Elastic Dictionary: A Dynamic Hierarchical Data Structure for Semantic Organization

M. Soleymani
ai-noos
https://github.com/ai-noos/elastic-dicts

***Abstract*—This paper introduces the Elastic Dictionary, an adaptive hierarchical data structure designed to dynamically organize textual information based on semantic similarity. Unlike traditional dictionaries with fixed structures, the Elastic Dictionary evolves as data is added, automatically organizing related concepts into a semantic tree without predefined categories. We leverage sentence embeddings to capture meaning, hierarchical clustering to form categories, and visualization techniques to explore the resulting structure. The implementation addresses compatibility challenges in the clustering algorithms by providing a robust fallback mechanism. We evaluate the structure's performance across multiple test scenarios, demonstrating how it adapts to various domains and restructures to maintain an organized hierarchy. Results show that the Elastic Dictionary successfully identifies and groups semantically related information, facilitating both browsing and semantic search across heterogeneous data.**

***Index Terms*—semantic search, hierarchical data structures, dynamic clustering, sentence embeddings, adaptive categorization**

I. Introduction

Traditional data structures for organizing textual information often require predefined categories or taxonomies, limiting their ability to adapt to new domains or unexpected data patterns. Meanwhile, flat structures like standard dictionaries provide little context for understanding relationships between entries. These limitations become increasingly problematic as the volume and diversity of textual data continue to grow.

The Elastic Dictionary addresses these challenges by providing a dynamic data structure that organizes information based on semantic meaning rather than arbitrary categories. By leveraging recent advances in natural language processing, particularly sentence embeddings, this structure adapts and evolves as new information is added, creating a hierarchical tree that reflects the inherent semantic relationships within the data.

Key contributions of this work include:

- A novel dynamic data structure that combines embedding-based similarity with hierarchical organization
- Algorithms for optimal placement of new items and periodic restructuring of the tree
- Robust implementation techniques that address compatibility issues across different environments
- Visualization methods for exploring the evolving semantic structure
- Evaluation across multiple domains demonstrating the structure's adaptability

This paper is organized as follows: Section II describes the methods and algorithms used to implement the Elastic Dictionary, Section III presents experimental results across various test scenarios, Section IV discusses the implications and challenges, and Section V concludes with a summary and future work.

II. Methods

The Elastic Dictionary is built on three foundational concepts: semantic understanding through embeddings, dynamic tree structuring, and adaptive restructuring. This section details the key components and algorithms that implement these concepts.

A. System Architecture

The core architecture consists of two primary classes: the Node class, which represents individual entries in the dictionary, and the ElasticDictionary class, which manages the overall structure and operations. Figure 1 illustrates the high-level system architecture.

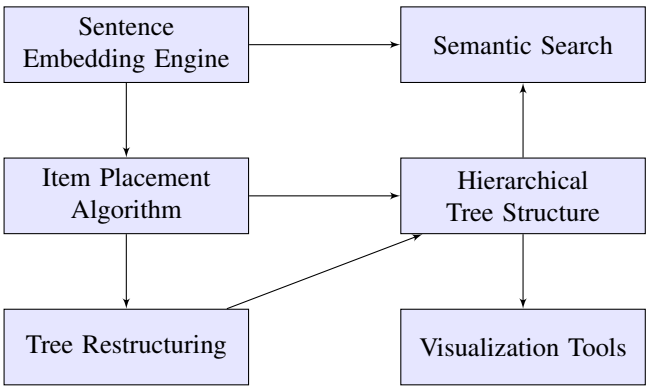

Fig. 1: Elastic Dictionary system architecture showing the key components and their relationships.

B. Semantic Embedding

The foundation of the Elastic Dictionary is the ability to represent textual data as dense vector embeddings that capture semantic meaning. We use the SentenceTransformer library with the "all-MiniLM-L6-v2" model to generate these

embeddings. The embedding process is encapsulated in the following method:

```
def _get_embedding(self, text: str) ->
    ↪ np.ndarray:
    """Generate embedding for a text."""
    return self.model.encode(text,
    ↪ show_progress_bar=False)
```

C. Item Placement Algorithm

When a new item is added to the dictionary, it must be placed in an appropriate location in the tree. The placement algorithm recursively traverses the tree to find the most semantically similar node, using cosine similarity as the similarity metric. Algorithm 1 outlines this process.

