Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, February : 2024

# DYNAMIC DATA RETRIEVAL USING INCREMENTAL CLUSTERING AND INDEXING

[#1]**CH. VAMSHI RAJ,** Research Scholar,
[#2]**Dr. YOGESH KUMAR SHARMA,** Associate Professor & Guide,
[#3]**Dr. M. ANJAN KUMAR,** Professor & Co-Guide,
Department of Computer Science & Engineering,
SHRI JAGDISHPRASAD JHABARMAL TIBREWALA UNIVERSITY, RAJASTHAN.

**ABSTRACT:** The evolution of the Internet and real-time applications has contributed to the growth of massive unstructured data which imposes the increased complexity of efficient retrieval of dynamic data. Extant research uses clustering methods and indexes to speed up the retrieval. However, the qualityof clustering methods depends on data representation models where existing models suffer from dimensionality explosion and sparsity problems. As documents evolve, index reconstruction from scratch is expensive. In this work, compact vectors of documents generated by the Doc2Vec model areused to cluster the documents and the indexes are incrementally updated with less complexity usingthe diff method. The probabilistic ranking scheme BM25+ is used to improve the quality of retrieval for user queries. The experimental analysis demonstrates that the proposed system significantly improves the clustering performance and reduces retrieval time to obtain top-k results.

***KEYWORDS:*** *Incremental Clustering, Inverted Indexing, Information Retrieval, Ranked Retrieval, Representation Learning*.

## 1. INTRODUCTION

With the innovation in technology over the past two decades, the emergence of social network organization, adoption of hand-held computerized gadgets, the explosion in the usage of the Internetand computing services contributed to the tremendous growth of heterogeneous data of structured, semi-structured, and unstructured type, commonly known as Big Data. Consistently, 2.5 quintillion bytes of data are generated every day (EDBD Statistics, 2015) as emails, audios, videos, web pages, social media messages, and so forth, where 90% account for unstructured data. The growth in datacontributes to the increased complexity of the efficient retrieval of these data. Available conventionalmethods are well suited for static data, but the above requirements demand a more efficient way oforganizing and processing the dynamic unstructured text data.

In this big data era, querying the large data necessitates the organized storage where the incomingdata (usually represented as vectors) are categorized based on the similarity of vectors. Thus, similar documents can be retrieved quickly for user queries instead of handling large data instantly. As documents evolve, the clustering algorithms should cope with the dynamic nature of data with minimum sacrifice to clustering quality. Several clustering algorithms are proposed with different data representation models (Ding and He, 2004; Campr and Jezek, 2015), similarity measures (Audhkhasi and Verma, 2007; Huang, 2008), and grouping techniques (Dhillon et al., 2004; Shindler et al., 2011; Cai et al., 2013). The data representation refers to the number of classes and the available patterns applicable to the clustering algorithm. Good representations capture a vast number of possible patterns. Hence, the quality of clustering algorithms is highly dependent on representation learning.To transform the data into more cluster-friendly in this big data era, representation learning models(Mikolov et al., 2013b; Pennington et al., 2014; Yang et al., 2016; Kim et al., 2017; Joshi et al., 2018; Ren et al., 2019) are used to generate the distributed representation of words. Good representations capture a vast number of possible patterns. Hence, the quality of clustering algorithms is highly dependent on representation

learning. Traditional machine learning models always result in a locally optimum solution, whereas distributed representation learners are trained by many samples to learn the representation. To state the expressiveness, traditional machine learning models such as decision tree, support vector machine (SVM), etc., requires O(N) input samples to distinguish O(N) regions. In contrast, distributed representation learning models represent the $O(2^k)$ region for the same samples (Bengio et al., 2013) (where k denote the count of non-zero elements in distributed representation). While clustering intends for efficient organization of data to improve the retrieval performance,

the complexity of the search operation in dynamic data is yet another challenge. Applying proper indexing methods shows the good impact on query processing by reducing the complexity of the search operation. Due to the unordered form of input, the mode of search is by its content, i.e., Keyword search. In practice, an inverted index is the most popular indexing method for keyword search on unstructured data. Considering the dynamic nature of the data, the indexing must be dynamic for efficient retrieval. Existing research works mainly concentrate on reducing the index build time and keyword query processing time. However, most of the current works focus on static data. On the other hand, this work differs in improving the accuracy of dynamic clustered data with less retrieval time. Even though several papers dealt with clustering, indexing, and ranked query retrieval individually, existing methods cannot fulfill the needs of the dynamic nature of unstructured data. Hence, to build an efficient data retrieval system for dynamic unstructured data, there is a necessity for incremental updates on clustering and indexing as well. This paper aims to use a learning model for data representation in order to improve the performance of clustering on dynamic data. To further improve the data retrieval efficiency, it uses an incremental inverted indexing structure with a probabilistic

ranking scheme to retrieve top-k relevant documents for the given user queries.

