INCREMENTAL TEXT CLUSTERING ALGORITHM FOR CLOUD-BASED DATA MANAGEMENT IN SCIENTIFIC RESEARCH PAPERS

Mahfuja Nilufar

Distributed Systems and Multimedia Processing (DSMP) Lab
Department of Computer Science
Ryerson University
350 Victoria St
Toronto, ON, CANADA
mahfuja.nilufar@ryerson.ca

Abdolreza Abhari

Distributed Systems and Multimedia Processing (DSMP) Lab
Department of Computer Science
Ryerson University
350 Victoria St
Toronto, ON, CANADA
aabhari@ryerson.ca

ABSTRACT

This study aims to build clusters of similar research papers. Text clustering for research articles is challenging because re-clustering is necessary to handle newly added papers. An incremental clustering algorithm is presented to find similar research papers for COVID-19 related literature. The proposed approach uses an incremental word embedding generation technique to extract feature vectors of the papers. The initial clustering is done by using the K-means algorithm by two NLP feature extraction models; TF-IDF and Word2vec. The clustering results show that the Word2vec outperforms the TF-IDF model. With increasing COVID-19 literature continuously, the ultimate focus is to add the newly published papers to the existing clusters without re-clustering. Title, abstract, and full body of papers are considered for testing the proposed incremental algorithm. Clustering quality is evaluated by the Microsoft language similarity package, which shows clustering of the full-text body outperforms the abstract and title of papers.

Keywords: K-means, Incremental cluster, Incremental model, Word2vec, TF-IDF.

1 INTRODUCTION

During the COVID-19 pandemic, many scientific research papers were published, which helped increase the speed of the investigation and control the spread of the infections. Maintaining track of and finding similar literature among the overwhelming volume of publishing is challenging. Literature reviews play an important role in each author's research, as they help identify the scholarly knowledge that exists within the field. In order to find similar content, text clustering can aggregate literature texts with the same meaning. As the dataset is dynamic, the traditional clustering algorithm will not be able to assign new data to the existing clustering result without doing re-clustering from scratch. As opposed to re-clustering the entire set of data from scratch, incrementally updating the new data points based on the previous clustering method will be more efficient. Dynamic data clustering has several shortcomings, which inspired us to design our own incremental algorithm. Computers can only read raw text data if they have been converted into numerical values. To convert text data to numerical data, we need to vectorize the text data; in NLP (Natural Language Processing), it is called word embeddings. As the dataset is dynamic, we can't get word embedding vectors from all data at once. To obtain the word embedding vectors, our approaches will train the model with newly added data sets in an online manner. To train the model incrementally, a modified Word2vec model based on incremental Huffman tree merging technique is proposed in this paper. In this approach, when a new dataset is added to the system, rather than training the model from scratch, it will only train the existing model with the new dataset. In essence, incremental training is intended to save the processing time. Once the word embedding vector has been obtained from the incremental model training, it is passed to incremental clustering algorithm for grouping similar meanings of papers together. The incremental clustering algorithm searches for the closest centroid from each point when a new data point arrives. If a new data point is considered as a part of the closest group and the average similarity score exceeds the predefined threshold value, the new data point will be added into that closest group and labeled accordingly. Otherwise, it will be considered as a new cluster. In our previous work, we implemented the traditional K-means clustering algorithm for initial dataset. A database prototype system was developed by considering abstracts, titles, and the full body of papers (Abhari and Nilufar 2020). In our study, we found that a higher similarity score is obtained when we process more words. Kazemi and Abhari (2017) proposed a content-based recommendation system for scientific literature. The study compares the similarity score for each recommended paper based on Microsoft Language Similarity API. For the feature extraction, TF-IDF (Term Frequency - Inverse Dense Frequency) and Word2vec models have been used. The result showed that Word2vec generates 22% higher accuracy than the TF-IDF method. Peng et al. (2017) proposed an incremental word embedding model with a hierarchical SoftMax function. In their study, they didn't consider incremental clustering. Radu, Radulescu, and Mariana (2020) used the word embedding technique – Doc2vec model to cluster the document. Their study also used the TF-IDF model to compare the accuracy with the Doc2vec model. According to their results, K-means algorithm outperforms with the Doc2vec model, whereas TF-IDF accuracy is comparatively very low. They applied word embedding techniques to document clustering but did not consider the incremental clustering for dynamic datasets. For feature extraction, the common methods TF-IDF and Word2vec are very popular. TF-IDF method used Bag of Words (BoW) to calculate frequency of words. One of the major drawbacks of TF-IDF is that it ignores the semantic relationship among words (Kazemi and Abhari 2017). On the other hand, Word2vec model generates word embedding using a two layer neural network (Mikolov et al. 2013). The main important part is Word2vec performs semantic analysis. In our approach, we combined incremental model learning based on the incremental Huffman tree merging technique with the incremental clustering algorithm to cluster papers with similar meaning together.