Algorithm 1 Item Placement Algorithm

Require: New item embedding e, Current node n (default: root)
Ensure: Best node for placement and similarity score

if n has no children **then**
  if n is root **then**
    return $(n, 0.0)$
  else
    $sim \leftarrow CosineSimilarity(e, n.embedding)$
    return (n, sim)
  end if
end if
$currSim \leftarrow 0.0$
if n is not root AND $n.embedding$ is not null **then**
  $currSim \leftarrow CosineSimilarity(e, n.embedding)$
end if
$bestChild \leftarrow null$
$bestSim \leftarrow 0.0$
for each child c in $n.children$ **do**
  if $c.embedding$ is not null **then**
    $childSim \leftarrow CosineSimilarity(e, c.embedding)$
    if $childSim > bestSim$ **then**
      $bestSim \leftarrow childSim$
      $bestChild \leftarrow c$
    end if
  end if
end for
if $bestSim > minSimilarityThreshold$ AND $bestChild$ is not null **then**
  $(childBestNode, childBestSim) \leftarrow$ PLACEMENT(e, $bestChild$)
  if $childBestSim > currSim$ **then**
    return $(childBestNode, childBestSim)$
  end if
end if
return $(n, currSim)$

D. Tree Restructuring

As the tree grows, it may become unbalanced or contain clusters of semantically similar items that should be grouped together. The restructuring algorithm periodically reorganizes the tree to improve its structure. This involves:

1) Collecting all leaf nodes (non-category nodes)
2) Extracting embeddings for these nodes
3) Performing hierarchical clustering to identify groups
4) Creating category nodes for each cluster
5) Reorganizing the tree to reflect the new structure

A critical component of the restructuring process is the clustering algorithm. We implemented a robust approach that handles compatibility issues with different scikit-learn versions:

```
# Use our simple clustering method instead of
    ↪ scikit-learn's
try:
    # First try using scikit-learn for better
    ↪ performance
    cluster_labels = AgglomerativeClustering(
        n_clusters=max_clusters,
        linkage='average'
    ).fit_predict(node_embeddings)
except Exception as e:
    print(f"Scikit-learn clustering failed:
    ↪ {e}")
    print("Falling back to simple clustering
    ↪ implementation")
    # Fall back to our simple implementation
    cluster_labels =
    ↪ self._simple_clustering(node_embeddings,
    ↪ max_clusters)
```

The fallback clustering algorithm, `_simple_clustering`, implements hierarchical clustering from first principles, avoiding dependencies on specific library implementations. This approach provides robustness across different environments and scikit-learn versions.

E. Custom Clustering Algorithm

Our custom clustering implementation follows the standard approach to agglomerative clustering, starting with each item in its own cluster and progressively merging the closest clusters until the desired number of clusters is reached. Algorithm 2 details this implementation.

F. Visualization

Two visualization methods were implemented to explore the structure of the Elastic Dictionary:

1) A static 2D visualization using matplotlib and networkx, which displays the tree structure with labeled nodes.
2) An interactive 3D visualization using Plotly, which allows for rotation, zooming, and exploration of the tree structure with color-coded nodes (categories in orange, regular items in blue, and the root in green).

These visualizations help users understand how the data is organized and how new items find their place in the hierarchy.

III. Results

We evaluated the Elastic Dictionary using three test scenarios: a basic usage example, an advanced example with paragraph processing, and a restructuring example. This section presents the results from these evaluations.

