JSON Clustering based on Structural, Semantic and Contextual Similarity

Ishaan Singh, Sudarshan Sundarrajan, Ikjot Singh, Ashutosh Anand

Roll No: 191CS124, 191CS225, 191CS123, 191CS111

Under the guidance of

Dr. P. Santhi Thilagam

Department of Computer Science and Engineering

National Institute of Technology Karnataka

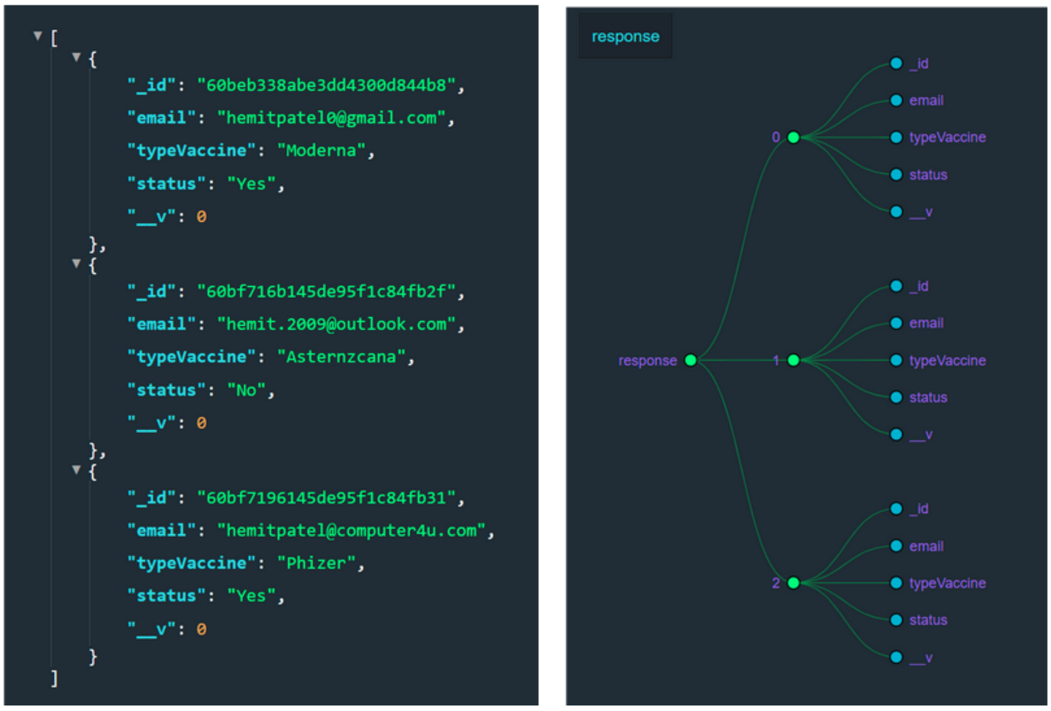
P.O. Srinivasnagar, Surathkal, Mangaluru-575025, India

April 2022

Contents

1 Introduction

JavaScript Object Notation (JSON) [1] is a widely adopted format for representing structured data in text based format. The notation is derived from the popular JavaScript programming language but its convention can be understood and parsed in several other programming languages. Currently, data is stored and exchanged using the JSON notation. A JSON is commonly structured as an order of key/value pairs. The key is usually a defined string while a value can be of type strings, numbers, booleans, objects or arrays (An ordered list of values). A JSON structure can be represented as a tree.



```
▼ [
    ▼ {
          "_id": "60beb338abe3dd4300d844b8",
          "email": "hemitpatel0@gmail.com",
          "typeVaccine": "Moderna",
          "status": "Yes",
          "__v": 0
      },
    ▼ {
          "_id": "60bf716b145de95f1c84fb2f",
          "email": "hemit.2009@outlook.com",
          "typeVaccine": "Asternzcana",
          "status": "No",
          "__v": 0
      },
    ▼ {
          "_id": "60bf7196145de95f1c84fb31",
          "email": "hemitpatel@computer4u.com",
          "typeVaccine": "Phizer",
          "status": "Yes",
          "__v": 0
      }
  ]
```

