clustering performance. BIRCH’s hierarchical approach allows for multi-level analysis of document clusters, which is beneficial for constructing a detailed and scalable knowledge graph.

**Algorithm: BIRCH**
Input: data points X, branching factor B, threshold T
Output: cluster assignments

1. Initialize CF tree
2. For each point:
   a. Insert point into CF tree
      i. Update leaf entry
      ii. If leaf overflows, split leaf
      iii. If root overflows, split root
3. Cluster leaf entries
4. Return cluster assignments

Once formed, each cluster is assigned a representative node (cluster node) which, as the name suggests, represents the cluster. It aggregates all the embeddings of each document, averaging them for its own, and all the unique tags it parents.

3.2.4 Edge Assignment

Edge assignment is a crucial step in constructing a knowledge graph from document embeddings. Once similarity metrics between document embeddings are computed and clusters are assigned, the next step is to determine how to connect these documents (nodes) to form a meaningful and efficient graph. In this work, we employ various algorithms for edge assignment, including K-Nearest Neighbors (KNN), Minimum Spanning Tree (MST), and others, to ensure robust connectivity and structure.

3.2.4.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is used to connect each node to its $K$ most similar nodes based on the similarity metrics. This ensures that each document has a consistent number of connections, leading to a balanced and uniformly connected graph.
Implementation and Choice: KNN was implemented by calculating the similarity between each document and connecting each node to its top $K$ nearest neighbors. This method was chosen for its simplicity and efficiency in creating a well-distributed network of connections.

Parameters:

- Number of Neighbors $K$: Specifies the number of nearest neighbors to connect each node.

3.2.4.2 Minimum Spanning Tree (MST)

To ensure that all nodes are connected with the minimum total edge weight, we employ the Minimum Spanning Tree (MST) algorithm. This helps in creating a backbone structure for the graph.

Implementation and Choice: MST was implemented to connect all nodes in a way that minimizes the overall connection cost, ensuring no cycles. It was chosen to provide a robust underlying structure that guarantees all nodes are interconnected efficiently.

Parameters:

- None: MST primarily relies on the similarity matrix to determine the minimum spanning connections.
3.2.4.3 Threshold-Based Edge Assignment

In threshold-based edge assignment, edges are created between nodes if their similarity exceeds a predefined threshold. This method is flexible and ensures that only strongly related documents are connected.

**Implementation and Choice:** Edges were created by checking if the similarity score between nodes exceeded a certain threshold. This method was chosen for its ability to filter out weak connections, ensuring only meaningful links are retained.

**Parameters:**

- *Threshold:* The similarity score threshold above which an edge is created.

3.2.4.4 Mutual K-Nearest Neighbors (Mutual KNN)

Mutual K-Nearest Neighbors edge assignment ensures that edges are only created if both nodes consider each other as one of their K-nearest neighbors, leading to a more reliable connection.

**Implementation and Choice:** Mutual KNN was implemented by creating edges only when both nodes were among each other’s nearest neighbors. This method was chosen to enhance the reliability of connections by ensuring mutual agreement.

**Parameters:**

- *Number of Neighbors (K):* Specifies the number of nearest neighbors to consider for mutual connections.
3.2.4.5 Spectral Clustering

Spectral clustering uses the similarity matrix to perform clustering, and edges are created based on the resulting clusters. This method is effective in identifying complex structures in the data.

**Implementation and Choice:** Spectral clustering was implemented by using the similarity matrix to assign nodes to clusters, and edges were created between nodes within the same cluster. This method was chosen for its ability to uncover complex patterns and structures within the data.

**Parameters:**

- *Number of Clusters:* Specifies the number of clusters to form.

3.2.5 Guaranteeing Connectedness:

As a final touch, each cluster representative node is connected to an overall parent node so that every node may have a path between every other node and the weight assigned to that edge is the similarity between the parent node’s embedding–aggregated from all embeddings–and each cluster node’s embedding.

In this chapter, we have detailed the methodologies employed in our study, from data preprocessing and embeddings to clustering and edge assignment. By systematically combining these techniques, we aim to construct a comprehensive and efficient knowledge graph. The next steps involve integrating these components, testing the pipeline on diverse datasets, and evaluating the performance at different stages. This will allow us to refine our approach and ensure the robustness and generalizability of our methodology across various contexts and applications.
Chapter 4

EVALUATION METHODOLOGY

This evaluation is structured as a case study to systematically assess the performance and robustness of our proposed methodology for generating knowledge graphs from digital documents.

We outline the metrics and methodologies used to evaluate the performance and effectiveness of our knowledge graph construction pipeline. The evaluation is conducted at different critical stages: after embedding and similarity scoring, after clustering, and after edge assignment. Each stage is assessed using specific metrics to ensure a comprehensive analysis of the pipeline’s performance.

Due to compute limitations during the timespan of this project, 2000 documents were used out of the Medium dataset and 1000 out of the ArXiv dataset. As the categories of data were decently randomly spread over thousands of rows, we selected the first 2000, or 1000 for each dataset.

We have two sets prepped for testing: Medium articles (n=2000) and Medium (n=200). These sizes were chosen due to practicality and time restraint reasons.

4.1 Embedding and Similarity Scores

After embedding the documents and computing the similarity scores, we evaluate the consistency and discrimination power of the embeddings by comparing similarity scores between entries that share one or more tags with those that do not.
Metrics:

- **Average Similarity**: Calculate the average similarity score between documents that share one or more tags and compare it to the average similarity score between all document pairs.

- **Median Similarity**: Calculate the median similarity score between documents that share one or more tags and compare it to the median similarity score between all document pairs.

- **Standard Deviation**: Calculate the standard deviation of similarity scores between documents that share one or more tags and compare it to the standard deviation of similarity scores between all document pairs.

**Purpose**: These metrics help in understanding how well the embeddings capture the semantic similarity between documents with shared tags, indicating the quality of the embedding model. A general rule of thumb is that the greater the relative difference between the two sets of documents that share one or more tags and all tags, the better the model can semantically differentiate documents.

### 4.2 Clustering Evaluation

After clustering the documents, we assess how well the clustering algorithm groups documents with the same tags together. This involves analyzing the clustering effectiveness for a sample of the most popular tags. Every embedding model is tested with every clustering algorithms and its corresponding hyperparameters.

**Metrics:**
• **Tag Concentration Purity:** For each of the $K$ most popular tags, find the cluster that contains the highest number of documents with that tag. Calculate the percentage of all documents with that tag contained within this cluster.

• **Homogeneity:** A clustering result satisfies homogeneity if all of its clusters contain only data points that are members of a single tag. In other words, a cluster is homogeneous if it does not mix documents from different tags; like precision.

• **Completeness:** A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster. In other words, a class is completely assigned to a single cluster without being split across multiple clusters; like recall.

• **V-measure:** The combination of Homogeneity and Completeness. See 2.4.1 for more information; like f1-score.