## 2. REVIEW OF LITERATURE

This section provides a brief review of various data representation and clustering methods. A review of the different update methods proposed for inverted indexing is also presented.

Clustering, a machine learning algorithm, is highly dependent on the data representation methods. The data representation methods can be divided into conventional and deep learning methods. The conventional methods include vector space model (Salton et al., 1975), Latent Semantic Analysis (LSA) (Deerwester et al., 1990), Latent Dirichlet Allocation (LDA) (Tang et al., 2003), and so on. Recently, researchers have extended the models for distributed vector representations, also known as word/document embeddings, where the word/document is represented as a vector of continuous real values. The various methods are proposed to build effective word/document-level representation (Mikolov et al., 2013a; Mikolov et al., 2013b; Joshi et al., 2018) and its applications in sentiment analysis (Tang et al., 2014; Ren et al., 2016; Wang et al., 2016; Fu et al., 2017; Lauren et al., 2018; Liu et al., 2018), text classification (Joulin et al., 2016; Bojanowski et al., 2017; Yang et al., 2018) and clustering (Kim et al., 2017; Ren et al., 2019). The above research works use either Word2Vec (Mikolov et al., 2013a) or Glove (Pennington et al., 2014) to represent the documents in vector representation. However, these approaches provide a vector representation of words rather than the whole document, and the vectors generated are of variable length representation. To overcome this issue, an extension of Word2Vec called Doc2Vec (Le and Mikolov, 2014) is developed that learns the meaning of documents in a fixed size.

The most commonly used machine learning algorithm for clustering is K-Means and its variants (Kumar and Reddy, 2017; Gupta and Chandra, 2019). Considering the dynamic nature of the data, the changes have to be updated incrementally in the clusters. Hence, the most popular incremental clustering technique “Mini-batch K-means” (Sculley, 2010) has been proposed to reduce the computational time of finding a partition. Both classical K-means and Mini-batch K-means are compared in (Feizollah et al., 2014). It shows that Mini-batch K-means gives better performance on a large dataset with a less computational cost.

Information Retrieval systems use clustering for organized storage of the data. In the context of the keyword search, an inverted index is the most commonly used index to improve the keyword query processing throughput further. Hence, a brief survey on

index update methods is presented inthis section. Brin and Page in Google search engine proposed a method called Forward Index (Brin and Page, 1998) where the forward index builds for every new document, and changes happen in the old index even for a modified document. The Index Rebuild (Zobel and Moffat, 2006) periodically rebuilds the index from scratch for every update. Re-scan is required even for a single update in a single document. The Intermittent merge method (Zobel and Moffat, 2006) maintains two indexes where the new document's indexes reside in the main memory to reduce the computation time and merged with the on-disk index once the memory is full. The most popular approach is Delta change-append only (Galambos, 2006). Here the index is merged with new documents index, and when a document has to be removed, the entry is denoted as "deleted." The variants of these general methods (Asadi and Lin, 2013; Rissola and Tolosa, 2015; Siddiqa et al., 2017; Kumar et al., 2018; Malik et al., 2019; Tekli et al., 2019) has been used in different applications. When the documents are highly dynamic, i.e., the documents are altered as well as newly inserted, inserting or removing a word in a document may influence the word positions broadly. Therefore, the word positions having modifications should be marked and updated on the same index. Creating different indexes for new documents leads to disk overhead. Hence, the requirement is on updating the same indexes with a less computational cost.

Text documents form a huge part of the Internet world today. Recent works on word/document embeddings have improved performance in various applications. Existing solutions on data representation and clustering satisfy the needs of static unstructured data. Similarly, efficient data retrieval has become inefficient due to the ever-growing sizes of the index, finding exact relevant documents, and the computational time taken for data retrieval. Hence, there is a huge scope to make information retrieval efficient that fulfill the requirements of dynamic unstructured data.

## 3. RELATED WORK

Large volumes of dynamic data are continuously generated from various applications such as social networks, telecommunications, search engines, and so on. Given a keyword query, the retrieval of exact relevant documents in a reasonable time, while the underlying data would be dynamically changed, is a challenge. Hence, we aim for an efficient information retrieval system that can deal with the dynamic nature of data with feasible time. Partitioning the data and indexing provides a huge scope to reduce information retrieval time. Clustering has turned out to be one of the most significant techniques used for partitioning the data. Applying a suitable vector representation model provides better results for partitioning the data promptly. Building an inverted index shows a good impact on query processing. Owing to the dynamic nature of data, the clusters and indexes must be updated immediately to provide exact relevant results. To achieve this, our system uses a representation learning model to generate low dimensional vectors for clustering and uses an inverted index structure with a probabilistic ranking scheme. The system supports dynamic operations incrementally to improve data retrieval efficiency and produces top-k results for the user queries.