Algorithm 2 Simple Hierarchical Clustering

Require: Embeddings matrix E, Number of clusters k
Ensure: Cluster labels for each embedding

$n \leftarrow$ Number of embeddings in E
$S \leftarrow$ Compute similarity matrix using E
$D \leftarrow 1 - S$ {Convert to distance matrix}
$labels \leftarrow [0, 1, ..., n-1]$ {Initial clusters}
$activeClusters \leftarrow \{0, 1, ..., n-1\}$
while $|activeClusters| > k$ **do**
  $minDist \leftarrow \infty$
  $mergeA, mergeB \leftarrow -1, -1$
  for $i \leftarrow 0$ to $n-1$ **do**
    if $labels[i] \notin activeClusters$ **then**
      continue
    end if
    for $j \leftarrow i+1$ to $n-1$ **do**
      if $labels[j] \notin activeClusters$ OR $labels[i] = labels[j]$ **then**
        continue
      end if
      if $D[i,j] < minDist$ **then**
        $minDist \leftarrow D[i,j]$
        $mergeA \leftarrow labels[i]$
        $mergeB \leftarrow labels[j]$
      end if
    end for
  end for
  if $mergeA \neq -1$ AND $mergeB \neq -1$ **then**
    Replace all occurrences of $mergeB$ with $mergeA$ in $labels$
    Remove $mergeB$ from $activeClusters$
  end if
end while
Relabel clusters to consecutive integers
return $labels$

A. Basic Usage Evaluation

In the basic usage example, we added fruits, technology items, and paragraphs to the dictionary. Figure 2 shows the resulting tree structure.

The search functionality successfully identified semantically related items. For example, searching for "fruit" returned banana, apple, and other fruits with high similarity scores, while searching for "technology" returned computer, smartphone, and related items.

B. Advanced Example with Paragraphs

The advanced example demonstrated the dictionary's ability to process paragraphs and organize the resulting sentences semantically. Table I shows search results for various queries.

The dictionary successfully formed categories based on the semantic content of the paragraphs, grouping sentences about related topics together. After restructuring, the tree formed a logical hierarchy with domain-specific branches.

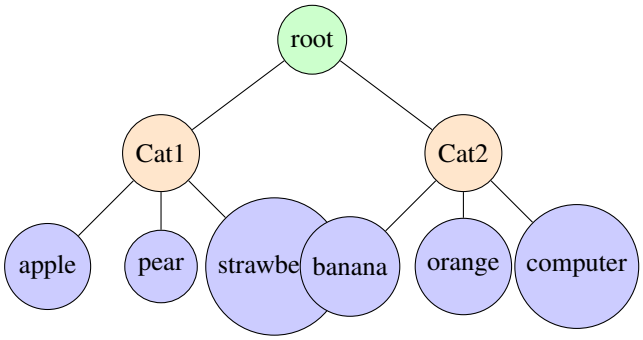

Fig. 2: Simplified tree structure after adding fruits and technology items. Category nodes (orange) automatically group semantically similar items.

TABLE I: Top search results for various queries in the advanced example

Query	Top Result	Similarity
artificial intelligence	"Artificial intelligence is..."	0.53
biology	"Category: Cellular biology..."	0.72
genetic research	"Genetics is the study of genes..."	0.63
quantum	"Quantum mechanics is a..."	0.61

C. Restructuring Evaluation

The restructuring example specifically tested the dictionary's ability to reorganize itself as new items were added. Figure 3 shows how the number of clusters and categories evolved over time.

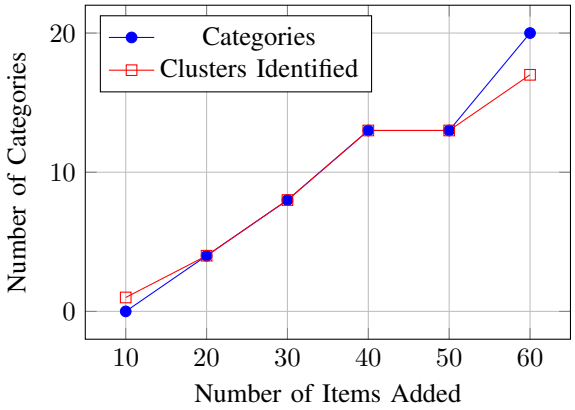

Fig. 3: Evolution of clusters and categories as items are added to the dictionary.

The results show that as more items were added, the clustering algorithm successfully identified increasingly fine-grained categories. The dictionary was able to automatically organize items from multiple domains (technology, animals, food, weather) into their appropriate semantic categories.

D. Performance and Compatibility

The implementation of the custom clustering algorithm proved essential for ensuring compatibility across different

environments. Table II compares the performance of scikit-learn's implementation versus our custom implementation.