**Purpose:** These metrics evaluates the clustering algorithm’s ability to group documents with the same tags together, which is crucial for the semantic coherence of the clusters and overall graph. However, we must be mindful of metrics like V-measure with smaller datasets. The closer the number of clusters, like in Kmeans and GMM, is to the total number of points, the higher it’s going to be. For example, if we have 200 points and 200 clusters, every cluster will be perfectly homogeneous but have awful completeness.

### 4.3 Edge Assignment Evaluation

After the edge assignment, we evaluate the effectiveness of the graph structure by analyzing the connectivity of documents with shared tags. This involves using a
breadth-first search to measure the percentage and number of connected nodes that share one or more tags to various depths (i.e. depth 1 is included when processing depth 2).

**Metrics:**

- **Tag Connectivity at Various Depths:** Using breadth-first search, calculate the percentage and number of nodes that share one or more tags for different depths ($d = 1, 2, 3, \ldots$). This measures how well the graph structure connects related documents.

**Purpose:** This metric evaluates the effectiveness of the edge assignment algorithm in creating a connected and meaningful graph structure that groups semantically related documents together.

### 4.4 All Models and Algorithms Tested:

**Embedding Models:**

- sentence-transformers/all-MiniLM-L6-v2
- sentence-transformers/all-mpnet-base-v2
- nomic-ai/nomic-embed-text-v1.5
- google-bert/bert-base-uncased
- allenai/specter
- gensim/word2vec
Clustering Algorithms:

- k-means
  - \( k = [1, 2, 5, 10, 15, 50, 100] \)

- DBSCAN
  - \( \text{eps} = [0.1, 0.3, 0.5, 0.7, 1.0] \)
  - \( \text{min\_samples} = [3, 5, 10, 15] \)

- Gaussian Mixture Models (GMM)
  - \( n\_\text{components} = [1, 2, 5, 10, 15, 50, 100] \)

- Birch
  - \( k = [1, 2, 5, 10, 15, 50, 100] \)

Edge Assignment Algorithms:

- random_edges
  - \( \text{num\_edges\_per\_node} = [5, 10, 25] \)

- knn
  - \( k = [1, 2, 5, 10, 15, 50, 70, 100] \)

- knn_mst
  - \( k = [1, 2, 5, 10, 15, 50, 70, 100] \)

- threshold_based_edge_assignment
  - \( \text{threshold} = [0.5, 0.7, 0.9] \)
• mutual_knn_edge_assignment
  - k = [1, 2, 5, 10, 15, 50, 70, 100]

• spectral_clustering_edge_assignment
  - n_clusters = [1, 2, 5, 10, 15, 50, 70, 100]

By employing these evaluation metrics at different stages of the pipeline, we aim to gain a comprehensive understanding of the performance and effectiveness of our methodology in the context of this case study. This approach not only allows for a detailed analysis of each component but also provides insights into the overall robustness and applicability of our knowledge graph construction process. By iteratively refining our methods based on these evaluations, we can ensure that our approach is both reliable and adaptable to diverse datasets and varying conditions. Ultimately, this thorough evaluation will contribute to the development of a more generalizable and robust methodology for constructing knowledge graphs from digital documents.
Chapter 5

RESULTS

5.1 Medium Articles (n=2000)

5.1.1 Embedding and Similarity Scores Analysis

The bar plot in figure 5.2, “Difference in Similarity Scores between Shared Tags and All Nodes”, visualizes the differences in median similarity scores between documents with shared tags and documents that don’t share any tags for the various embedding models using three different similarity metrics: cosine, euclidean, and soft cosine. Here are the key observations and analyses:

Embedding Models Comparison:

- **all-MiniLM-L6-v2 (minilm)**: Shows the highest difference in similarity scores across all three metrics (cosine: 0.030, euclidean: 0.027, soft cosine: 0.023). This indicates that minilm is highly effective in distinguishing documents with shared tags compared to those without.

- **all-mpnet-base-v2 (mpnet)**: Also performs very well across all metrics (cosine: 0.030, euclidean: 0.025, soft cosine: 0.028), showing significant differences in similarity scores.

- **nomic-embed-text-v1.5 (nomic)**: Shows moderate differences in similarity scores (cosine: 0.017, euclidean: 0.022, soft cosine: 0.012), indicating decent performance but not as high as minilm and mpnet.
Figure 5.1: Medium Articles (n=2000): Top 10 common tags in the set.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>425</td>
</tr>
<tr>
<td>Art</td>
<td>350</td>
</tr>
<tr>
<td>Science</td>
<td>189</td>
</tr>
<tr>
<td>Business</td>
<td>177</td>
</tr>
<tr>
<td>Marketing</td>
<td>149</td>
</tr>
<tr>
<td>Productivity</td>
<td>140</td>
</tr>
<tr>
<td>Writing</td>
<td>82</td>
</tr>
<tr>
<td>Health</td>
<td>79</td>
</tr>
<tr>
<td>Coronavirus</td>
<td>67</td>
</tr>
<tr>
<td>Entertainment</td>
<td>58</td>
</tr>
</tbody>
</table>
• **allenai/specter2 (specter):** Displays lower differences in similarity scores (cosine: 0.008, euclidean: 0.014, soft cosine: 0.002), suggesting it is less effective than minilm and mpnet in distinguishing documents with shared tags.

• **gensim/word2vec (word2vec):** Shows moderate performance (cosine: 0.009, euclidean: 0.012, soft cosine: 0.006), but not as effective as the top performers.

• **google-bert (bert):** Has the lowest differences in similarity scores (cosine: 0.008, euclidean: 0.013, soft cosine: 0.001), indicating it is the least effective among the models tested for this task.

**Metric Conclusion:**

• **Cosine:** Generally shows the highest difference in similarity scores for most models, especially for minilm and mpnet.
• **Euclidean**: While lower than cosine in the models with the greatest difference, it is the greatest differentiator on average.

• **Soft Cosine**: Shows inconsistent performance, with differences in similarity scores way lower than both cosine and Euclidean for most models.

Based on the observed differences in similarity scores, the top 3 embedding models are (at this stage):

• **minilm with cosine similarity**: (Cosine: 0.030, Euclidean: 0.027, Soft Cosine: 0.023)

• **mpnet with cosine similarity**: (Cosine: 0.030, Euclidean: 0.025, Soft Cosine: 0.028)

• **nomic with Euclidean similarity**: (Cosine: 0.017, Euclidean: 0.022, Soft Cosine: 0.012)

These models and corresponding similarity metrics show the highest relative differences in similarity scores between documents with shared tags and all documents indicating better semantic differentiation capabilities. It makes sense that these worked the best so far as these are modern transformers fine-tuned over general and casual writing which fits Medium’s description. Specter embeddings are highly fine-tuned on technical papers, not like Medium articles. Moving forward, we will only analyze the results from these three.
5.1.2 Clustering Evaluation

Here, we assess how well the clustering algorithm groups documents with the same tags together. This involves analyzing the clustering effectiveness with homogeneity, completeness, and tag concentration purity.