The flow diagram of the proposed system is shown in the Figure 1. It consists of a master server, index servers, and slave nodes for data storage. The master server has direct access to the index servers and slave nodes, respectively. Similarly, index servers have direct access to the slave nodes. As shown in the figure, the system works on five steps with the given entities. The responsibilities of each entity are described with the workflow of the system.

**Step I:** The master server is the primary server where the raw input data is given, users communicate the query and request results. The server maintains the metadata about the whole system and responsible for the following tasks: pre-processing of documents, training and testing the data model, and clustering of documents. Pre-processing is performed on all the documents to remove redundant and unnecessary words. When documents arrive, the pre-trained model is used to produce document-level vectors. The generated vectors are clustered with the incremental clustering algorithm.

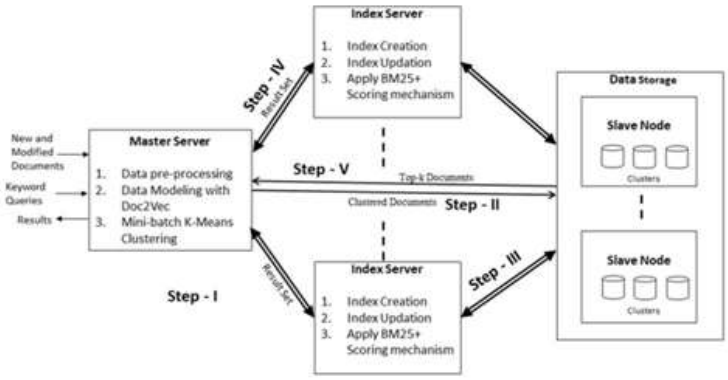
**Step II:** The master server clusters the documents and distributes to the respective slave nodes for storage. The distribution happens based on the topics obtained as a result of clustering. To support the dynamic nature of data, whenever new or modified documents arrive, the step-1 repeats. Thus, the set of new or modified documents are clustered and distributed to the slave nodes.

**Step III:** For each slave node, an inverted index is created on the documents at the index server for efficient retrieval of data. The

document store called slave nodes is separate, and all the index servers have access to this data. The indexes are stored locally on the respective index server for each slave node. This completes the index creation phase. An index update is performed when a document is to be modified/deleted. Every slave node is responsible for sharing the updates with the corresponding index server. When the cluster has updates with existing documents, an old document is retrieved from the slave node, and depending on the results of the comparison of old and new documents, the index is updated on the index server. In the same way for the deletion of documents, the respective entry is updated in the index server.

**Step IV:** When the user submits the keyword queries to the master server, the queries are forwarded to the corresponding index servers to retrieve the results. Hence, Data Retrieval involves locating the relevant index servers and processing the request on the index structure. The docIDs (document ID) are retrieved from the index and then ranking these documents based on probabilistic scoring. The result set is forwarded to the master server.

Figure 1. Flow diagram of the proposed system




**Step V:** Based on the result set received from index servers, the master server fetched the resulting documents from the selected slave nodes and presented to the user. The steps mentioned above show the workflow of our proposed system. The description of every step is given in the next section.

## 4. SYSTEM DESIGN

This work aims to develop a system for the efficient retrieval of dynamic unstructured data using incremental clustering and indexing. The system is divided into two logical phases. In Phase 1, the unstructured data is pre-processed, and the learning model is applied for vector representation that makes the clustering process agile. In Phase 2, the clustered documents are indexed on the respective servers, and the probabilistic scoring scheme is applied for ranked retrieval of user queries.

**Data Modeling and Clustering**

The objective of the first phase is to pre-process the raw unstructured data into cluster-friendly data. Any redundant or irrelevant information needs to be removed, retaining any useful information before indexing. To transform the data into more cluster-friendly, in our work, an unsupervised learning model called Doc2Vec is used. Doc2Vec has two models, such as Paragraph Vector-Distributed Memory (PV-DM) and Paragraph Vector-Distributed Bag of Words (PV-DBOW). This paper uses PV-DBOW, which is based on the Skip-gram model (Mikolov et al., 2013a) to generate low dimensional vectors of document-level data.

