# EFFICIENT CLUSTERING OF SHORT TEXT STREAMS WITH AN APPLICATION TO FIND DUPLICATE QUESTIONS IN STACK OVERFLOW

by

Md Rashadul Hasan Rakib

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
November 2023

# Table of Contents

**List of Tables** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **iv**

**List of Figures** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **vi**

**Abstract** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **vii**

**Chapter 1 Introduction** . . . . . . . . . . . . . . . . . . . . . . **1**

**Chapter 2 Related Work** . . . . . . . . . . . . . . . . . . . . . . **5**

2.1 Static Clustering of Short Texts . . . . . . . . . . . . . . . . . . . . 5

2.2 Dynamic Clustering of Short Texts . . . . . . . . . . . . . . . . . . 7
2.2.1 Similarity-based Stream Clustering . . . . . . . . . . . . . . . . 8
2.2.2 Model-based Stream Clustering . . . . . . . . . . . . . . . . . . 9

2.3 Finding Duplicate Questions in Stack Overflow . . . . . . . . . . . . 11

**Chapter 3 Static Clustering of Texts** . . . . . . . . . . . . . . . . . **13**

3.1 Short Text Clustering by Similarity Matrix Sparsification . . . . . . . 13
3.1.1 Generating Similarity Matrix . . . . . . . . . . . . . . . . . . . 13
3.1.2 Sparsify Similarity Matrix . . . . . . . . . . . . . . . . . . . . . 14
3.1.3 Clustering Sparsified Matrix . . . . . . . . . . . . . . . . . . . . 16
3.1.4 Experimental Results of Short Text Clustering by Similarity Matrix Sparsification . . . . . . . . . . . . . . . . . . . . . . . 16

3.2 Short Text Clustering by Iterative Classification . . . . . . . . . . . . 20
3.2.1 Algorithm for Enhancement of Clusters by Iterative Classification 20
3.2.2 Stopping Criterion for Iterative Classification . . . . . . . . . . 22
3.2.3 Experimental Results of Short Text Clustering by Iterative Classification . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

**Chapter 4 Dynamic Clustering of Texts** . . . . . . . . . . . . . . . **28**

4.1 Short Text Stream Clustering via Frequent Word Pairs and Reassignment of Outliers to Clusters . . . . . . . . . . . . . . . . . . . . . . 28
4.1.1 Clustering Texts by Frequent *Word Pairs* . . . . . . . . . . . . 29
4.1.2 Building the Lexical Clustering Model . . . . . . . . . . . . . . 29
4.1.3 Clustering the Rest of the Texts . . . . . . . . . . . . . . . . . . 30
4.1.4 Removing Outliers in Clusters . . . . . . . . . . . . . . . . . . . 31
4.1.5 Building the Semantic Clustering Model . . . . . . . . . . . . . 31

4.1.6 Assigning Outliers to Clusters . . . . . . . . . . . . . . . . . . 32
4.1.7 Deleting Outdated Clusters . . . . . . . . . . . . . . . . . . . 32
4.1.8 Experimental Results of Short Text Stream Clustering via Frequent Word Pairs and Reassignment of Outliers to Clusters . . 33

4.2 Efficient Clustering of Short Text Streams using Online-Offline Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.2.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.2.2 Proposed Method . . . . . . . . . . . . . . . . . . . . . . . . 37
4.2.3 Experimental Results of Efficient Clustering of Short Text Streams using Online-Offline Clustering . . . . . . . . . . . . . . . . . 44

**Chapter 5 Finding Duplicate Questions using Clustering . . . . . 53**

5.1 Data Preparation for Finding Duplicate Questions . . . . . . . . . . . 53
5.1.1 Extracting Groups of Duplicate Questions . . . . . . . . . . . 54
5.1.2 Training and Test Set Generation . . . . . . . . . . . . . . . . 56

5.2 Finding Duplicate Question by Clustering . . . . . . . . . . . . . . . 57
5.2.1 Finding Close Clusters . . . . . . . . . . . . . . . . . . . . . . 58
5.2.2 Finding Duplicate Question using Clusters . . . . . . . . . . . 59

5.3 Experimental Study on Finding Duplicate Questions . . . . . . . . . 60
5.3.1 Statistics of Clustering Stack Overflow Questions in Training Set 60
5.3.2 Experimental Results on Finding Duplicate Questions . . . . . 61

**Chapter 6 Conclusion and Future Work . . . . . . . . . . . . . . . 65**

**Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68**

**Appendix A Impact of Similarity Matrix Sparsification on Hierarchical Clustering . . . . . . . . . . . . . . . . . . . . . . . . 74**

**Appendix B Justification of Similarity Threshold ($\mu+\sigma$) for Assigning Texts to Clusters . . . . . . . . . . . . . . . . . . . . . . 76**

# List of Tables

3.1 Summary of the short text datasets . . . . . . . . . . . . . . . 17

3.2 Accuracy and NMI for short text clustering. The top part is for hierarchical clustering using similarity matrix, with the best result for each combination of similarity and dataset bold. The bottom part is for state-of-the-art previous methods for short text clustering. The best overall result for each dataset is shaded. 18

3.3 Run-time in seconds of sparsification based clustering (using Glove embedding) and STC2-LE . . . . . . . . . . . . . . . . 19

3.4 ACC and NMI of different clustering methods, their corresponding enhancements by iterative classification, and state-of-the-art methods for short text clustering. $^{\Delta}$ indicates that this method is statistically significantly inferior to its corresponding enhancement obtained by iterative classification. * indicates that this method is statistically significantly inferior to HAC_$SD$_$IC$. . . 26