5.1.2.1 Homogeneity

**Definition:** A clustering result satisfies homogeneity if all its clusters contain only data points that are members of a single tag. We calculate the most common tag within each cluster and define a cluster’s *homogeneity score* of each cluster as the number of data points with that tag over the total number of data points of the cluster. In other words, a cluster is more homogenous the less it mixes documents from different tags.

This table 5.3 presents the top 10 configurations ranked by homogeneity for the Medium articles dataset with n=2000. The configurations include various combinations of embedding models and clustering algorithms. Here are the key observations and conclusions:

**Best Clustering Algorithm:**

- **GMM (100 components):** Demonstrates the highest homogeneity values overall, making it the best clustering algorithm for achieving homogeneous clusters in this dataset. It works particularly well with mpnet and nomic embeddings.

- **K-means (100 clusters):** A close second, also showing strong homogeneity values and performing well with mpnet, nomic, and minilm embeddings.
Figure 5.3: Medium Articles (n=2000): Top 10 configurations by homogeneity.

- **Birch (100 clusters)**: Performs well but slightly lower than GMM and K-means. It is still a strong contender, especially with mpnet and minilm embeddings.

**Best Embedding Models:**

- **mpnet**: Shows the highest and most consistent performance across different clustering algorithms.

- **nomic**: Also performs well, particularly with GMM and K-means.

- **minilm**: Effective with K-means, Birch, and GMM, though slightly behind mpnet and nomic.

- **specter**: Less consistent but still capable of achieving high homogeneity with GMM.
Overall, for the Medium articles dataset with $n = 2000$, the combination of mpnet embeddings with GMM or K-means clustering algorithms yields the best homogeneity. Birch also performs well, particularly with mpnet and minilm embeddings. These insights guide the selection of clustering strategies for maintaining semantically coherent knowledge graphs.

5.1.2.2 Completeness

**Definition:** A clustering result satisfies completeness if all the data points that are members of a given tag are elements of the same cluster. We calculate the completeness score of each tag as the number of data points from that tag in the largest cluster over the total number of data points with that tag. In other words, a tag is more completely assigned the less it is split across multiple clusters.

This section presents the top 10 configurations ranked by completeness for the Medium articles dataset with $n = 2000$. The table 5.4 below summarizes the results:

**Best Clustering Algorithm:**

- **K-means (5 clusters):** Demonstrates the highest completeness values overall, making it the best clustering algorithm for achieving high completeness in this dataset. It works particularly well with nomic, mpnet, and minilm embeddings.

- **GMM (5 components):** A close second, also showing strong completeness values and performing well with nomic and mpnet embeddings.

- **Birch (2 clusters):** Performs well, especially with nomic and mpnet embeddings, but appears less frequently than K-means and GMM.
Figure 5.4: Medium Articles (n=2000): Top 10 configurations by completeness.

Best Embedding Models:

- **nomic**: Shows the highest and most consistent performance across different clustering algorithms, particularly with K-means and GMM.

- **mpnet**: Also performs well, particularly with K-means and GMM.

- **specter**: Less consistent but still capable of achieving high completeness with K-means.

- **minilm**: Shows moderate performance with K-means.

Overall, for the Medium articles dataset with \( n = 2000 \), the combination of nomic embeddings with K-means or GMM clustering algorithms yields the best completeness. Birch also performs well, particularly with nomic and mpnet embeddings. These insights guide the selection of clustering strategies for achieving high completeness in knowledge graphs.
5.1.2.3 V-measure

**Definition:** V-measure is the harmonic mean of homogeneity and completeness, providing a balanced measure of both aspects.

This table 5.5 presents the top 10 configurations ranked by V-measure for the Medium articles dataset with n=2000. Here are the key observations and conclusions:

**Best Clustering Algorithm:**

- **GMM (100 components):** Demonstrates the highest V-measure values overall, making it the best clustering algorithm for achieving high V-measure in this dataset. It works particularly well with mpnet and nomic embeddings.

- **K-means (100 clusters):** A close second, also showing strong V-measure values and performing well with mpnet, nomic, and minilm embeddings.

- **Birch (100 clusters):** Performs well, especially with mpnet and nomic embeddings, but appears less frequently than GMM and K-means.

**Best Embedding Models:**

- **mpnet:** Shows the highest and most consistent performance across different clustering algorithms.

- **nomic:** Also performs well, particularly with GMM and Birch.

- **minilm:** Effective with K-means and GMM, though slightly behind mpnet and nomic.
Overall, for the Medium articles dataset with \( n = 2000 \), the combination of mpnet embeddings with GMM or K-means clustering algorithms yields the best V-measure. Birch also performs well, particularly with mpnet and nomic embeddings. These insights guide the selection of clustering strategies for achieving high V-measure in knowledge graphs.

### 5.1.2.4 Tag Concentration Purity

**Definition:** Tag concentration purity is defined in section 4.2 as “for each of the \( K \) most popular tags, find the cluster that contains the highest number of documents with that tag. Calculate the percentage of all documents with that tag contained within this cluster.”

This section presents the top 10 configurations ranked by tag concentration purity for the Medium articles dataset with \( n = 2000 \). The configurations include various com-

<table>
<thead>
<tr>
<th>embedding_model</th>
<th>clusterer</th>
<th>homogeneity</th>
<th>completeness</th>
<th>v-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpnet</td>
<td>gmm100</td>
<td>0.308</td>
<td>0.221</td>
<td>0.257346</td>
</tr>
<tr>
<td>nomic</td>
<td>gmm100</td>
<td>0.304</td>
<td>0.220</td>
<td>0.255267</td>
</tr>
<tr>
<td>mpnet</td>
<td>kmeans100</td>
<td>0.302</td>
<td>0.220</td>
<td>0.254559</td>
</tr>
<tr>
<td>mpnet</td>
<td>birch100</td>
<td>0.304</td>
<td>0.218</td>
<td>0.253916</td>
</tr>
<tr>
<td>nomic</td>
<td>birch100</td>
<td>0.284</td>
<td>0.228</td>
<td>0.252937</td>
</tr>
<tr>
<td>nomic</td>
<td>kmeans100</td>
<td>0.297</td>
<td>0.217</td>
<td>0.250774</td>
</tr>
<tr>
<td>mpnet</td>
<td>gmm50</td>
<td>0.271</td>
<td>0.228</td>
<td>0.247647</td>
</tr>
<tr>
<td>minilm</td>
<td>kmeans100</td>
<td>0.294</td>
<td>0.213</td>
<td>0.247030</td>
</tr>
<tr>
<td>mpnet</td>
<td>kmeans50</td>
<td>0.268</td>
<td>0.229</td>
<td>0.246970</td>
</tr>
<tr>
<td>minilm</td>
<td>gmm100</td>
<td>0.291</td>
<td>0.212</td>
<td>0.245296</td>
</tr>
</tbody>
</table>

Figure 5.5: Medium Articles (n=2000): Top 10 configurations by completeness.
binations of embedding models and clustering algorithms. The table 5.6 summarizes the results:

Best Clustering Algorithm:

- **Birch (100 and 50 clusters):** Demonstrates the highest tag concentration purity values overall, making it the best clustering algorithm for achieving high tag concentration purity in this dataset. It works particularly well with mpnet embeddings.