The Doc2Vec, a feed-forward neural network, comprises an input layer, one hidden layer, and an output layer. Every document consists of 'docID' and 'set of words.' Initially, the 'docID' and the words in the document are mapped to a unique vector. Let the dimension (size of vocabulary) of the 'docID,' and the words are 'D' and 'W' respectively. Now, the input layer comprises a concatenation of a docID vector and a set of word vectors (D+W), which is fed into the network. After the network is trained, every document has its embedded vectors of size 1*K, where 'K' represents the size of the hidden layer. Regardless of the length of the document, the vectors generated are of fixed size. In this paper, the pre-trained model is used to generate the continuous dense

representation of the entiredocument. When the two documents have a similar context, then they yield similar embeddings, which takes less time to cluster the data. The order of time complexity for Doc2Vec is given in equation 1.

$$Time Complexity\left(Doc2Vec\right) = E \times T \times \left(K \times \left(\mathrm{I} + \log_2\left(D + W\right)\right)\right) \quad (1)$$

where I, E, and T represents the window size, the number of epochs and words used for training. Thevectors generated are clustered with the Mini-batch K-means clustering algorithm. The vital goal ofthis clustering algorithm is to utilize small random batches of fixed size vectors to store them easilyin memory.

Algorithm 1 shows the flow of Doc2Vec and Mini-batch K-means for dynamic documents. Thenumber of clusters, mini-batch size, and the maximum iterations is initialized. The Doc2Vec modelthat is already trained is saved in the disk, which can be loaded from disk for a set of new documents,and the vectors are inferred automatically. Hence, the model produces a set of document vectors. The similar embeddings help to find similar context documents efficiently. Whenever new documents arrive, or the existing document comes with an update, a fresh sample of document vectors is obtainedfrom the data model. Then, the nearest cluster center for the new documents is calculated with the objective function f(C,d). The objective function determines the distance between each point in 'N' (randomly selected samples) and 'K' centers, and then the document is assigned to the closest center. The count for each center is updated with the vectors, and the learning rate is determined as the inverse of the number of samples assigned to a cluster during the process. The applied learning rate decreases with the number of iterations. The clusters are updated, and the process is repeated until the clusters converge. An increase in the count of iterations reduces the outcome of recent examples so that convergence can be noticed in the existing condition without any modifications in the clusters. Moreover, the perfect balance of accuracy and computation time is taken care of that makes the real- time clustering efficient. The time complexity of Mini-batch K means is linear. Using document level representation produces fixed size dense vectors, and similar context documents are close in a low dimensional semantic space. However, the traditional TF-IDF generates similar words and synonyms as independent features in a high-dimensional space, which increases the time to find similarity.

Algorithm 1. Doc2Vec and Mini-batch K-means Algorithm

```
Input: Given Dataset D, Number of clusters K, Mini-batch size b, Max-iterations t
Output: List of updated clusters, C = {C1, C2, …)

Load Doc2Vec Model
        for document d ϵ D do
                infer a fixed size document-level vector, v
        end for

Initialize each center, c ϵ C with a 'd' randomly picked from D
u ← 0
for i ← 1 to t do
        N ← b samples randomly picked from D
        for d ϵ N do
                x[d] ← f(C, d)
        end for
        for d ϵ N do                    // update cluster center with each batch
                c ← x[d]
                u[c] ← u[c] + 1
                η ← 1/u[c]              // learning rate for each cluster center
                c ← (1- ηc + ηd         // update cluster center with gradient
        end for
end for
```

**Inverted Indexing**

The objective of the second phase is to reduce the complexity of the search operation using indexing.An inverted index contains two major components, such as the dictionary and the postings list. Forevery term in the set of documents, the posting list holds data about the term's occurrences in the document collection. Each posting list relates to a word. It stores all the docIDs where this word appearsin sorted order. However, when collections are varied frequently with new or updated documents, new terms must be updated to the dictionary, and the posting lists must be reviewed for the updates.A disk-based B+ Tree implementation is created

for building inverted indexes where the keys are distributed over the nodes. Nodes are brought into memory as they are accessed, so the entire tree does not have to engage memory. Indexing is done at the index server for each slave node locally. The tree supports the standard dictionary operations such as insert, delete, update, and lookup for access and modification of indexes. The following subsection describes the operations briefly.

Algorithm 2. Index creation

```
Input: Set of pre-processed documents, D = d1, d2, d3 ... dn
Output: Inverted Index structure using disk-based B+ tree

set threshold for a node in B+ Tree.
for each document 'd' in D
        for each term in the document, d
                if term not present
                        insert (term, docID) into B+ tree
                else if the term already added once
                        increase the term frequency count for the document, d
                end for
                if a threshold is reached, flush node to disk.
        end for
merge the set of partial nodes to form single index tree.
```