2 METHODOLOGY

2.1 Data Extraction and Preprocessing

The White House and a coalition of leading research groups have prepared the COVID-19 Open Research Dataset (Wang et al. 2020). For the purposes of this study, only COVID-19 related research papers were selected. To filter the research papers, COVID-19 related keywords "COVID-19", and "SARS-CoV-2" is used. For this filtering process, the title and the abstract were excluded from the keyword search process; only the full body of text was considered. The dataset is preprocessed in order to check for missing values, noisy data, and other inconsistencies before the algorithm is applied.

2.2 Feature Extraction using TF-IDF and Word2vec

Feature extraction is very important in machine learning. We used TF-IDF and Word2vec model for feature extraction. TF-IDF model is used for initial dataset clustering to extract feature vector from papers. However, this model is not used in our incremental approach, because when new data is added into the system, the vocabulary size will increase, and thereby the vector size will increase too. For the incremental purpose, it's required the similar vector size for newly added data. We chose the CBOW (Continuous Bag of Words) Word2vec model for initial and incremental approach. In Word2vec, the vector size remains unchanged when new data is added. All the drawbacks of TF-IDF lead us to choose the Word2vec model for the incremental model training.

2.3 Incremental Model Training

Word2vec does not support incremental learning by default for the newly added dataset. To reduce computing costs, Word2vec (Mikolov et al. 2013) uses a hierarchical SoftMax technique. In the hierarchical SoftMax approach, the output layer is replaced by SoftMax layer, where a Huffman binary tree is presented for all the words in the leaf nodes. The architecture of hierarchical SoftMax, and more specifically the Huffman binary tree is redesigned for incremental learning in this study.

2.3.1 Incremental Huffman Tree

This study proposed an incremental Huffman tree merging technique during incremental model training. Whenever new data is added, instead of creating the whole Huffman tree from the scratch, the incremental Huffman tree merging technique will apply. Generally, in hierarchical SoftMax method when new words are added to the vocabulary list, the Huffman tree is re-created by combining with old and new words, which is computationally expensive and time consuming to perform these repetitive tasks. As shown by Figure 1, the initial Huffman tree is created with four words (W1, W2, W3, W4), where W1 is the most frequent word in the list.

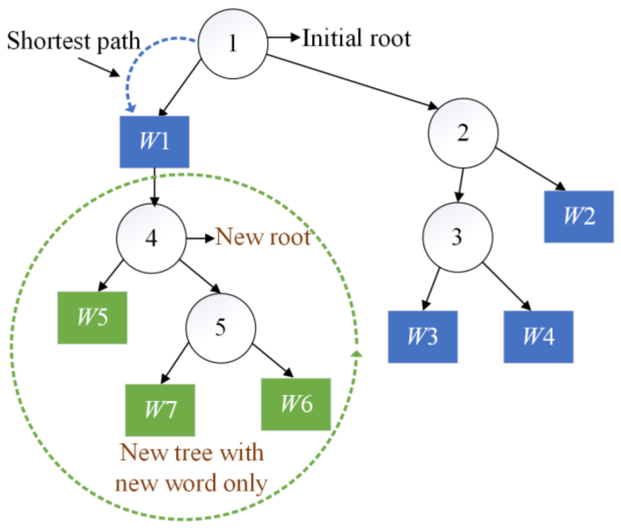


Figure 1: Incremental Huffman tree.