- **GMM (100 and 50 components):** A close second, also showing strong tag concentration purity values and performing well with mpnet and nomic embeddings.

- **K-means (100 and 50 clusters):** Performs well, especially with minilm and mpnet embeddings, but appears slightly less frequently than Birch and GMM.

Best Embedding Models:

- **mpnet:** Shows the highest and most consistent performance across different clustering algorithms.

- **minilm:** Also performs well, particularly with K-means and Birch.

- **nomic:** Effective with GMM, though slightly behind mpnet and minilm.

For the Medium articles dataset with $n = 2000$, the combination of mpnet embeddings with Birch or GMM clustering algorithms yields the best tag concentration purity. K-means also performs well, particularly with minilm and mpnet embeddings. These
<table>
<thead>
<tr>
<th>embedding_model</th>
<th>clusterer</th>
<th>average_tag_score</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpnet</td>
<td>birch100</td>
<td>0.781133</td>
</tr>
<tr>
<td>mpnet</td>
<td>birch50</td>
<td>0.731000</td>
</tr>
<tr>
<td>minilm</td>
<td>kmeans100</td>
<td>0.720667</td>
</tr>
<tr>
<td>mpnet</td>
<td>gmm100</td>
<td>0.719267</td>
</tr>
<tr>
<td>nomic</td>
<td>gmm50</td>
<td>0.712600</td>
</tr>
<tr>
<td>mpnet</td>
<td>kmeans100</td>
<td>0.706933</td>
</tr>
<tr>
<td>mpnet</td>
<td>gmm50</td>
<td>0.704533</td>
</tr>
<tr>
<td>minilm</td>
<td>birch100</td>
<td>0.686200</td>
</tr>
<tr>
<td>minilm</td>
<td>gmm100</td>
<td>0.683267</td>
</tr>
<tr>
<td>mpnet</td>
<td>kmeans50</td>
<td>0.680800</td>
</tr>
</tbody>
</table>

Figure 5.6: Medium Articles (n=2000): Top 10 configurations for tag concentration purity.
insights guide the selection of clustering strategies for achieving high tag concentration purity in knowledge graphs.

5.1.3 Edge Assignment Evaluation

In this section, we evaluate the effectiveness of different edge assignment configurations by measuring the percentage of connected nodes at varying depths (1, 2, 3). The goal is to understand how well different combinations of embedding models, similarity metrics, clustering algorithms, and edge constructors maintain connectivity within the knowledge graph. See section 4.3.

The results are visualized in figure 5.7 and summarized in the table figure 5.8. Each figure shows the top 10 configurations based on their performance across three depths. This is decided by which has the greatest first depth tag connectivity.

It is good to see a great drop in the percentage connected metric as this indicates nearby nodes (depth=1) are more related than all the nodes found further away (depth=2, 3). However, we must also consider if a more gradual drop at each depth is more beneficial for information retrieval.

Overall, the configuration with the greatest first depth tag connectivity and greatest drop is edge assignment with mutual K-nearest neighbors (k=50) with nomic embeddings, Euclidean similarity, and GMM clustering with ratios of 0.724, 0.047, and 0.003 at depths 1, 2, and 3 respectively.

However, if we want to be on the side of spreading information with a less dramatic drop, the top configuration is edge assignment with threshold (t=0.7) with minilm embeddings, cosine similarity, and kmeans (k=70). It produces ratios of 0.479, 0.224, and 0.114 at depths 1, 2, and 3 respectively.
Figure 5.7: Medium Articles (n=2000): Percentage Connected Plot, (depth=1, 2, 3)

Figure 5.8: Medium Articles (n=2000): Percentage Connected Table, (depth=1, 2, 3)
Metric Conclusion: In summary, the evaluation demonstrates that the choice of edge assignment configuration significantly influences the connectivity patterns within the knowledge graph. Depending on the desired application—whether it is prioritizing strong immediate connections or ensuring more consistent connectivity across different depths—different configurations may be preferred. The findings provide valuable guidance for selecting the most appropriate configuration to balance isolation and connectivity in a knowledge graph:

- Nomic embeddings provide the greatest first-depth tag connectivity with a sharp drop-off.
- MiniLM embeddings provide the most gradual descent in tag connectivity which can be desirable.
- Euclidean similarity has the best record here.
- GMM (n=50) and K-means (k=50) are the top two clustering algorithms.
- Mutual KNN (k=50) is used for the top 5 configurations.

5.2 Medium Articles (n=200)

5.2.1 Embedding and Similarity Scores Analysis

For the subset of 200 Medium articles, the embedding models and their performance in distinguishing documents with shared tags were analyzed using the same metrics as for the larger dataset. See figure 5.10 Here are the key observations and analyses:

Embedding Models Comparison:
<table>
<thead>
<tr>
<th>Tag</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science</td>
<td>38</td>
</tr>
<tr>
<td>Art</td>
<td>33</td>
</tr>
<tr>
<td>Technology</td>
<td>23</td>
</tr>
<tr>
<td>Productivity</td>
<td>22</td>
</tr>
<tr>
<td>Writing</td>
<td>17</td>
</tr>
<tr>
<td>Business</td>
<td>16</td>
</tr>
<tr>
<td>Marketing</td>
<td>13</td>
</tr>
<tr>
<td>Coronavirus</td>
<td>8</td>
</tr>
<tr>
<td>Health</td>
<td>7</td>
</tr>
<tr>
<td>Entertainment</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 5.9: Medium Articles (n=200): Top 10 common tags in the set.
Figure 5.10: Medium Articles (n=200): Median Difference in Similarity Scores between Shared Tags and All Nodes.

- **all-MiniLM-L6-v2 (minilm)**: Continues to show high effectiveness in distinguishing documents with shared tags, demonstrating significant differences in similarity scores across all three metrics.

- **all-mpnet-base-v2 (mpnet)**: Also performs well, with substantial differences in similarity scores, indicating good semantic differentiation.

- **nomic-embed-text-v1.5 (nomic)**: Shows moderate performance, with noticeable differences in similarity scores but not as high as minilm and mpnet.

- **allenai/specter2 (specter)**: Displays lower differences in similarity scores, suggesting less effectiveness compared to minilm and mpnet.

- **gensim/word2vec (word2vec) and google-bert (bert)**: Both show moderate to low performance, with smaller differences in similarity scores.

**Metric Conclusion:**
• **Cosine Similarity**: Shows the highest difference in similarity scores for minilm and mpnet, but is not the favoriate for the rest.

• **Euclidean Similarity**: Shows consistent performance, with higher differences in similarity scores for most models.

• **Soft Cosine Similarity**: While lower than cosine in the top models, it provides significant differentiation on average with the modern transformer models.