**Dictionary Operations (Insert, Delete, Update, Lookup)**

The term or query keyword and the docIDs are involved in performing these operations. The first operation is the insertion of the terms into the index. Insertion is straightforward. If the term being inserted is new, then depends on the availability of node, a B+ tree insertion is called to add a new data item, else if the term is already present, the docID is appended to the list of docIDs for the respective term. Algorithm 2 outlines the process of inverted index creation. The time complexity of insertion is $\Theta(\log_2 n)$ where 'n' represents the number of terms in the B+ tree and the time complexity of number of disks read and writes demand $O(\log_t n)$ where the co-efficient 't' is due to the operations performed for each node in the memory . The second operation is the deletion of the terms from theindex. If the term being deleted has only one docID as value, then a B+ tree node deletion is calledto delete the data item, else if the term has more than one docID present, simply the docID is deletedfrom the list of docIDs for the respective term. The time complexity of deletion is the same as of insertion. The third operation is an index update. When a new document or existing document withupdates is added to the collection, all the distinct terms extracted needs to be indexed. If the document is new, then insert operation is called for indexing new terms. When the document stored in the disk has updates on the terms already indexed, with reference to the landmark method (Lim et al., 2003),the update function involves the following items:

- The score of deleted document $d_1$,
- The score of new document $d_2$,
- The Diff operation between $d_1$ and $d_2$ does the update.

The update involves the deletion of index terms that are removed from the document and the insertion of new terms that are added to the documents. The node which contains changes alone taken for processing. In order to perform the delete operation, the frequency count (with which score is calculated) of the old document is compared with the one in the new document. If there are any changes, then the difference between the frequency count is calculated, and the old docID is deletedfrom the index, i.e., the node which contains update is removed from the tree. Then, the updated docID is added to the index. Finally, the index is modified with new scores. Update operation makes use of earlier mention insert and delete functions. The Algorithm 3 outlines the procedure of index update. If a single edit operation involves insertion or deletion of contiguous words, then the number of update operations(U) is at most $U +$ □ . Hence, the complexity of diff operation involves Ω (m + n) where 'm' and 'n' represents the number of words in the old and new document. This update involves fewer operations compared to the complete index rebuild. The last operation is lookup, where the user gives the query terms as input. The B+ tree nodes are traversed from the root, and the list of docIDs containing query terms are retrieved. The results from all the slave nodes are collected and

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, February : 2024

produced a final result at the master server. The time complexity of the search is O(*t*. log*t n*) where the number of disks read and writes demand O(log*t n*) .

Algorithm 3. Index Updation

Input: Old document tokens, $D_o = \{o_1, o_2, \ldots, o_n\}$, New document tokens, $D_n = \{n_1, n_2, \ldots, n_n\}$
Output: Modified Inverted Index tree

pre-process both $D_o$ and $D_n$.
compare the terms among $D_o$ and $D_n$.
make a Diff list with new and old terms.
for each insertion
    insert (term, docID) into B+ tree
    increase the term frequency count
end for
for each deletion
    delete (term, docID) from B+ tree
    decrease the term frequency count
end for
replace the old document with the new one.

**Data Retrieval**

Algorithm 4 outlines the steps of data retrieval using BM25+. The pre-processed queries are submitted to the system, and the search retrieves a list of docIDs which contain the terms. Initially, ascore is assigned for each term present in the document, and the weight for a term is calculated with a BM25+ weighting scheme. The final score represents the cumulative score of each term present in the document. Once the calculation of the score is complete, sorting of documents takes place in non-increasing order, and the master server retrieves top-k documents. The scoring function used toretrieve top-k documents is called the Best Matching function (BM25+) (Lv and Zhai, 2011), which is based on a probabilistic retrieval framework, BM25. The possibilities for very long documents inunstructured data is high. To successfully retrieve the long documents, there are two constraints to besatisfied. 1) The occurrence of a query term in long documents plays a significant role in the score calculation. Suppose if a query term is present in document D1 but not in D2, and both documents have the same relevance score based on remaining query terms, then the D1 score should be higherthan D2. BM25+ can satisfy this condition by taking the weight of the missing term as 0. 2) The coverage of more distinct query terms. Given a query 'Q', having keywords $q_1, \ldots, q_n$, the calculation of BM25+ score of document 'D' is given in equation 2 as:

$$Score(D,Q) = \sum_{i=1}^{n} IDF(q_i).\frac{f(q_i,D)(k_1+1)}{f(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)} + \delta \qquad (2)$$

where $f(q_i, D)$ represents the frequency count of $q_i$'s term in D, '|D|' represent the size of D in words, 'avgdl' represent the average of D's size in the text collection from where documents are retrieved, '$k_1$' and 'b' are open parameters, generally preferred, in the lack of an advanced optimization, b = 0.75, 'IDF($q_i$)' is Inverse Document Frequency weight of $q_i$, 'δ' is lower bounding term-frequency normalization factor. The computation of IDF is shown in equation 3 as follows:

$$IDF(q_i) = \log\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \qquad (3)$$

Where 'N' represents the number of documents in the collection, 'n($q_i$)' represents the number of documents containing the query term '$q_i$'.