Figure 1 contains four leaf nodes, and all the internal nodes (1,2,3) from Huffman tree are connected to the hidden layer. When new data is added, the Huffman tree will be created with all the new words only. If any new words exist in the previous vocabulary list, those new words wouldn't be considered in the Huffman tree creation. However, during training the model, all words from the new dataset will be considered for learning. By this way the processing time is saved by not creating the tree from the scratch. In the next batch (Figure 1), total three words (W5, W6, and W7) presented, where W5 is the most frequent word among these three words. The shortest path is found from the initial training by traversing the tree from the root node to leaf node. In Figure 1, the path from the root node (node 1) to W1 is the shortest path and the leaf node (W1) in the shortest path will be the next root node. The new tree is created from the root node and all other words are assigned left or right based on the frequency (Huffman 2006). Table 1 shows the Huffman code, word frequency and internal node. For the generation of the Huffman code, we followed the rule that the code would be 0 if the node is on the left side and 1 otherwise. From Figure 1, the Huffman code for word W2 is '11'. From the initial training, the shortest path code is '0' and this code will be combined with the new code for a new word. For example, the Huffman code for the new word W7 is '10'. In order to obtain the final Huffman code for W7, this new code will be combined with the shortest path code, which is '010'. To be more precise, the shortest path code will be included as a prefix to the new Huffman code for all new words. The Huffman code that we have from the current training phase remains unchanged in future training phases. The input vector for each word needs to be initialized during the learning phase. In the incremental learning process, the initial input weight vector for old words is retained from previous learning sessions, and for new words the vectors are randomly initialized. In each training session, the Huffman code, number of internal nodes, and shortest path information will be saved for the next session.

Table 1: Incremental Huffman code.

Words	**Frequency**	**Huffman Code**	**Internal Node**
*W*1 (Initial training phase)	10	0	1
*W*2 (Initial training phase)	7	11	12
*W*3 (Initial training phase)	3	100	123
*W*4 (Initial training phase)	3	101	123
*W*5 (Incremental training phase)	5	00	14
*W*6 (Incremental training phase)	3	011	145
*W*7 (Incremental training phase)	2	010	145

Algorithm 1: Huffman Tree merging
Input: New Dataset (D), New dataset size (N), Old dictionary (O_D), Previous Model (M_O)
Output: New tree (Huffman code, Internal node, last epoch number, dictionary with new words)

```
1. Initialize empty dictionary for new dataset, $W_N$
2. For $i$ from 0 to $N$
   2.1 Split all words from $D_i$, $S_i$ = split words
       2.1.1  For $j$ in $S_i$
                   If $j$ does not exist in $O_D$ then
                       If $j$ exists in $W_N$ then
                           $W_N[j]$ + = 1
                       Else
                           $W_N[j]$ = 1
                           $A_W$ = append $j$ in the list
                       End if / *calculate the frequency of each unique word
                   Else
                       $A_W$ = append $j$ in the list / * append all words in the list
                   End if /* Check if new words exist in the old dictionary
       2.1.2  End for
3. End for
4. Obtain the shortest path ($S_P$) from the previous model ($M_O$)
5. Obtain the Huffman code ($H_{SC}$) at the leaf node on the shortest path ($S_P$)
6. Get all internal nodes ($I_{SN}$) from root node to shortest path's ($S_P$) leaf node
```

```
7. The leaf node ($L_N$) from the shortest path ($S_P$) is considered as the new root node, $N_R = L_N$
8. Create the new Huffman binary tree ($T_N$) with the new words only ($W_N$), where root identified from
   7# step
9. Obtain the new code by combining the shortest path code ($H_{SC}$) with new code ($H_{NC}$).
      For $i$ in $W_N$:
           Huffman code for each new word, $H_C[i] = H_{SC} + H_{NC}[i]$
           Internal node for each new word, $I_N[i] = I_{SN} + I_{NN}[i]$  /* new internal node ($I_{NN}$)
      End for
```

2.3.2 Feature Extraction from Incremental Word2vec Model

From the initial phase, the model is saved for incremental process. In the incremental model, only a new dataset is going to be trained. After incremental training, the embedding vector for all words in the new dataset is extracted. Therefore, embedding vectors for any words in the new dataset that appear in the previous corpus will be updated. Algorithm for extracting feature vectors from incremental Word2vec model is shown below.