Based on the observed differences in similarity scores, the top 3 embedding models remain:

• **minilm with cosine similarity**: (Cosine: 0.041, Euclidean: 0.038, Soft Cosine: 0.036)

• **mpnet with cosine similarity**: (Cosine: 0.037, Euclidean: 0.033, Soft Cosine: 0.033)

• **nomic with Euclidean similarity**: (Cosine: 0.019, Euclidean: 0.025, Soft Cosine: 0.035)

These models show the highest relative differences in similarity scores, indicating better semantic differentiation capabilities.

### 5.2.2 Clustering Evaluation

Here, we assess how well the clustering algorithm groups documents with the same tags together. This involves analyzing the clustering effectiveness with homogeneity, completeness, and tag concentration purity.
5.2.2.1 Homogenity

This is the top 10 configurations ranked by homogeneity for the Medium articles dataset with \( n = 200 \). The configurations include various combinations of embedding models and clustering algorithms. The table 5.11 summarizes the results:

**Best Clustering Algorithm:**

- **GMM (100 components):** Demonstrates the highest homogeneity values overall, making it the best clustering algorithm for achieving homogeneous clusters in this dataset. It works particularly well with mpnet and nomic embeddings.

- **K-means (100 clusters):** A close second, also showing strong homogeneity values and performing well with mpnet, nomic, and minilm embeddings.

- **Birch (100 clusters):** Performs well, especially with mpnet and minilm embeddings, but appears slightly less frequently than GMM and K-means.

**Best Embedding Models:**

- **mpnet:** Shows the highest and most consistent performance across different clustering algorithms.

- **nomic:** Also performs well, particularly with GMM and K-means.

- **minilm:** Effective with K-means, Birch, and GMM, though slightly behind mpnet and nomic.

- **specter:** Less consistent but still capable of achieving high homogeneity with GMM.
For the Medium articles dataset with \( n = 200 \), the combination of mpnet embeddings with GMM or K-means clustering algorithms yields the best homogeneity. Birch also performs well, particularly with mpnet and minilm embeddings. These insights guide the selection of clustering strategies for maintaining semantically coherent knowledge graphs.

### Completeness

Here are the top 10 configurations ranked by completeness for the Medium articles dataset with \( n = 200 \). The configurations include various combinations of embedding models and clustering algorithms. The table 5.12 summarizes the results:

**Best Clustering Algorithm:**

<table>
<thead>
<tr>
<th>embedding_model</th>
<th>clusterer</th>
<th>homogeneity</th>
<th>completeness</th>
<th>v-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpnet</td>
<td>kmeans100</td>
<td>0.770</td>
<td>0.435</td>
<td>0.555934</td>
</tr>
<tr>
<td>mpnet</td>
<td>gmm100</td>
<td>0.756</td>
<td>0.432</td>
<td>0.549818</td>
</tr>
<tr>
<td>nomic</td>
<td>gmm100</td>
<td>0.748</td>
<td>0.438</td>
<td>0.552486</td>
</tr>
<tr>
<td>mpnet</td>
<td>birch100</td>
<td>0.747</td>
<td>0.427</td>
<td>0.543388</td>
</tr>
<tr>
<td>nomic</td>
<td>kmeans100</td>
<td>0.742</td>
<td>0.429</td>
<td>0.543669</td>
</tr>
<tr>
<td>specter</td>
<td>gmm100</td>
<td>0.736</td>
<td>0.426</td>
<td>0.539649</td>
</tr>
<tr>
<td>minilm</td>
<td>birch100</td>
<td>0.733</td>
<td>0.429</td>
<td>0.541234</td>
</tr>
<tr>
<td>minilm</td>
<td>kmeans100</td>
<td>0.727</td>
<td>0.421</td>
<td>0.533218</td>
</tr>
<tr>
<td>minilm</td>
<td>gmm100</td>
<td>0.721</td>
<td>0.414</td>
<td>0.525981</td>
</tr>
<tr>
<td>nomic</td>
<td>birch100</td>
<td>0.717</td>
<td>0.445</td>
<td>0.549165</td>
</tr>
</tbody>
</table>
• **K-means (2 and 5 clusters):** Demonstrates the highest completeness values overall, making it the best clustering algorithm for achieving high completeness in this dataset. It works particularly well with mpnet and nomic embeddings.

• **GMM (2 and 5 components):** A close second, also showing strong completeness values and performing well with mpnet and minilm embeddings.

• **Birch (2, 5, and 100 clusters):** Performs well, especially with mpnet and minilm embeddings, but appears slightly less frequently than K-means and GMM.

**Best Embedding Models:**

- **mpnet:** Shows the highest and most consistent performance across different clustering algorithms.

- **nomic:** Also performs well, particularly with K-means and GMM.

- **minilm:** Effective with Birch, K-means, and GMM, though slightly behind mpnet and nomic.

For the Medium articles dataset with \( n = 200 \), the combination of mpnet embeddings with K-means or GMM clustering algorithms yields the best completeness. Birch also performs well, particularly with mpnet and minilm embeddings. These insights guide the selection of clustering strategies for achieving high completeness in knowledge graphs.
Here are the 10 configurations ranked by V-measure for the Medium articles dataset with \( n = 200 \). The configurations include various combinations of embedding models, aggregation methods, and clustering algorithms. The table 5.13 summarizes the results:

**Best Clustering Algorithm:**

- **GMM (100 components):** Demonstrates the highest V-measure values overall, making it the best clustering algorithm for achieving high V-measure in this dataset. It works particularly well with mpnet and nomic embeddings.

- **K-means (100 clusters):** A close second, also showing strong V-measure values and performing well with mpnet, nomic, and minilm embeddings.
Figure 5.13: Medium Articles (n=200): Top 10 configurations for V-measure.

- **Birch (100 clusters):** Performs well, especially with mpnet and minilm embeddings, but appears slightly less frequently than GMM and K-means.

**Best Embedding Models:**

- **mpnet:** Shows the highest and most consistent performance across different clustering algorithms.

- **nomic:** Also performs well, particularly with GMM and Birch.

- **minilm:** Effective with Birch, K-means, and GMM, though slightly behind mpnet and nomic.

- **specter:** Less consistent but still capable of achieving high V-measure with GMM and K-means.

For the Medium articles dataset with $n = 200$, the combination of mpnet embeddings with GMM or K-means clustering algorithms yields the best V-measure. Birch
also performs well, particularly with mpnet and minilm embeddings. These insights guide the selection of clustering strategies for achieving high V-measure in knowledge graphs.

5.2.2.4 Tag Concentration Purity

This is the top 10 configurations ranked by tag concentration purity for the Medium articles dataset with \( n = 200 \). The configurations include various combinations of embedding models, aggregation methods, and clustering algorithms. The table 5.14 summarizes the results:

Best Clustering Algorithm:

- **K-means (100 clusters)**: Demonstrates the highest tag concentration purity values overall, making it the best clustering algorithm for achieving high tag concentration purity in this dataset. It works particularly well with mpnet, specter, and nomic embeddings.

- **GMM (100 components)**: A close second, also showing strong tag concentration purity values and performing well with specter, mpnet, and nomic embeddings.