**Algorithm 4. Ranked Data Retrieval using BM25+**

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, February : 2024

Input: Queries, Q = $q_1$, $q_2$, $q_3$,..., $q_n$, Inverted Index, B
Output: Top-k documents

set score($d_i$)=0 for each document, d
pre-process the query terms
for each query term, q in Q
    fetch the inverted list for q
    for each pair of (document,$f_d$) in Inverted list
        calculate weight, $W_{q,d}$ using scoring mechanism BM25+
        score(d) = score(d) + $W_{q,d}$
    end for
end for
sort the documents in decreasing order
retrieve the top-k documents and present it to the user.

## 5. PERFORMANCE ANALYSIS

In this section, the experimental outcomes of the proposed system are presented. The experiments were carried out on a Linux machine with the following configurations. To speed up the overall computation, the data are distributed to multiple nodes. The Reuters-21578 dataset from David D. Lewis (1999) is used in this system for evaluation. The primary goal is to analyze the potential effectiveness of the proposed system in modelingthe documents, clustering, and indexing incrementally for dynamic data. Further, we present the experimental results evaluated by the evaluation metrics. Then the effectiveness of clustering by using data modeling is studied. Finally, the incremental clustering and indexing results are reviewed.

**Experimental Setup**

The complete implementation is conducted on systems with the following configurations:

Master Server node: 64 bit Intel Xeon CPU E5-2650 v2 @ 2.60GHz x 4 core processor with 64GBRAM. Disk size is 200GB.

Index Server nodes: 64 bit Intel Xeon CPU E5-2650 v2 @ 2.60GHz x 2 core processor with 16GBRAM. Disk size is 200GB.

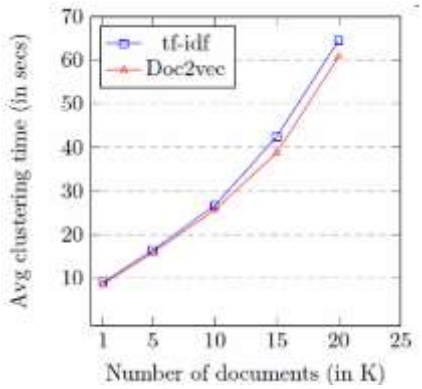
Slave nodes: 64 bit Intel Xeon CPU E5-2650 v2 @ 2.60GHz x 2 core processor with 32GB RAM.Disk size is 200GB.

**Dataset Description**

The Reuters-21578 dataset is used for training and testing. It is a widely used test collection for datacategorization. Reuters-21578 includes 22 .sgm files, which consists of news reports from 22 different sources. Each report consists of a set of information all enclosed in tags. It includes 'data,' 'time,' 'place,' 'category' of news, training/test data, 'title' and 'body' of the news report. Initially, these files are trimmed to include only the 'title' and 'body' of the news report.

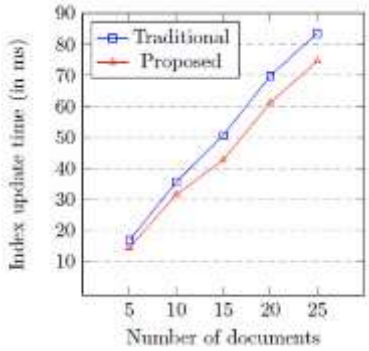
The dataset consists of 90 classes, 3019 testing documents, and 7769 training documents inthe ModApte subset of the Reuters-21578 benchmark. The average number of words per document, which are grouped by class, lies between 93 and 1263 in the training set. The count of unique wordsin the dataset after pre-processing is equal to 58573.

Figure 2. Clustering time analysis



Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, February : 2024

Figure 3. Index Update time for modified documents




**Clustering Analysis**

The pre-processed dataset is given for training the Doc2Vec model. The model is created with epoch(E)=6. The parameters set for the training model with four workers. The window size(I) givenis 5. The vocabulary built consist of 17228 items and 100 features. The model runs for six times, andthe words that are not frequently accessed are ignored at every iteration. Once the model is built, it is saved on disk, so that the clustering algorithm can access the data model to generate vectors and cluster the documents dynamically. The recent works on clustering with word embedding works onstatic data. Hence, this work has been compared with traditional TF-IDF in incremental scenarios. Figure 2 illustrates the comparison of the clustering time using Doc2Vec and TF-IDF. The clusteringtime calculated is the average value of the runs performed on the documents. The iterations valueis set to 10. The similarity measure used for clustering is Euclidean distance. The batch size given is 100. The number of random initializations is equal to 10. As shown in the graph, the Mini-batch K-means clustering using the Doc2Vec model of vector representation performs much better than the traditional TF-IDF. As the documents evolve, a subset of new random data records is used to findthe cluster center, which reduces the convergence time, and therefore the clustering time is reduced.The performance of Mini-batch K-means is better with the Doc2Vec model in the incremental update scenario also. Traditional models incur a substantial computational cost in clustering the documents due to their sparse representation of vectors. Hence, in this work, the pre-trained Doc2Vec is preferred to improve the performance of incremental clustering because of their fixed-sized dense representation of vectors.