Algorithm 2: Feature Extraction from Incremental Word2vec model
Input: New Data (D_N)
Parameters: Previous Word2vec Model (M_O), All data from Algorithm 1(Dictionary for new words (W_N), List for all words from new dataset (A_W)), old data (D_O)
Output: Updated Model (Word embedding vector, shortest path, Huffman code, Last epoch number)

```
1. Load the previous pre-trained Word2vec model ($M_O$)
2. Load all data from Huffman Tree merging algorithm (Algorithm 1)
3. For $i$ in $A_W$ (iterate through each word in list that contains all words from new dataset)
      If $i$ exist $W_N$ then /* check if word exist in the dictionary that holds only new words
           Randomly Initialize the input weight vector for each word ($i$)
      Else
           The input weight vector is retained from the previous trained model ($M_O$)
      End if
4. End for
5. To train the model, prepare the training data by selecting the $W_s$ left and right words for each word
   in the list ($A_W$)
6. Train the Word2vec CBOW model with all training data that contains all words from new dataset
   and get the optimal weight for each word and hidden layer weight using the backpropagation
   algorithm by minimizing the loss function
7. Combine dictionary from previous model with new dictionary, $W\_Dict = W\_Dict + W_N$
8. Calculate the word embedding vector ($E_V$) for each word in the dictionary ($W\_Dict$)
9. Combine old and new data, $D_O = D_N + D_O$
10. For $i$ from 0 to size ($D_O$) (iterate through for each data)
   10.1    Split each data into words ($S_i$)
   10.2    For $j$ in $S_i$ (iterate through for each word in split data, $S_i$)
                $F_v[i] += Ev[j]$
   10.3    End for
   10.4    $F_v[i] = F_v[i]/j$
11. End for /* Calculate the feature vector by averaging each word embedding vectors for all data
12. Save the model with shortest path, Huffman code from the Huffman tree, and the last epoch number
```

2.4 Microsoft Language Similarity Package

This study used the Microsoft language similarity package to calculate the centroid and evaluate the clustering performance by calculating the similarity score for each group. Microsoft Academic Graph (MAG) released the Language Similarity Package to calculate a similarity score between two sentences. The Microsoft language similarity package provides a pre-trained model with the API (Application Programming Interface) which accepts two strings (S1, S2) as input, and as an output it returns a semantic similarity score between S1 and S2 (Sinha et al. 2015). The similarity score is a floating-point value, closer to 1.0 meaning most similar and values closer to -1.0 meaning less similar.

2.5 Initial Clustering

The popular K-means algorithm is used for initial dataset clustering. K-means clustering with a large dataset is always a big challenge. The feature vector (FV) that is collected from Word2vec and TF-IDF method is fed into the K-means algorithm for clustering similar meanings of papers together. As a result, the cluster number for each paper and centroid (*C*) for each cluster are saved for incremental clustering.

2.5.1 Centroid Calculation for K-means

Choosing the centroids and *K* value is one of the big challenges in K-means algorithm. To get the finest clusters in terms of the low number of papers in each cluster, we calculated optimal K using Microsoft language similarity API. Our goal is to do clustering with the highest number of K such that when a researcher gets a direction to a cluster, there should not be large numbers of papers in the cluster. In our proposed algorithm, we set a threshold (T_H) value at 0.8 for abstract and 0.65 for title to calculate K value. The main idea is to iterate through all data into the system and select only those data to make a group where similarity score is greater than the threshold (T_H). Each time *K* is increased, the main dataset is updated by removing the matching data point that fulfills the above condition.

Algorithm 3: Calculate *K* value for K-Means
Input: Dataset (D), Total number of data (N), Threshold (T_H)
Output: Number of cluster (K)

```
1.  For $i$ from 0 to $N$
    1.1 For $j$ from $i$+1 to $N$
        1.1.1   Get the similarity score ($S_C$) by calling Microsoft language similarity API between $D_i$
                and $D_j$: $S_C$ = Language_Similarity_API ($D_i$, $D_j$)
        1.1.2   If $S_C > T_H$ then
                    Remove $D_j$ from $D$
                    Update $N$, where $N$= Updated dataset size
                End if
    1.2 End For
    1.3 $K = K$+1
    1.4 Remove $D_i$ from $D$
    1.5 Update $N$, where $N$= Updated dataset size
2.  End For
```

The overall complexity of the calculation *K* value is estimated as $O(N^2)$, where N is the dataset size.

2.6 Incremental Clustering

In this study, the main goal is to do clustering similar meaning of papers together without re-clustering when new papers are added. In the incremental clustering algorithm, the new data is iterated through all centroids by calculating the Euclidean distance between new data and all centroids to get the closest group of similar papers to the new paper. After obtaining the closest group, the average cosine similarity is measured between the new data and all members of the closest group. This average cosine similarity score is then compared with a threshold value. In this algorithm, the threshold (T_H) value is set at 0.8 for text body, and 0.7 for abstract and 0.60 for the title.