- **Birch (100 and 70 clusters)**: Performs well, especially with mpnet and minilm embeddings, but appears slightly less frequently than K-means and GMM.

Best Embedding Models:
• **mpnet**: Shows the highest and most consistent performance across different clustering algorithms.

• **specter**: Also performs well, particularly with GMM and K-means.

• **minilm**: Effective with Birch and K-means, though slightly behind mpnet and specter.

• **nomic**: Shows good performance with GMM and K-means.

### 5.2.3 Edge Assignment Evaluation

• **Greatest Isolation**: Edge assignment with random neighbors (n=5) using mpnet embeddings, soft cosine, and K-means (k=50) shows the highest isolation...
Figure 5.15: Medium Articles (n=200): Performance of the Top 10 Configurations Across All Depths

<table>
<thead>
<tr>
<th>embedding_model</th>
<th>similarity</th>
<th>clusterer</th>
<th>edge_constructor</th>
<th>depth=1</th>
<th>depth=2</th>
<th>depth=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpnet</td>
<td>soft_cosine</td>
<td>kmeans50</td>
<td>random5</td>
<td>0.454</td>
<td>0.080</td>
<td>0.223</td>
</tr>
<tr>
<td>mimin</td>
<td>cosine</td>
<td>kmeans70</td>
<td>random10</td>
<td>0.413</td>
<td>0.059</td>
<td>0.288</td>
</tr>
<tr>
<td>mpnet</td>
<td>euclidean</td>
<td>kmeans70</td>
<td>random10</td>
<td>0.406</td>
<td>0.074</td>
<td>0.303</td>
</tr>
<tr>
<td>nomic</td>
<td>soft_cosine</td>
<td>kmeans50</td>
<td>random5</td>
<td>0.406</td>
<td>0.116</td>
<td>0.203</td>
</tr>
<tr>
<td>mimin</td>
<td>cosine</td>
<td>kmeans50</td>
<td>random10</td>
<td>0.394</td>
<td>0.084</td>
<td>0.347</td>
</tr>
<tr>
<td>mpnet</td>
<td>soft_cosine</td>
<td>gmm50</td>
<td>random5</td>
<td>0.394</td>
<td>0.092</td>
<td>0.231</td>
</tr>
<tr>
<td>mpnet</td>
<td>soft_cosine</td>
<td>kmeans70</td>
<td>random5</td>
<td>0.391</td>
<td>0.063</td>
<td>0.232</td>
</tr>
<tr>
<td>mimin</td>
<td>cosine</td>
<td>gmm50</td>
<td>random10</td>
<td>0.390</td>
<td>0.076</td>
<td>0.295</td>
</tr>
<tr>
<td>mpnet</td>
<td>cosine</td>
<td>gmm50</td>
<td>random5</td>
<td>0.390</td>
<td>0.120</td>
<td>0.283</td>
</tr>
<tr>
<td>mpnet</td>
<td>euclidean</td>
<td>gmm50</td>
<td>random10</td>
<td>0.386</td>
<td>0.096</td>
<td>0.343</td>
</tr>
</tbody>
</table>

Figure 5.16: Medium Articles (n=200): Performance of the Top 10 Configurations Across All Depths
with connectivity ratios of 0.454, 0.080, 0.223 at depths 1, 2, and 3 respectively as seen in figures 5.15 and 5.16.

5.2.3.1 Analysis of Random Edge Assignment Performance

While unexpected, interestingly, in the context of small graphs, random edge assignment can sometimes outperform more sophisticated algorithms. Several factors could contribute to this phenomenon:

- **Simpler Structures**: In small graphs, the inherent structure is simpler, which allows random edge assignments to approximate optimal connectivity by chance more effectively.

- **Higher Connectivity by Chance**: With fewer nodes, the probability of achieving high connectivity through random assignment is greater. Random edges can often create a well-connected graph purely by statistical likelihood.

- **Algorithm Overhead**: Sophisticated algorithms designed for larger, more complex graphs may introduce overhead that does not translate into better performance in smaller graphs. The benefits of these algorithms are more apparent in larger networks where their optimizations are necessary.

- **Edge Density**: Random edge assignments can lead to a denser graph in small networks, resulting in higher immediate connectivity (depth 1), which significantly influences connectivity metrics.

- **Edge Assignment Algorithm Suitability**: Certain edge assignment algorithms are tuned for specific data types or connectivity patterns. If the small graph does not align well with these assumptions, the performance of these algorithms might not surpass that of random edge assignment.
5.3 Summary of Results

Based on the detailed analysis of the various configurations for both larger (n=2000) and smaller (n=200) datasets, we can derive the best pipeline configurations. Here are the top 3 configurations for different dataset sizes, along with the reasoning behind each choice.

Top 3 Pipeline Configurations for Medium Articles (n=2000)

1. **mpnet embeddings with GMM (100 components) clustering, Euclidean similarity, and Mutual KNN (k=50) edge assignment**

   - **Reasoning:** This configuration consistently shows high performance across all clustering metrics—homogeneity, completeness, and V-measure. The combination of mpnet embeddings and GMM clustering achieves a balanced and coherent clustering structure, ensuring that documents with similar tags are grouped effectively. The Mutual KNN edge assignment provides strong first-depth connectivity with significant drops, which is desirable for certain information retrieval applications.

2. **minilm embeddings with K-means (100 clusters) clustering, cosine similarity, and Threshold-based (t=0.7) edge assignment**

   - **Reasoning:** Minilm embeddings, when paired with K-means clustering, show robust performance in both homogeneity and completeness. The cosine similarity metric further enhances the clustering quality, making it an excellent choice for medium-sized datasets. The Threshold-based edge assignment provides a more gradual descent in tag connectivity, which can be beneficial for information retrieval.
3. nomic embeddings with Birch (100 clusters) clustering, Euclidean similarity, and Mutual KNN (k=50) edge assignment

- **Reasoning:** Nomic embeddings with Birch clustering provide strong results for tag concentration purity and V-measure. This configuration is particularly effective in maintaining tag coherence within clusters, making it suitable for larger datasets where semantic differentiation is critical. The Mutual KNN edge assignment ensures strong first-depth connectivity.

Top 3 Pipeline Configurations for Medium Articles (n=200)

1. mpnet embeddings with K-means (100 clusters) clustering, cosine similarity, and Random (n=5) edge assignment

- **Reasoning:** This configuration stands out for its high tag concentration purity and balanced performance across homogeneity and completeness. It is particularly well-suited for smaller datasets due to its effective semantic differentiation and clustering coherence. The Random edge assignment provides surprisingly strong performance in smaller datasets due to simpler structures and higher connectivity by chance.