**Index Construction and Updation**

This section involves the analysis of index updates for dynamic data. Index creation time includesthe time required to build the initial index for the whole system. Once the documents are clustered,they are distributed according to the clusters that they fall in. These documents are then indexed on index servers locally. The proposed method uses a B+ tree for indexing and gives better performance. In our work, the index update is one of the prominent functionalities which is added. Hence measuring the update time is essential. The update involves the addition/deletion of inverted indexterms based on the modifications present in the new document. Figure 3 shows the index updatetime for updating the index of modified documents and sending the document to the slave node.The documents which are taken as input are new documents and existing documents that have beenmodified. Each document has been modified by more than 75%. Both the traditional way of index rebuilds from scratch and the proposed approach have been compared. The conventional approach slows down the process of updates when there are lots of dynamic updates. Initially, the index rebuildcreates a new index for a document collection. When the document comes with the update, the new index is created with current data, and the new terms are also indexed. This new index substitutes the old index. If the same document is updated multiple times, the overhead of modifying the indexes is high, and hence the time taken is longer. Our approach only works on the changes made in thedocuments which reduce the number of edit operations, and therefore the time taken for an updateis reduced. With the increase in the number of documents, the difference in update time with thetraditional approach is increasing. This shows that our approach works well, even for a large number

of documents.

**Data Retrieval**

Data Retrieval time is dependent on the way indexing is done and how the top-k document retrievalhappens. Figure 4 shows the results for ranked document retrieval. It compares the traditional TF-

Figure 4. Data Retrieval for top-50 documents IDF scoring with our proposed method. The evaluations have been done for top-50 ranked documentretrieval. The time for processing the keyword queries includes pre-processing of the query terms, calculating the scores, and retrieval of documents to present to the user. The results support the factthat having BM25+ as the scoring mechanism improves the scoring over the traditional scoring mechanism. The conventional scoring mechanism depends on the occurrences of the query terms in a document without checking the relevance when compared with other documents in the whole collection while the BM25+ calculates the relevance of the document in the entire collection, and performance is improved for long documents. Hence, the query processing time is less in our systemcomparatively for any number of query terms.

**Precision, Recall and Mean Average Precision**

The effectiveness of our system is evaluated by the following metrics: Precision(P), Recall(R) and Mean Average Precision (MAP). The formula for the metrics is shown in equation 4 and 5.

$$P = \frac{a}{a+b} \tag{4}$$

$$R = \frac{a}{a+c} \tag{5}$$

where a, b, c represents true positives, false positives, and false negatives for the query. Average precision (AP) is the mean of precision values at all ranks wherever relevant documents are found.Figure 5 and Figure 6 show the average precision and the recall trend for top-k document retrieval. It has been performed for 25 query terms, and it is kept constant. From top-10 to top-100 documents, theprecision and recall have been computed for both the proposed system and the traditional approach. The average precision has been calculated for the first query. The ‘k’ value is fixed at 100, and averageprecision is estimated for every query. The results show that our system yields good precision and recall results, as shown in the results when compared with the traditional one. MAP is a primary measure of single number to compare search algorithms. It is calculated by taking the average of APvalues over a large set of queries. The formula of the MAP is given in equation 6 as follows.

Figure 5. Average Precision for top-k documents

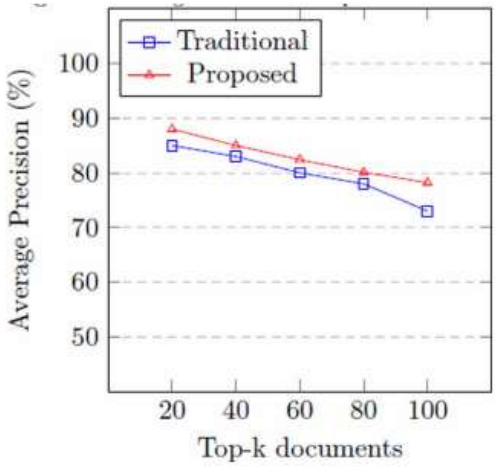



Figure 6. Recall for top-k documents

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 53, Issue 2, February : 2024