Figure 1: JSON tree representation

As mentioned earlier, JSON notations are popularly being used for storing structured data and is used in NoSQL or nonrelational databases. Databases that make use of JSON representation to store, index and query data are called JSON document databases. Due to developments in distributed database systems, for the efficient storage, querying and insertions of documents in these systems, organizations of the documents plays a vital role. JSON clustering improves the effectiveness of document management systems by organizing the documents from a large collection into groups that can be useful for similarity-based retrieval.

2 Motivation

A JSON document database is arguably the most popular category in the NoSQL family of databases. NoSQL database management differs from traditional relational databases that struggle to store data outside of columns and rows. Instead, they flexibly adapt to a wide variety of data types, changing application requirements and data models. In an era where physical storage limits are no longer a bottleneck, JSON databases deliver superior scale and performance.

JSON document classes are able to do this because of forming JSON clusters or shards. This concept helps reduce overhead over a centralized approach, while increasing query performance. Figure 2 shows how MongoDB [2] processes queries using sharding techniques.

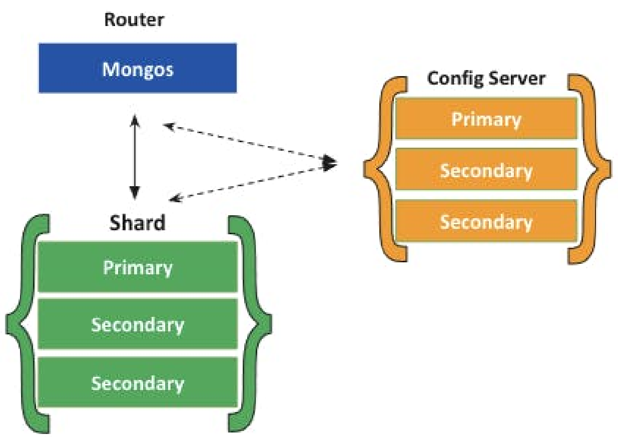


Figure 2: Sharding in MongoDB with Router and Config Server

3 Literature Survey

We can start by looking over a few clustering techniques used for XML documents. Asghari etl. al [3] reviews different existing XML clustering methods with the use of different similarity scores. Some of the similarity scores it uses to compare are Tree-based similarity measures, Vector based similarity measures and Time series based similarity measures. From the comparison table it was seen that the vector and tree based similarity measures were widely adopted for XML clustering. Further, clustering of the XML documents based on the similarity scores were also surveyed. These included Hierarchical clustering, Partitioning clustering, Combination clustering and Multilevel clustering approaches.

Since similarity score plays a big role in clustering documents, we can look over a few more similarity measures in detail. Salman et. al [7] surveys the use of different similarity measurers such as:

- Data Type Similarity for finding the similarity of the document based on different types.
- Cardinality Constraint Similarity to find the minimum and maximum number of times an element occurs in the documents.

- Element Context Similarity Matching which matches based on the similarity of their schema structure. This similarity measuring approach focus on the structural and geometrical characteristics of the trees and does not consider the conceptual semantics of the tree nodes
- XML Schema Similarity Measurement is a proposed measurement which is a combination of schema matching techniques in order to produce best matching results.