2. specter embeddings with GMM (100 components) clustering, Euclidean similarity, and Threshold-based (t=0.7) edge assignment

- **Reasoning:** Specter embeddings with GMM clustering achieve high tag concentration purity and V-measure. This combination is effective in smaller datasets, providing a good balance of semantic grouping and cluster purity. The Threshold-based edge assignment ensures a more gradual descent in tag connectivity.
3. minilm embeddings with Birch (100 clusters) clustering, Euclidean similarity, and Mutual KNN (k=50) edge assignment

- **Reasoning:** Minilm embeddings paired with Birch clustering show strong performance in both homogeneity and completeness. This configuration is ideal for maintaining tag coherence and achieving balanced clustering in smaller datasets. The Mutual KNN edge assignment provides strong first-depth connectivity with significant drops, which is desirable for certain applications.

Overall Best Configuration  For Larger Datasets (n=2000):

- **mpnet embeddings with GMM (100 components) clustering, Euclidean similarity, and Mutual KNN (k=50) edge assignment**
  
  This configuration consistently achieves high scores across all evaluated metrics, making it the most robust choice for large datasets. The combination ensures balanced and coherent clusters, which is crucial for effective semantic structuring. The Mutual KNN edge assignment strengthens first-depth connectivity, making it ideal for applications requiring immediate strong connections.

For Smaller Datasets (n=200):

- **mpnet embeddings with K-means (100 clusters) clustering, cosine similarity, and Random (n=5) edge assignment**
  
  This setup is particularly effective in small datasets, showing high tag concentration purity and balanced clustering metrics. It provides excellent semantic differentiation and cluster coherence. The Random edge
Figure 5.17: 100 Medium articles with MiniLM embeddings, cosine similarity, K-means (k=10), and KNN (k=3). Each is labeled with its title and cluster assignment.

assignment works surprisingly well due to higher connectivity chances and simpler structures in smaller datasets.

Additional Insights  While random edge assignments performed surprisingly well in smaller datasets, they are generally not recommended for larger datasets due to their unpredictability and lack of structure. For both dataset sizes, the choice of clustering algorithm (GMM, K-means, Birch) and similarity metric (Euclidean, cosine) plays a significant role in determining the overall effectiveness of the clustering pipeline.

A quality analysis on clustering that should have been done is directly looking at the graph generated and seeing if clustering assignments are appropriate to the human eye rather than mathematical equations based on tags that aren’t potentially biased (discussed in section 6.)
Figure 5.17 shows a graph of 100 Medium articles with MiniLM embeddings, cosine similarity, K-means (k=10), and KNN (k=3). Each is labeled with its title and cluster assignment. The following are some exploratory cases—non-comprehensive—to show off clustering capabilities.

Figure 5.18 focuses on cluster nine, composed of three documents and one representative cluster node. Just by the titles alone, we can tell this is a well-clustered cluster. They are all related to COVID-19. They all have related tags too including “Health” and “Coronavirus.”

Figure 5.19 focuses on cluster seven, composed of nine documents and one representative node. It’s confusing why a document titled “Exploring New York City Restaurants” is so closely connected to all the other documents that seem to focus on sustainability by comparing the titles. Perhaps, there may be some mention of restaurant sustainability in the article. However, reading the actual article shows...
there is nothing related to sustainability. It is purely about the popularity of restaurants. So, the combination of the embedding model and clustering failed here. The tags also are not shared as “Exploring New York City Restaurants” has a single tag of Data Visualization while the rest have tags such as “Environment.”

Finally, figure 5.20 focuses on cluster two composed of six documents, which looks like a case of leftover documents as the clustering algorithm only allows ten clusters which means something that might now belong together is forcefully squished into one cluster. If we were to cluster these manually, we would have four clusters:

1. The two green nodes representing “Music Star Alex Boye Is Doing an Anti-Suicide Concert. The Former Member of the Mormon Tabernacle Choir Performed in an Anti-Gay…” and “A Social Worker Offered Mormon Lingo to Me When I Was in Crisis” look like they belong together as they focus on controversial Mormon-related situations.

3. “AI Diagnoses Alzheimer’s With More Than 95% Accuracy” would probably best match with a cluster of “AI” or “Health” but the intersection of both would be best.
6.1 Threats to Validity

In this section, we address several potential threats to the validity of our case study and its findings:

6.1.1 Dataset Size and Variety

The document sets used in this study (2000 and 200) are relatively small and specific to one type of document (Medium articles). To enhance the robustness of our methods, it is necessary to perform multiple runs with different random samples of 2000 or at least 200 documents. This would provide an average performance measure and demonstrate the consistency of our approach across varied datasets.

6.1.2 Dependency on Specific Tag Distributions

Our recommendations for the number of edges assigned, parameter settings, and optimal number of clusters are based on the tag distributions observed in this case study. These recommendations might not hold for other datasets with different visible or invisible tag distributions. Therefore, further validation of diverse datasets is required to generalize our findings.
6.1.3 Tag Aggregation Methodology

The aggregation of tags into broader categories (e.g., combining “poetry” into “art”) was done subjectively. This lack of an objective methodology could introduce bias, affecting the clustering results. Future work should consider a more rigorous and transparent approach to tag aggregation.

6.1.4 Practical Examples and Clustering Quality

To provide practical insights into the quality of clustering, we need to include examples of good and bad clustering decisions. Showing actual articles that illustrate these decisions would help in understanding the practical implications of our clustering methodology like seen in section 5.3.

6.1.5 Subjectivity of Tagging

The evaluation of our method depends on tags that are inherently subjective. Each Medium article’s tags are created by different individuals who might tag the same document differently based on their interpretation. This subjectivity can affect the perceived accuracy of our clustering and edge assignment methods. Exploring ways to reduce this subjectivity or to account for it in the evaluation metrics could strengthen the study.

6.1.6 Practical Use Case: Digital Libraries and Academic Research

Digital libraries and academic institutions are constantly challenged with the task of organizing vast amounts of scholarly articles, research papers, and other digital doc-
documents. The proposed knowledge graph generation system can significantly enhance the management, retrieval, and discovery of information within these repositories. Here’s how:

- **Enhanced Information Retrieval:**
  
  - **Semantic Search:** Traditional keyword-based search often falls short in capturing the context and semantic relationships between terms. By leveraging knowledge graphs, researchers can perform semantic searches that understand the meaning behind their queries, leading to more accurate and relevant results.

  - **Example:** A researcher searching for “machine learning applications in healthcare” would retrieve not only documents containing those exact keywords but also papers discussing related topics like “AI in medical diagnostics” and “predictive analytics in healthcare.”

- **Improved Document Organization:**

  - **Clustering of Related Documents:** The system groups similar documents into clusters based on their semantic content. This helps in organizing the digital library in a more structured manner, making it easier for users to navigate through related research areas.

  - **Example:** All documents related to “neural networks” could be clustered together, providing researchers with a comprehensive view of all relevant work in that area.

- **Discovery of New Research Connections:**

  - **Knowledge Graph Exploration:** By visualizing documents as nodes in a knowledge graph with edges representing semantic relationships, re-
searchers can discover connections between different studies that they might not have noticed through traditional methods.

- **Example:** A knowledge graph might reveal a link between research on “genomic data analysis” and “machine learning algorithms,” suggesting potential interdisciplinary collaborations.