Algorithm 4: Incremental clustering:
Input: Feature Vector for New Dataset (*FN*), New Data (*DN*), New Dataset size (*N*), Feature vector for old dataset (*FO*), Threshold (T_H)
Parameters: Centroid (*C*), Previous Dataset (D_O), Number of Centroid (*K*)
Output: New dataset with cluster numbers

```
1.  For m from 0 to $N$ (iterate through for each new feature vector, $FN$)
    1.1 For $i$ from 0 to $K$ (iterate through for each centroid)
        1.1.1   Find the closest centroid by using the Euclidean distance between each centroid ($C_i$)
                and new feature vector ($FN_m$).
    1.2 End for
    1.3 Get all the members from the closest centroid group and count the number of member ($NM$)
    1.4 Get the feature vector for all members in the closest group ($FC$)
    1.5 For $j$ from 0 to $NM$ (iterate through for all members from the closest centroid group)
        1.5.1   Calculate cosine similarity score between each feature vector in the closest group
                member and new data's feature vector, $cos\_sim$ = cosine_similarity ($FN_m$ , $FC_j$)
    1.6 Calculate the average cosine similarity score ($avg\_cos$) by averaging $cos\_sim$ from steps 1.5.1
    1.7 If $avg\_cos > T_H$ then
            Update centroid for the closest centroid by adding new feature vector ($FN_m$)
            Update the closest group member by adding the new data ($DN_m$) into that group
        Else
            Assign the new feature vector ($FN_m$) as a new centroid
            Assign a new cluster number for the new data ($DN_m$)
        End if
2.  End for
```

Time performance is very important for an algorithm. For the algorithm 4, the time complexity is $O(N*K*M*H)$, where N is the number of new datasets, K is the number of centroids, H is the size of our word embedding vector, which is 300. M is the number of members in the closest centroid group.

2.7 Calculate Similarity Score from Microsoft Language API

In this study, the clustering quality is evaluated using the Microsoft language similarity tools. After incremental clustering, the average group similarity score was measured using Microsoft language similarity API. The following formula is used to calculate the similarity score in a group.

$$T = \frac{(M*(M-1))}{2} \tag{1}$$

Where, *M* is number of members in a cluster. Calculate average similarity score in a group required total *T* times to call Microsoft language similarity API.

$$AS = \frac{\sum_{i=1}^{M}\sum_{j=i+1}^{M} Compute(S_i, S_j)}{T} \tag{2}$$

In the formula (2), the Microsoft language similarity API is called using the Compute (S_i,S_j) method, where S_i and S_j are two sentence in *i*th and *j*th position and *AS* is the average similarity score for a cluster.

The following formula is used for calculating weighted average score for all cluster.

$$Weighted\ Average\ Score = \frac{\sum_{i=0}^{N}(AS_i \ * M_i)}{\sum_{i=0}^{N} M_i} \quad (3)$$

Where AS_i is the average similarity score for ith cluster and M_i is the numbers of members in ith cluster and N is the total number of clusters.

3 EXPERIMENTAL RESULT

3.1 Initial Clustering Analysis

The first experiment was conducted for the initial dataset clustering with an experimental dataset by using TF-IDF and Word2vec. As shown in Table 2, Word2vec has fewer number of clusters with single member as compared to TF-IDF. By comparing all clustering results on the title, abstract and full_body, a higher weighted score is achieved for full_body using both methods. It can be observed the system performs better with more words.

Table 2: Initial Clustering Analysis using TF_IDF and Word2vec (44,232 dataset).

	Title		**Abstract**		**Full_body**	
	TF-IDF	**Word2vec**	**TF-IDF**	**Word2vec**	**TF-IDF**	**Word2vec**
Weighted average score	0.54	0.59	0.78	0.81	0.85	0.88
No. of clusters with single member	2025	240	707	224	497	203
No. of words	390,690		4378,223		6,5731,965	

3.2 Incremental Clustering Analysis

In this study, three different datasets for incremental clustering were collected. The first dataset (called Dataset 1) was released on July 5th, 2021, and the total size of the dataset after filtering for COVID-19 related keywords was 17,949 papers. The second batch of papers was released on September 9th, 2021. The final dataset (Dataset 3) was released on November 29th, 2021. The initial dataset was released on April 19th, 2021.