It can be seen that most document clustering approaches perform the following steps

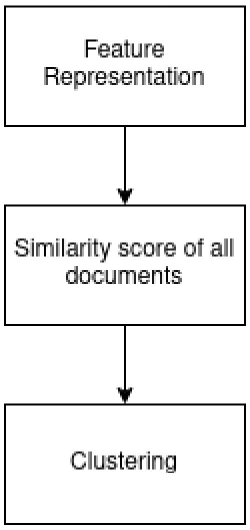


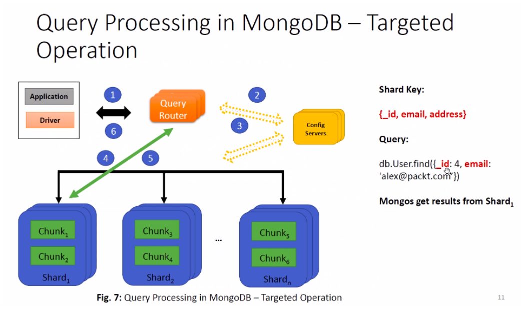
Figure 3: Flow of clustering documents

From the literature survey we observe that:

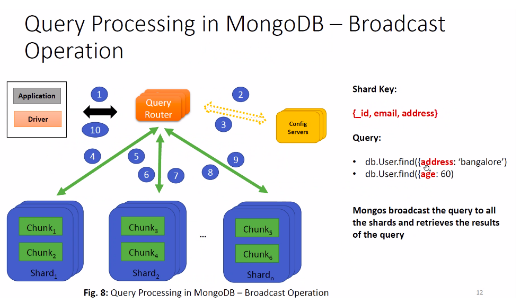
- Widely used feature representation of documents are trees, graphs, vectors etc..
- The represented documents are now used for calculating similarity scores between all documents.
- Based on the output after measuring similarity scores we would be able to perform a clustering algorithm.

4 Problem Statement and Objectives

In distributed data systems, there are two types of major query operations that can be performed based on the availability of the requested data on a given shard. These two operations are - targeted operations and broadcast operations. If we take the example of the MongoDB system, there exists an application/driver that will be making the requests, which will be passed to the MongoDB ecosystem. The first point of contact of this request will be a query router, which is connected to a Config server. Now, based on the query, and the shard key that is available,


Query Processing in MongoDB – Targeted Operation
Application
Driver
1
6
Query Router
2
3
Config Servers
4
5
Chunk$_1$
Chunk$_2$
Shard$_1$
Chunk$_3$
Chunk$_4$
Shard$_2$
...
Chunk$_5$
Chunk$_6$
Shard$_n$
Shard Key:
{_id, email, address}
Query:
db.User.find({_id: 4, email: 'alex@packt.com'})
Mongos get results from Shard$_1$
Fig. 7: Query Processing in MongoDB – Targeted Operation
11

(a) Targeted Operations


Query Processing in MongoDB – Broadcast Operation
Application
Driver
1
10
Query Router
2
3
Config Servers
4
5
6
7
8
9
Chunk$_1$
Chunk$_2$
Shard$_1$
Chunk$_3$
Chunk$_4$
Shard$_2$
...
Chunk$_5$
Chunk$_6$
Shard$_n$
Shard Key:
{_id, email, address}
Query:
db.User.find({address: 'bangalore')
db.User.find({age: 60)
Mongos broadcast the query to all the shards and retrieves the results of the query
Fig. 8: Query Processing in MongoDB – Broadcast Operation
12

(b) Broadcast Operations

Figure 4: Different operations in Distributed Databases

the query will be either routed to a particular targeted shard, or it will be broadcasted to all shards. In case the query contains the shard key of a shard, the operation will be targeted to that particular shard. Otherwise, the query will be routed to all shards, and the collected results are sent back to the application.

As is obvious, targeted operations have a much lesser overhead in comparison to the broadcast operation. So, our work in this project is mainly targeted towards increasing the number of targeted operations through grouping/clustering similar documents together in a single shard. The main objectives are enumerated below:

1. Research and identify the best representation of JSON documents to calculate similarity scores.
2. Find the best ways to calculate similarity scores between JSON documents.
3. Experiment different clustering techniques and compare them on relevant performance metrics.

5 Proposed Methodology

In the proposed solution, we have two main steps. First, we do a similarity matching of different documents on various measures, and then cluster them based on the similarity scores we received from the first step. Both these steps are explained below:

5.1 Similarity measurements

As seen in the above literature survey, there are different types of similarity measurements that can be done, such as structural similarity, contextual similarity, semantic similarity, etc. So for the purpose of our problem, we used a combination of all these three similarity scores.