**• Tagging and Categorization:**

- **Automated Tagging:** The system can automatically generate and assign tags to documents based on their content, ensuring consistent and comprehensive metadata that enhances searchability and categorization.

- **Example:** A newly published paper on “deep reinforcement learning” could be tagged with “AI,” “machine learning,” “reinforcement learning,” and “deep learning,” improving its discoverability.

**• Academic Collaboration and Networking:**

- **Connecting Researchers:** The knowledge graph can highlight researchers who have published extensively in specific areas, facilitating networking and collaboration opportunities.

- **Example:** By analyzing the knowledge graph, a researcher interested in “bioinformatics” could identify other leading experts in the field and explore their work for potential collaboration.

**• Trend Analysis and Insights:**

- **Research Trends:** The system can help identify emerging trends and patterns in research by analyzing the evolving structure of the knowledge graph over time.
– **Example:** By examining changes in the graph, academic institutions can gain insights into growing research areas and allocate resources accordingly.

By implementing this knowledge graph generation system, digital libraries and academic institutions can provide a more efficient, intuitive, and insightful research environment, ultimately advancing the pace and quality of academic discoveries.

### 6.2 Discussion

In this section, we discuss the implications of our findings, potential improvements, and future directions for research.

#### 6.2.1 Improving Robustness and Generalization

Future research should focus on conducting multiple runs with different samples to obtain a more robust measure of performance. Expanding the study to include diverse datasets will help generalize our findings and validate the methodology across different contexts.

#### 6.2.2 Objective Tag Aggregation Methods

Developing an objective and transparent methodology for tag aggregation is crucial. This could involve automated clustering techniques or expert consensus to minimize bias and improve the accuracy of clustering results.
6.2.3 Enhancing Practical Relevance

Providing practical examples of clustering decisions and their implications can enhance the relevance of our findings. Future work should include detailed case studies that demonstrate the application of the system in real-world scenarios.

6.2.4 Addressing Subjectivity in Tagging

Exploring ways to mitigate the impact of subjective tagging on the evaluation metrics is essential. This could involve developing methods to standardize tags or using crowdsourced tagging to average out individual biases.

6.2.5 Technical Improvements and Optimization

Further technical improvements, such as optimizing the pipeline for larger datasets and exploring advanced clustering and edge assignment algorithms, can enhance the performance and scalability of the system.

6.2.6 More Data, More Models, More Methods

Expanding the dataset to include more diverse documents from different specific niches and different lengths (general notes, news, contracts, legal, etc.) can help us better evaluate the pipeline. Additionally, trying out a wider range of methods (e.g., hierarchical clustering) and testing more niche fine-tuned models could lead to discovering more effective combinations for embedding, clustering, and edge assignment.
6.2.7 Multimodal Models

Adding support for multimodal models that can handle text, images, and other data types would make the pipeline more versatile. Testing embedding models from sources like OpenAI, which can embed various data types, could lead to richer and more informative knowledge graphs.

6.2.8 Aggregation Methods for Sentence Embeddings

Experimenting with different ways to combine sentence embeddings into document embeddings could improve the quality of the final embeddings. Future work could explore techniques such as “mean pooling”, “sum pooling”, “max pooling”, “min pooling”, “self-attention”, “global attention”, “content-based attention”, and “learned self-attention” to see which method best captures the document’s meaning.

6.2.9 Degree of Separation

The degree of separation in a knowledge graph refers to the number of edges connecting two nodes. This tests how closely related two entities are within a graph. Understanding the interconnectedness of documents with fewer degrees indicating stronger or more immediate connections can help identify influential documents and optimize the organization of the knowledge graph for efficient information retrieval.

6.2.10 Statistical Significance Testing

Implementing statistical significance testing will help in validating the results of different configurations. Techniques such as t-tests, ANOVA, or permutation tests can
be used to determine whether the observed differences in clustering performance metrics (e.g., homogeneity, completeness, V-measure) between different configurations are statistically significant. This will ensure that the observed improvements are not due to random chance but are genuinely attributable to the chosen configurations.

By addressing these aspects, we aim to provide a comprehensive discussion that not only reflects on the current study but also paves the way for future advancements in the field.
Chapter 7

CONCLUSION

The development of this pipeline for generating meaningful semantically organized knowledge graphs from only intrinsic information found in natural language with contemporary open-source technologies represents another step in the field. By utilizing modern architectures, such as sentence embedding transformers provided by HuggingFace and Sci-Kit Learn for clustering, this work enables a detailed exploration of knowledge graph generation pipelines. The modular and flexible design with overarching methods and granular hyperparameters makes it accessible.

The initial experiments demonstrated the feasibility of using embedding models and clustering algorithms to create semantically rich knowledge graphs. By evaluating various combinations of models, algorithms, and methods, we were able to identify the strengths and weaknesses of different approaches.

These experiments show that the pipeline’s strengths lie in its ability to semantically differentiate and cluster documents effectively. The use of modern transformer models, such as all-MiniLM-L6-v2 and all-mpnet-base-v2, proved to be highly effective for larger datasets, while nomic-embed-text-v1.5 showed promise for smaller datasets. Clustering algorithms like GMM and K-means consistently provided high homogeneity and completeness scores indicating their suitability for creating coherent clusters.

The broader implications of this research extend beyond the specific datasets and methods used. The pipeline developed in this thesis represents a step forward in the semantic structuring of digital documents, offering a scalable and flexible solution for
knowledge graph generation. By providing a more structured and interconnected view of data, this approach can significantly improve information retrieval and knowledge discovery in various domains.

In conclusion, this work provides valuable insights into the process of generating knowledge graphs from textual data. It highlights the importance of selecting the right combination of embedding models, clustering algorithms, and evaluation metrics to achieve the best results. While there are challenges and limitations, the progress made in this thesis represents a meaningful advancement in the field, paving the way for future research and development in semantic knowledge representation.
BIBLIOGRAPHY


https://www.newhorizons.com/resources/blog/dbscan-vs-kmeans-a-guide-in-python#:~:text=Two%20popular%20clustering%20algorithms%20are,any%20cluster%20are%20considered%20noise.


[10] google-bert/bert-base-uncased · Hugging Face — huggingface.co.

https://www.learndatasci.com/glossary/hierarchical-clustering/
#:~:text=a%20real%20dataset-,What%20is%20Hierarchical%20Clustering%3F,
hierarchical%20tree%20called%20a%20dendrogram.


[17] nomic-ai/nomic-embed-text-v1.5 · Hugging Face — huggingface.co.


[23] scikit-learn: machine learning in Python &x2014; scikit-learn 1.5.0 documentation — scikit-learn.org.


[25] sentence-transformers/all-mpnet-base-v2 · Hugging Face — huggingface.co.
   https://huggingface.co/sentence-transformers/all-mpnet-base-v2.
   [Accessed 28-05-2024].


   https://www.techtarget.com/searchcio/definition/Semantic-Web#:~:text=95
The Semantic Web is a 3.0 is about linked data. [Accessed 28-05-2024].