3.2.1 Incremental Clustering Analysis on Title Dataset

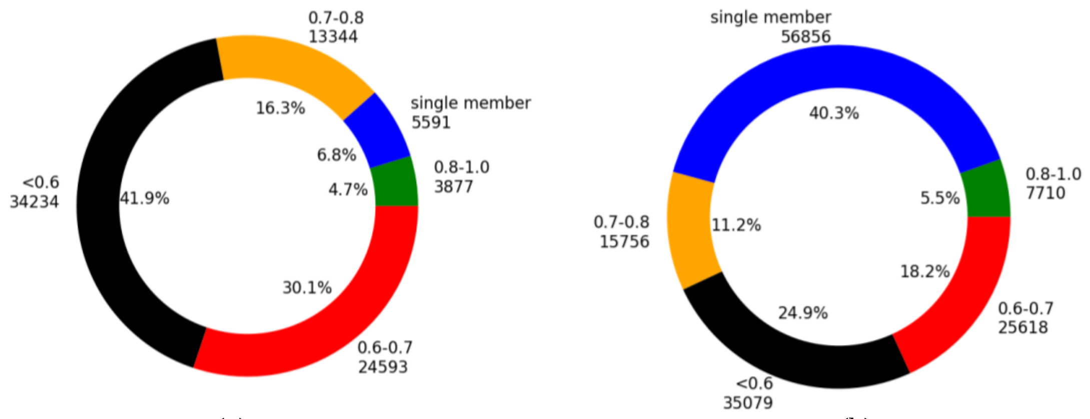
Table 3 shows that number of clusters are increased significantly in each batch of title clustering. Many members from the new dataset were not able to find their matching cluster to the previous clustering result.

Table 3: Incremental clustering analysis on title dataset.

Clustering algorithm	**No. of data**	**No. of total words**	**No. of clusters**	**No. of clusters with single member**	**No. of members where group score < 0.6**	**Weighted average score**
Initial K-means	81,639	755,484	13,821	5,591	34,234	0.56
Incremental (Dataset 1)	17,949	177,633	30,646	22,094	34,477	0.47

Incremental (Dataset 2)	25,582	256,575	53,493	43,878	37,189	0.38
Incremental (Dataset 3)	15,849	159,931	67,265	56,856	35,079	0.37

The main group is divided according to the average similarity score. The first group is referred to as "0.80-1.0", which is represented by green color in Figure 2, and all clusters with an average similarity score between 0.8 and 1.0 will be placed in that group. Similarly, the yellow and red colors represented two groups which include all clusters with an average similarity score between 0.7 to 0.8 and 0.6 to 0.7. The "single member" group includes all clusters that have single member only and represents by blue color. The last group is designated as "<0.6" or "lowest-scoring group", which is represented by black color, and it contains all clusters with an average similarity score of less than 0.60. From Figure 2, the number of single member increased to 56,856 after third batch of incremental clustering. The number of members also increased in "lowest-scoring group" (<0.6). The weighted average score dropped significantly in each batch because of increasing the new clusters with the single member.



(a) (b)

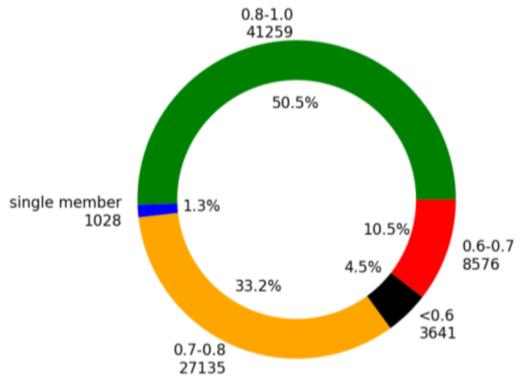
Figure 2: Clustering analysis - title dataset (a) Initial clustering (b) Third batch of incremental clustering.

3.2.2 Incremental Clustering Analysis on Abstract Dataset

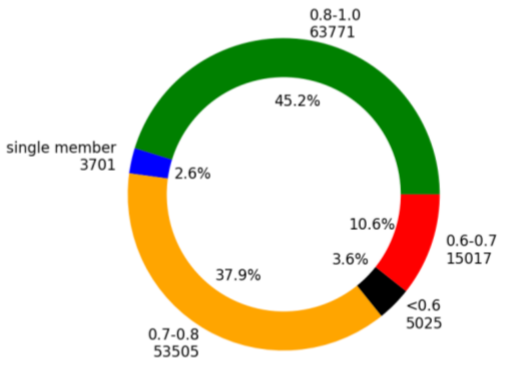
In comparison with title clustering, abstract clustering performs better. Table 4 shows that most of the members on the new dataset were paired with their corresponding cluster on the previous clustering result. The weighted average score slightly dropped because of increasing the total dataset as well as increasing the new clusters with the single member. From Figure 3 we can see that by increasing the number of members in the "highest-scoring group" (0.8-1.0), the average similarity score increased.