5.1.1 Structural Similarity

This is perhaps the most important similarity measure of the three because, it will be a good score only if the majority part of the Node-to-Leaf (NTL) structure will be the same. For calculating the structural similarity for two documents, we first calculate the set of Node-to-Leaf paths for each document. On these constructed sets, we perform a similarity algorithm, in our case Jaccard similarity. The result from this Jaccard similarity step will be our final result from this step.

1. **Why have a Structural Similarity Score?**

   Structural Similarity is a measure of how distant two documents are, with respect to their schema. If documents are created by the same application, and are to be queried in a similar fashion, they will intrinsically have the same schema. Hence, the structural similarity score of a pair of documents is a direct indication of why and how the documents will be queried for the data they represent.

2. **Range, Meaning of the Similarity Score Value**

   The range of the score is 0-1, 0 being the lowest score for documents that are not similar structurally, and 1 being the highest score, indicating that the 2 documents are exactly the same in terms of their schema.

5.1.2 Contextual Similarity

In this section, we first take the Root-to-Leaf (RTL) paths' set from each of the two documents that we are trying to compare. This set of RTL path strings is passed onto the pre-trained BERT model [6] in order to get the sentence embeddings for the same. Once that is complete, we compare the embeddings by using some similarity algorithm, in our case, we used cosine similarity for the same. The result from the cosine similarity algorithm is what was returned as a result from this step. This particular similarity model allowed us to obtain some similarity based on the dependence of different parts of the sentence on each other.

1. **Why use BERT instead of other transformer-based models?**

   While implementation, we compared the performance of three different models, namely BERT, RoBERTa [5] and ELECTRA [4]. RoBERTa being a larger model, took more time to generate the sentence embeddings and once generated, it gave similar results to that seen with BERT embeddings. In the case of ELECTRA, the time taken to generate the embeddings was the fastest among the three, however, the similarity scores between different documents (in our case, an IMDB and DBLP document) were higher and undesirable. Hence, BERT was the final model of choice.

5.1.3 Semantic Similarity

For the final similarity phase, we employed the use of the WordNet thesaurus. In this, basic NLP was used to extract semantic relations between sentences. A feature set out of it was used to train with a classifier to boost for accuracy. For this, the same sentences that were used to get the BERT embeddings were used to be passed on to the WordNet lexical database based model. Subsequently, the Michael Lesk algorithm was used to get the best word sense of each word in sentence. After this, two similarity scores, Wu-Palmer similarity and path similarity were used to obtain a combined similarity scores for these.

5.2 Combining the different scores

After the completion of all the three similarity phases, the scores are combined into a single score. This will be in the ratio of 0.5 - 0.25 - 0.25 for Structural Similarity - Contextual Similarity - Contextual Similarity respectively. Based on this proportion, we calculate a weighted sum to be a representative of the final similarity score. The main reason for the high weightage given to the structural similarity score is the fact that if the structural similarity score is a high value, then the two documents are definitely going to be clustered together, and hence their overall score needs to be high as well. Since, this may not be true for the other two measures, we have considered lower weightages for them.

Further, it is to be noted that the thresholds chosen by the authors (0.5 - 0.25 - 0.25) for the three scores is a simple heuristic. Depending on the application and the user of the database, the types of queries being sent for processing will change. These queries could be solely based on the schema if the application that generated the data is trying to access it. This is where structural similarity will play a major role to make the processing faster. Alternatively, the query might be generated with the aim to combine data generated by different applications, on the same topic. This is where semantic similarity will play a major role to handle documents whose key-naming is not consistent. Finally, the queries might be string searches, or deal with the actual data stored in the document. This is where contextual similarity will play a major role.