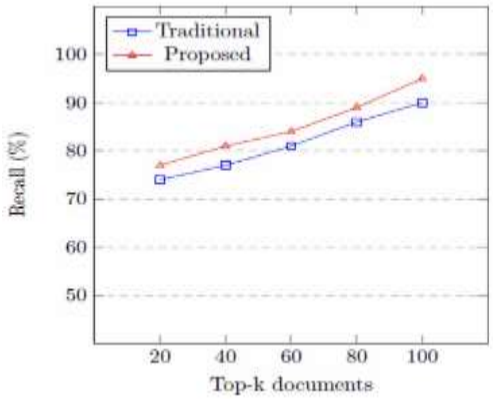



Figure 7. Mean Average Precision

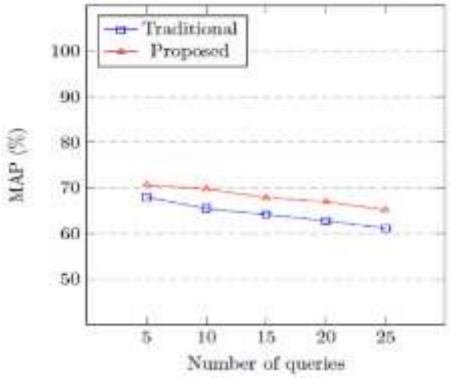



Figure 7 shows the MAP values for the different number of queries stating from 5 to 25 which gives the improved results compared to the traditional method.

## 6. CONCLUSION

The growth in data requires a more efficient way of organizing and processing dynamic unstructured data. A robust system is presented to improve the efficiency of data retrieval. Initially, the distributedvector representation model is used to generate the low dimensional document level vectors that enhance the performance of the clustering method. Later, the inverted index is built on the index servers; each connects with the slave nodes containing clustered documents. On-disk implementation of the B+ tree is used for building index, which is compared with the traditional method, gives betterperformance regarding index update. Finally, keyword query processing is done by using a probabilisticmodel BM25+, which helps in retrieving the most relevant data. On the whole, the efficient information retrieval system achieves better performance with all the steps mentioned above. Experimental analysis has shown that our system significantly improves the efficiency of clustering, indexing, and keyword query processing performance when comparedwith the traditional method.

## REFERENCES

1. Asadi, N., & Lin, J. (2013). Fast, incremental inverted indexing in main memory for web-scale collections. Audhkhasi, K., & Verma, A. (2007). Keyword search using modified minimum edit distance measure. In
2. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP*. IEEEPress.
3. Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEETransactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828. doi:10.1109/TPAMI.2013.50
4. Cai, X., Nie, F., & Huang, H. (2013). Multi-view k-means clustering on big data. In *Proceedings of the Twenty- Third International Joint Conference on Artificial Intelligence IJCAI '13* (pp. 2598-2604). AAAI Press.

5. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407. doi:10.1002/ (SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9
6. Feizollah, A., Anuar, N. B., Salleh, R., & Amalina, F. (2014). Comparative study of k-means and mini batch k-means clustering algorithms in android malware detection using network traffic analysis. In *Proceedings of the 2014 International Symposium on Biometrics and Security Technologies (ISBAST)* (pp. 193-197). Academic Press. 10.1109/ISBAST.2014.7013120
7. Fu, X., Liu, W., Xu, Y., & Cui, L. (2017). Combine hownet lexicon to train phrase recursive autoencoder for sentence-level sentiment analysis. *Neurocomputing*, *241*, 18–27. doi:10.1016/j.neucom.2017.01.079
8. Galambos, L. (2006). Dynamic inverted index maintenance. In Proceedings of World Academy Of Science, Engineering and Technology (Vol. 11, pp. 171-176). Academic Press.
9. Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the sixth New Zealand computer science research student conference (NZCSRSC2008)* (pp. 49-56). Academic Press.
10. Kim, H. K., Kim, H., & Cho, S. (2017). Bag-of-concepts: Comprehending document representation through clustering words in distributed representation. *Neurocomputing*, *266*, 336–352. doi:10.1016/j.neucom.2017.05.046
11. Kumar, K. M., & Reddy, A. R. M. (2017). An efficient k-means clustering filtering algorithm using density based initial cluster centers. *Information Sciences*, *418*, 286–301. doi:10.1016/j.ins.2017.07.036
12. Lauren, P., Qu, G., Zhang, F., & Lendasse, A. (2018). Discriminant document embeddings with an extreme learning machine for classifying clinical narratives. *Neurocomputing*, *277*, 129–138. doi:10.1016/j.neucom.2017.01.117
13. Liu, Y., Liu, B., Shan, L., & Wang, X. (2018). Modelling context with neural networks for recommending idioms in essay writing. *Neurocomputing*, 275.
14. Malik, B. H., Maryam, M., Khalid, M., Khlaid, J., Rehman, N. U., Sajjad, S. I., & Nasr, M. S. et al. (2019). Fastand Efficient In-Memory Big Data Processing. *Optimization*, *10*(5).
15. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543). Academic Press. 10.3115/v1/D14-1162
16. Ren, Y., Hu, K., Dai, X., Pan, L., Hoi, S. C., & Xu, Z. (2019). Semi-supervised deep embedded clustering. *Neurocomputing*, *325*, 121–130. doi:10.1016/j.neucom.2018.10.016