Table 4: Incremental clustering analysis on abstract dataset

Clustering algorithm	**No. of data**	**No. of total words**	**No. of clusters**	**No. of clusters with single member**	**No. of members where group score < 0.6**	**Weighted average score**
Initial K-means	81,639	8175,733	6,960	1,028	3,641	0.80
Incremental (Dataset 1)	17,949	2133,608	8,140	2,089	4,054	0.79
Incremental (Dataset 2)	25,582	3027,858	9,394	3,097	4,688	0.78
Incremental (Dataset 3)	15,849	1926,801	10,185	3,701	5,025	0.78



(a)



(b)

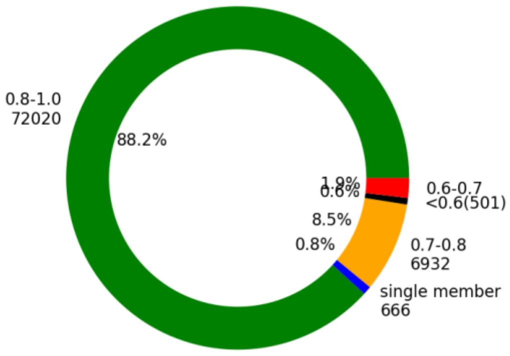
Figure 3: Clustering analysis for abstract (a) Initial clustering (b) Third batch of incremental clustering.

3.2.3 Incremental Clustering Analysis on Full_body Dataset

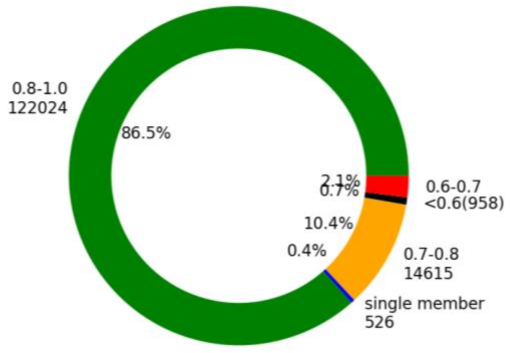
Table 5 and Figure 4 show the results of clustering on the full_body dataset for initial and incremental clustering. In each batch of incremental clustering, a few numbers of new clusters were added. The number of clusters with single member decreased in full_body clustering, while in the title and abstract clustering, this number increased. It is clear that full_body clustering outperforms abstract and title clustering with regard to clustering quality.

Table 5: Incremental clustering analysis on Full_body dataset.

Clustering algorithm	**No. of data**	**No. of total words**	**No. of clusters**	**No. of clusters with single member**	**No. of members where group score < 0.6**	**Weighted average score**
Initial K-means	81,639	14,1132,910	8,431	8,431	501	0.86
Incremental (Dataset 1)	17,949	3,6769,763	8,435	8,435	606	0.86
Incremental (Dataset 2)	25,582	5,1554,772	8,438	8,438	791	0.86
Incremental (Dataset 3)	15,849	3,4863,541	8,439	8,439	958	0.86



(a)

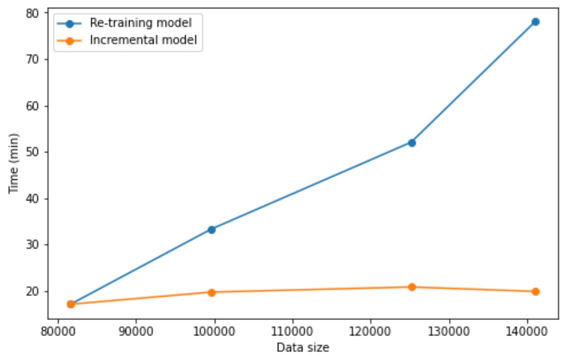


(b)

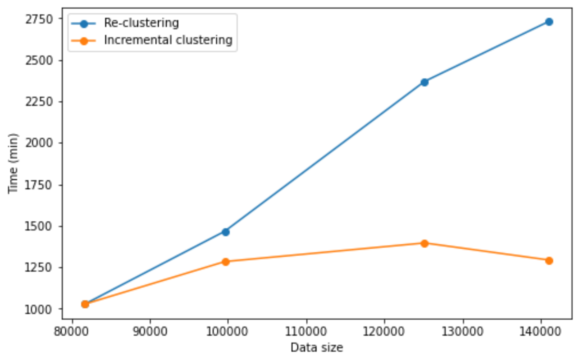
Figure 4: Clustering analysis for Full_body (a) Initial clustering (b) Third batch of incremental clustering.

3.3 Computation's Time

Analyzing the computational time is essential along with evaluating performance through similarity score calculations. To generate the embedding vectors, this study used an incremental model that does not require re-training the model with newly added dataset. Also, the incremental clustering algorithm does not required re-clustering. By using this approach, we were able to reduce computation time. Figure 5 illustrates how our proposed approach involved a shorter computation time in comparison to re-clustering and re-training the model from scratch on abstract data.