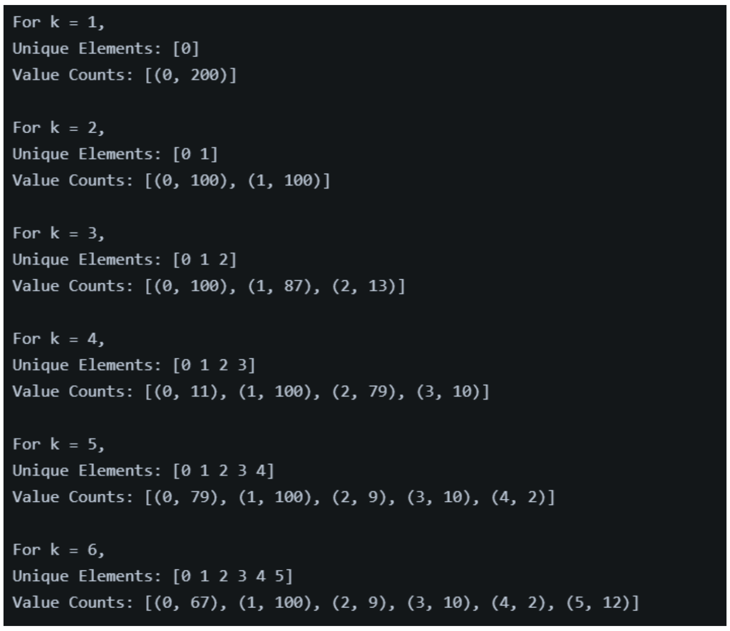
As an extension, many more thresholds can be chosen for finding a combined similarity score for a pair of documents. One may consider only structural similarity for the clustering operation, while a few others might like to include contextual similarity, but not semantic differences, if all data is generated from the same source. Due to this dynamic, setting the thresholds in advance, by any author is a wrong step to take. For testing, in this paper, (0.5 - 0.25 - 0.25) thresholds are used for the 3 similarity scores (Structural - Semantic - Contextual).

5.3 Clustering

As seen in the above literature survey, there are different types of similarity measurements that can be done, such as structural similarity, contextual similarity, semantic similarity, etc. So for the purpose of our problem, we used a combination of all these three similarity scores. After obtaining the score, we performed a K-Means clustering algorithm on it, to obtain the clusters of related documents. We have not used DBSCAN and Hierarchical Agglomerative Clustering because the k-distance graph used to obtain optimal *eps* and *minPts* did not give us a point of inflexion for DBSCAN, and the dendrogram generated was not satisfactory to perform hierarchical clustering. Hence, we went ahead with the K-Means algorithm for clustering. The results obtained from the same are elaborated on below.

6 Results

The proposed methodology was implemented on a combination of two datasets, one was the IMDb movies dataset (nearly 14000 JSON documents), and one was the synthetic dataset consisting of paper and author JSON documents (nearly 1L JSON documents). As mentioned in the proposed methodology, we obtained the pairwise similarity scores for selected documents, and combined them into one single similarity score. These scores were passed into the clustering algorithm, which then finally returned a clustering result. The generated combined similarity matrix is shown in Figure 10. The KMeans clustering output for different values of K is shown in Figure 5. The dendogram output for the agglomerative clustering algorithm is shown in Figure 6.



```
For k = 1,
Unique Elements: [0]
Value Counts: [(0, 200)]

For k = 2,
Unique Elements: [0 1]
Value Counts: [(0, 100), (1, 100)]

For k = 3,
Unique Elements: [0 1 2]
Value Counts: [(0, 100), (1, 87), (2, 13)]

For k = 4,
Unique Elements: [0 1 2 3]
Value Counts: [(0, 11), (1, 100), (2, 79), (3, 10)]

For k = 5,
Unique Elements: [0 1 2 3 4]
Value Counts: [(0, 79), (1, 100), (2, 9), (3, 10), (4, 2)]

For k = 6,
Unique Elements: [0 1 2 3 4 5]
Value Counts: [(0, 67), (1, 100), (2, 9), (3, 10), (4, 2), (5, 12)]
```