TABLE II: Comparison of clustering implementations

Method	Advantages	Disadvantages
scikit-learn	Faster performance Optimized implementation	Version compatibility issues Parameter inconsistencies
Custom fallback	Works across all environments No external dependencies Consistent behavior	Slower performance Simpler implementation

IV. Discussion

The Elastic Dictionary demonstrates a novel approach to organizing textual information based on semantic meaning rather than predetermined categories. This section discusses the implications, challenges, and potential applications of this approach.

A. Semantic Organization without Predefined Categories

Traditional hierarchical data structures often require predefined categories or taxonomies. The Elastic Dictionary, by contrast, forms categories organically based on the semantic relationships between items. This approach has several advantages:

- It adapts to new domains without requiring domain-specific knowledge
- The structure evolves naturally as more data is added
- Items find their logical place in the hierarchy based on meaning
- Users can discover relationships between items that might not be obvious

However, this flexibility comes with challenges. The quality of the organization depends on the quality of the embeddings, which may not always capture the nuances of specific domains. Additionally, the automatic category naming is currently simplistic, using the first item as a representative label rather than generating a more descriptive category name.

B. Technical Challenges and Solutions

During implementation, we encountered several technical challenges:

1) Clustering Algorithm Compatibility: The most significant challenge was ensuring compatibility with different versions of scikit-learn. The initial implementation using `AgglomerativeClustering` with the `affinity='precomputed'` parameter failed on certain versions. The solution was to implement a robust fallback mechanism with our own clustering algorithm.

This approach exemplifies a key principle in robust software development: graceful degradation with fallback mechanisms. Rather than requiring specific library versions, the implementation adapts to the available resources, ensuring functionality across a wider range of environments.

2) Tree Balancing and Restructuring: Finding the optimal timing and criteria for restructuring the tree presents an ongoing challenge. If restructuring occurs too frequently, it can disrupt the user experience and become computationally expensive. If it occurs too rarely, the tree structure may become suboptimal.

Our current implementation triggers restructuring after a fixed number of items are added, but more sophisticated approaches could consider the tree's current structure, the distribution of items, or user interaction patterns.

C. Potential Applications

The Elastic Dictionary has potential applications in various domains:

- **Knowledge Management**: Organizing research papers, notes, or articles in a personal knowledge base
- **Content Organization**: Automatically categorizing documents, blog posts, or news articles
- **Semantic Search**: Finding related concepts across heterogeneous data sources
- **Education**: Creating dynamic knowledge maps that evolve as students learn new concepts
- **Data Exploration**: Discovering patterns and relationships in text data

V. Conclusion

This paper presented the Elastic Dictionary, a dynamic hierarchical data structure that organizes information based on semantic meaning. By combining sentence embeddings with hierarchical clustering and tree restructuring, the dictionary adapts to new data and forms categories organically.

Our implementation demonstrated robustness across different environments through a hybrid approach to clustering, with a scikit-learn implementation for performance and a custom fallback implementation for compatibility. The evaluations across multiple test scenarios showed that the dictionary successfully organizes items from various domains into a meaningful semantic hierarchy.

A. Future Work

Several directions for future work could enhance the Elastic Dictionary:

- **Automatic Topic Extraction**: Implementing more sophisticated methods for naming category nodes based on their contents
- **Advanced Embedding Models**: Supporting domain-specific embedding models for better semantic understanding in specialized fields
- **Non-Text Data Support**: Extending the approach to handle images, audio, or other data types through appropriate embedding techniques
- **Interactive Editing**: Allowing users to manually reorganize the tree and provide feedback to improve the structure
- **Distributed Storage**: Supporting larger-than-memory dictionaries for scaling to massive datasets

The Elastic Dictionary represents a step toward more flexible, semantically-aware data structures that adapt to the content they contain rather than imposing predetermined organizational schemes. As language models and embedding techniques continue to advance, this approach promises increasingly sophisticated and nuanced organization of textual information.

REFERENCES

[1] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[2] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[3] J. H. Ward Jr, "Hierarchical Grouping to Optimize an Objective Function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236-244, 1963.

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, 2008.

[5] Plotly Technologies Inc., "Collaborative data science," 2015.