(a)



(b)

Figure 5: Comparing time between incremental process and re-doing from scratch (a) Incremental model and re-training model (b) Incremental clustering and re-clustering.

4 CONCLUSION AND FUTURE WORKS

For a large dataset of research papers, clustering text data is a challenging task. In particular, when the dataset is dynamic as new research papers are frequently published. In order to avoid re-clustering, the incremental algorithm is proposed in this paper. The proposed Word2vec model is trained incrementally with a new dataset of the published papers instead of training the entire dataset from scratch. The proposed incremental approach decreases the processing time 93% in comparison to re-clustering the whole data set for 141,019 papers.

Moreover, the experiments showed the clustering algorithm achieves a better similarity score with the full body of papers in comparison with clustering the title or abstract of the papers. It demonstrates that using more words in the text dataset is more effective in increasing cluster quality. The conducted experiments shows when clustering with the full body of papers, the weighted average score is 0.86 in compared to 0.78 and 0.37, which are achieved for clustering with abstracts and titles of the papers respectively using incremental Word2vec model. Regrading to comparison of used NLP method to find similar papers, the experiments with clustering the initial dataset demonstrate that Word2vec performs better than TF-IDF method. To improve the proposed method, further steps can be taken for the improvement of assigning the new data to the clusters for future research. Another direction of this work would be feeding the papers of other research topics instead of COVID-19 into the database system that is created in our research lab.

ACKNOWLEDGMENTS

This research is funded by FOS Dean Research Fund DRF-CONNECT, Ryerson University, Canada.

REFERENCES

Abhari, A., and M. Nilufar. 2020. “A Prototype System for Clustering Covid-19 Research Papers”. In *Proceedings of the 2020 Winter Simulation Conference (WSC20)*, edited by K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing. Available via https://informs-sim.org/wsc20papers/080.pdf. Accessed Jun. 06, 2022.

Abhari, A. 2020. “Database System”. Distributed Systems and Multimedia Processing Lab (DSMP). Available via http://dsmp.ryerson.ca/COVID_report/COVID_report/COV19_database.htm. Accessed Jun. 06, 2022.

Huffman, D. A. 2006. “A Method for the Construction of Minimum-redundancy Codes”. *Resonance*, vol. 11, (2), pp. 91-99.

Kazemi, B., and A. Abhari. 2017. “A Comparative Study On Content-Based Paper-to-Paper Recommendation Approaches in Scientific Literature”. In *Proceedings of the 2017 Spring Simulation Conference*, Virginia Beach, VA, USA.

Microsoft Language Similarity Package. 2015. Available via https://docs.microsoft.com/en-us/academic-services/graph/language-similarity. Accessed Jun. 06, 2022.

Mikolov, T., K. Chen, G. Corrado, and J. Dean. 2013. “Efficient Estimation of Word Representations in Vector Space”. In *Proceedings of International Conference on Learning Representations*, Arizona, USA.

Peng, H., J. Li, Y. Song, and Y. Liu. 2017. “Incrementally Learning the Hierarchical Softmax Function for Neural Language Models”. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, Association for the Advancement of Artificial Intelligence, San Francisco, California, USA, pp. 3267–3273.

Radu, R., L. Radulescu, M. Mariana. 2020. "Clustering Documents using the Document to Vector Model for Dimensionality Reduction," *IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pp. 1-6.

Sinha, A., Z. Shen, Y. Song, H. Ma, D. Eide, B. Hsu, and K. Wang. 2015. “An Overview of Microsoft Academic Service (MAS) and Applications”. In *Proceedings of the 24th International Conference on World Wide Web (WWW ’15 Companion)*, ACM, NY, USA. pp. 243-246.

Wang, L. L., et al. 2020. “CORD-19: the COVID-19 Open Research Dataset”. In *Proceedings of the 1st Workshop on NLP for COVID-19*, Association for Computational Linguistics.

AUTHOR BIOGRAPHIES

MAHFUJA NILUFAR is a graduate student at the department of computer science, Ryerson university since September 2019. Her email address is mahfuja.nilufar@ryerson.ca.

ABDOLREZA ABHARI is a Professor in the Department of Computer Science at Ryerson University and director of DSMP lab. He holds a Ph.D. in Computer Science from Carleton University. His research interests include data science, web social networks, AI and agent systems, network simulation, and distributed systems. His email address is aabhari@ryerson.ca.