Figure 5: Clusters with different values of K

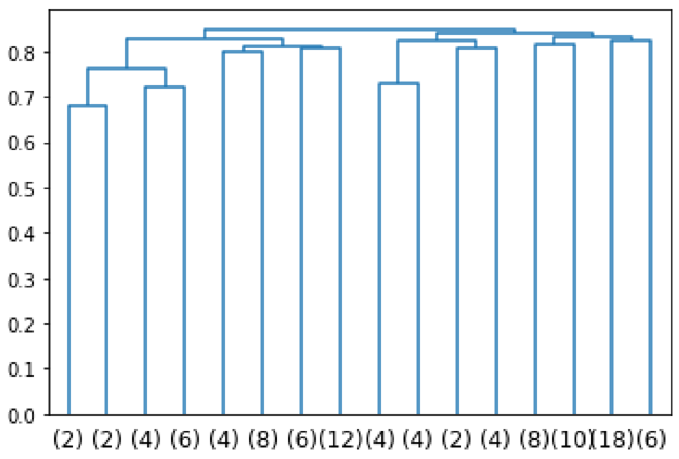


Figure 6: Generated dendogram

The K-Means algorithm was optimized by using the elbow method on the inertia in the clusters. From the following graph (Figure 7), the optimal K-value chosen was 2.

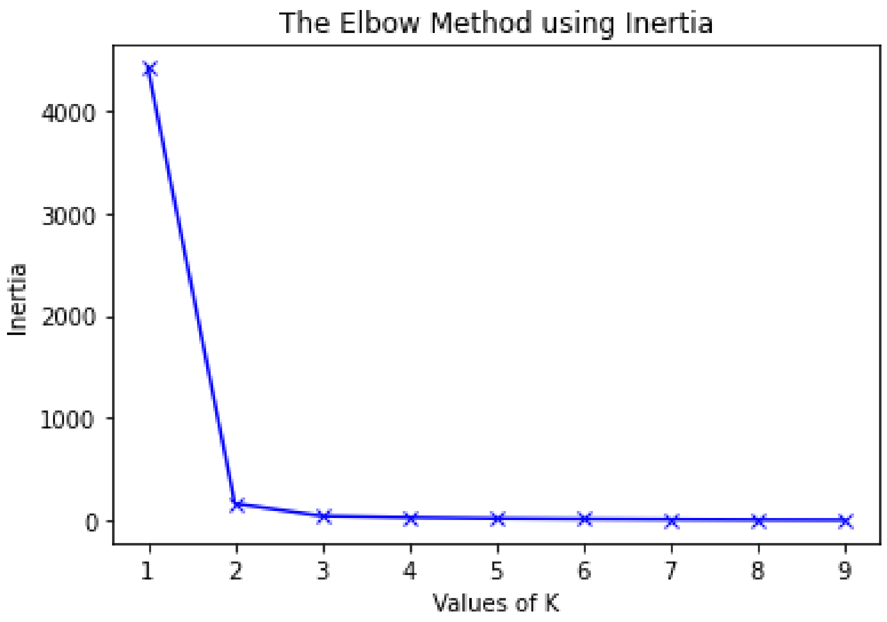
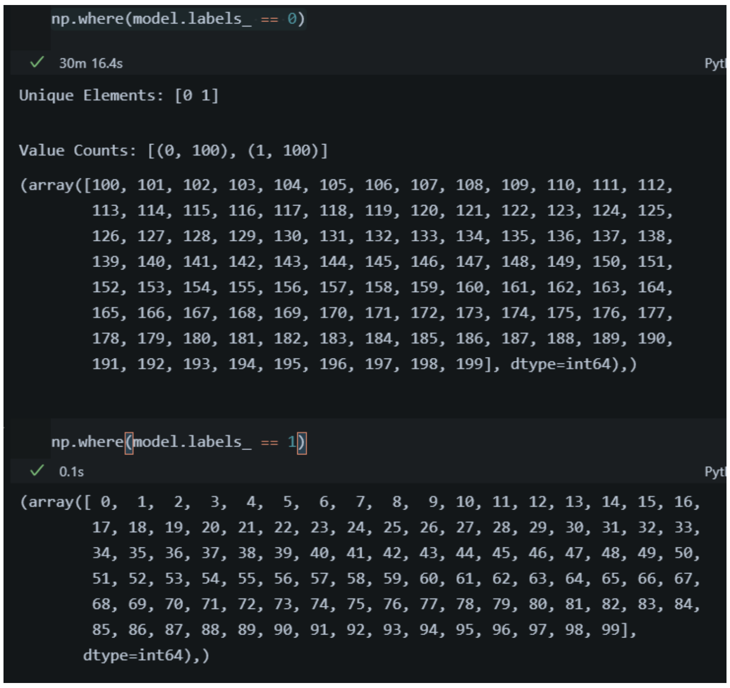


Figure 7: Elbow Curve



```
np.where(model.labels_ == 0)
```

✓ 30m 16.4s   Pyt

```
Unique Elements: [0 1]

Value Counts: [(0, 100), (1, 100)]

(array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
        113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
        126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138,
        139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
        152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
        165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
        178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
        191, 192, 193, 194, 195, 196, 197, 198, 199], dtype=int64),)
```

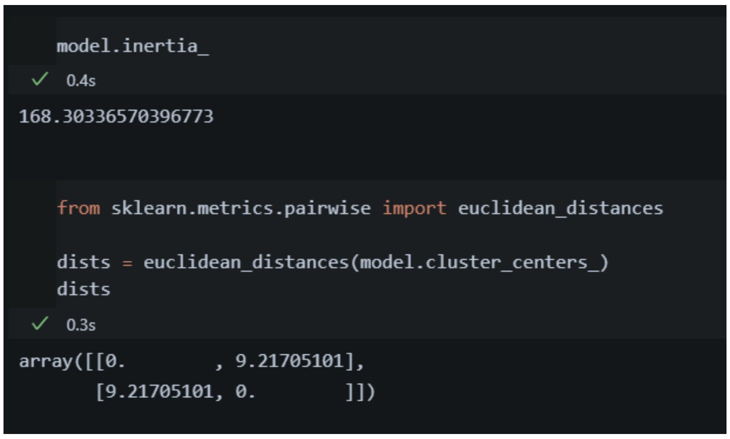
```
np.where(model.labels_ == 1)
```

✓ 0.1s   Pyt

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
       dtype=int64),)
```

Figure 8: Cluster formed by KMeans with K = 2

The clusters formed are not overlapping with each other. Given that the documents were taken from two different sources, and each set of documents was structurally, semantically and contextually different from the other set, splitting them into two different clusters is ideal with respect to the dataset. Hence, a 2-cluster K-Means algorithm is giving us the desired result.



```
model.inertia_
```

✓ 0.4s

```
168.30336570396773
```

```
from sklearn.metrics.pairwise import euclidean_distances

dists = euclidean_distances(model.cluster_centers_)
dists
```

✓ 0.3s

```
array([[0.        , 9.21705101],
       [9.21705101, 0.        ]])
```

Figure 9: Inertia and Cluster distances

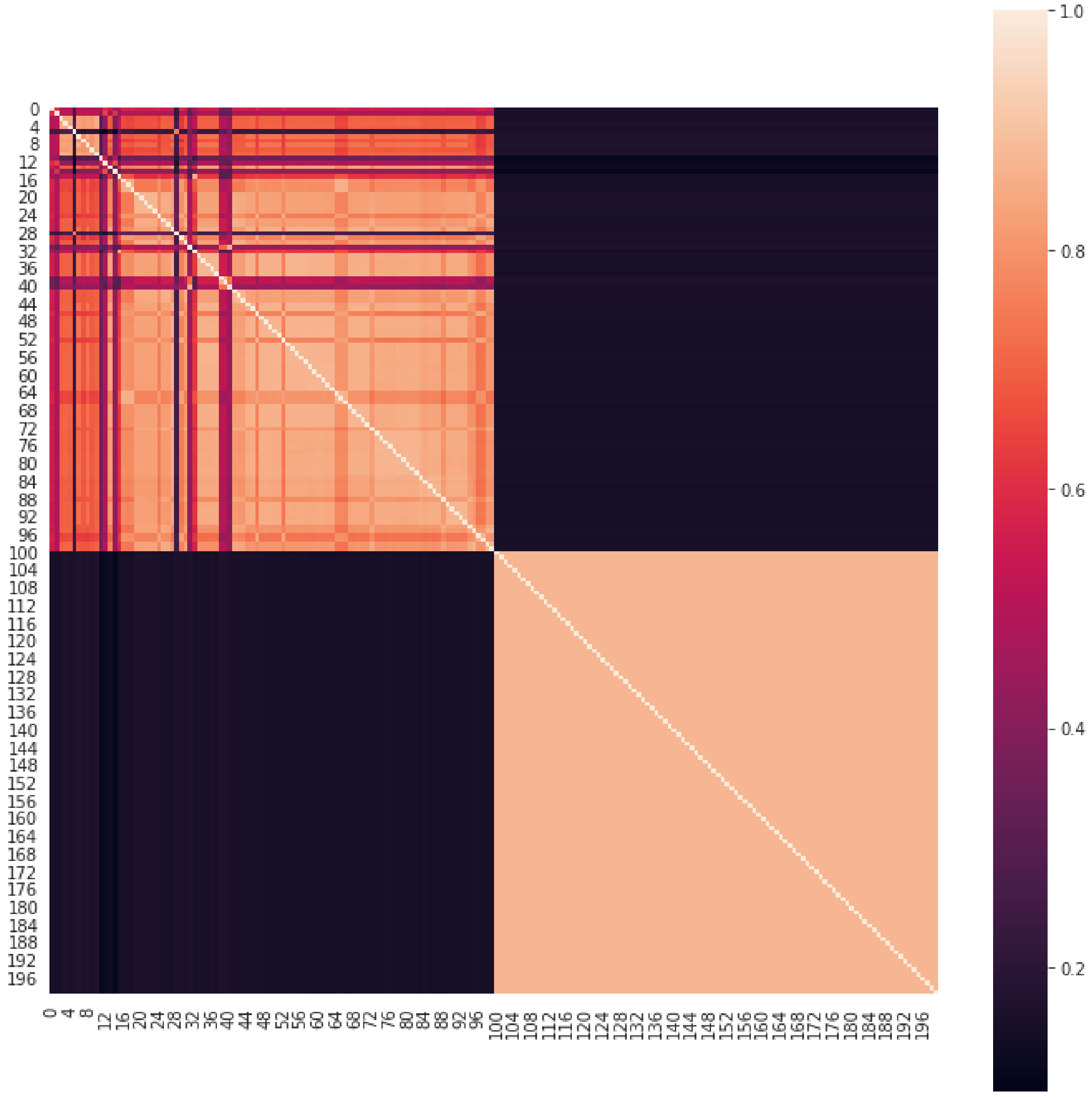


Figure 10: Generated Heatmap

References

[1] Mdn web docs - json. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON`.

[2] Mongodb. `https://www.mongodb.com/`.

[3] Elaheh Asghari and MohammadReza KeyvanPour. Xml document clustering: techniques and challenges. *Artificial Intelligence Review*, 43(3):417–436, 2015.

[4] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. 2020.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.

[6] Usman Naseem, Imran Razzak, Shah Khalid Khan, and Mukesh Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 20(5), jun 2021.

[7] Ayse Salman. Similarity matching of xml schema. *Contracampo*, 39(9):417–436, 121-129.