3.5 ACC results . . . . . . . . . . . . . . . . . . . . . . . . 26

3.6 NMI results . . . . . . . . . . . . . . . . . . . . . . . . 26

4.1 Summary of short text datasets . . . . . . . . . . . . . . . . . 33

4.2 NMI score of different clustering methods. * indicates that the proposed method is statistically significantly better than other methods on a particular dataset in terms of NMI. . . . . . . . 34

4.3 Average running times (in seconds) of the algorithms. . . . . . 34

4.4 Lexical Feature Representation of Text . . . . . . . . . . . . . 36

4.5 Notations used in the Algorithm of Online Text Stream Clustering 40

4.6 Normalized Mutual Information (NMI), Homogeneity (Ho.), V-Measure (VM) score of different clustering methods. * indicates that the EStream is statistically significantly better than other methods on a particular dataset in terms of NMI, Ho., or VM. The highest result for a particular dataset is denoted bold. . . 50

4.7 Average running times (in seconds) of different methods. . . . . 51

5.1 Summary of the StackOverflow Questions . . . . . . . . . . . 54

5.2 Statistics of the Groups of Duplicate Questions for Different Languages . . . . . 55

5.3 Sample Content (Tag, Title, and Body) of a Group of Duplicate Questions for Java . . . . . 56

5.4 Statistics of Training and Test Set . . . . . 56

5.5 Statistics of clustering questions in Training Set using Proposed Stream Clustering Method for Different Languages . . . . . 61

5.6 Experimental results of finding duplicate question for different languages by Our Method and Lucene . . . . . 62

A.1 Sample texts from small subset with indices and corresponding cluster topics for the dendrogram in Figure A.1 . . . . . 75

# List of Figures

4.1 Proposed short text stream clustering using Online-Offline clustering. . . . . 38

4.2 Sample StackOverflow question titles (with *PostId*) of a cluster in the dataset **SO-T**. . . . . 46

4.3 NMI results of our short text stream clustering method for different Update-intervals ($UI$) on dataset Ns-T. . . . . 46

5.1 Proposed duplicate question finding system using stream clustering. . . . . 57

A.1 Dendrograms for clustering similarity matrix . . . . . 74

B.1 Distribution of normally distributed similarity values . . . . . 77

B.2 Distribution of not normally distributed similarity values . . . 77

## Abstract

This thesis focuses on the efficient clustering of short texts along with an application to find duplicate questions in Stack Overflow. In the first part of this thesis, we discuss static and dynamic clustering methods for short text corpora. In the second part, we discuss how we can apply static and dynamic clustering of short texts to find duplicate questions in Stack Overflow.

Short text clustering is an important but challenging task due to the lack of context contained in short texts. In our first work, we overcome this problem by representing text using word embedding which allows us to capture similarity between texts sharing few or no common words. In addition, we investigate the impact of similarity matrix sparsification on the performance of short text clustering. In our second work, we improve the clustering result obtained from our first work by removing outliers from clusters and reclassifying them to proper clusters. This is repeated several times until the cluster partitions stabilize.

In our first and second work, we cluster static collections of short texts where the number of clusters to be produced is known. However, in real time, short texts are continuously being generated in large volumes from different sources. This motivates us to develop an efficient dynamic clustering method that creates or updates the clusters when text collection changes over time (e.g., new tweets arrive or new questions are posted on a question-answering site). In this method, we index clusters to reduce the number of similarity computations while assigning a text to a cluster. Using our dynamic clustering method along with static clustering, we cluster Stack Overflow questions as they arrive over time. Using the clusters of questions, we recommend potential duplicates of a newly posted question.

Experimental studies demonstrate that both our static and dynamic clustering methods of short texts perform better than that of the existing state-of-the-art methods in terms of clustering quality and running time on several short text datasets. We also demonstrate that by using the clusters obtained by our clustering method, we find more duplicate questions than an existing duplicate question finding system.

# Chapter 1

# Introduction

Short text clustering is a well known and popular research topic in the area of text mining. Due to technological advances, short texts are generated at large volumes from different sources such as micro-blogging, question-answering, and social news aggregation websites. Organizing these texts (e.g., grouping them by topic) is an essential step towards discovering trends (e.g., political, economic) in conversations and other data mining tasks, such as data summarization, frequent pattern analysis, and searching for and filtering information. Clustering texts into groups of similar texts is the foundation for many of these organizational strategies (Aggarwal and Zhai, 2012). What makes clustering (grouping) of short texts difficult is the much lower accuracy in identifying the topic of each text, from the few words it contains. The goal of our research is to develop novel methods for accurately and efficiently clustering collections of short texts; and efficiently updating the clusters, if text collection changes over time.

Due to the lack of context in short texts, traditional text similarities based on frequency or tf-idf scores is not adequate Xu et al. (2017). In particular, two short texts may share few common words or no words at all. In our first work Rakib et al. (2018), we alleviate this problem by using word embedding Pennington et al. (2014); Mikolov et al. (2013) based text representation. Using this representation, we generate similarity matrix containing similarity scores between texts. Then we sparsify the matrix using our proposed method based on similarity distribution ($SD$) as characterized by the mean and standard deviation of similarity values in each row. The objective of sparsification is to reduce noise in similarity matrix so as to improve clustering performance by keeping similarities between a text and its most similar (nearest) texts while discarding similarities with less similar ones Kumar (2000); Gollub and Stein (2010); Rakib et al. (2018). After performing sparsification, we cluster the sparsified matrix using hierarchical agglomerative clustering Müllner

(2013) as described in Section 3.1.

In our second work Rakib et al. (2020a), we improve the clustering result obtained from our first clustering method using iterative classification. The objective of this research is to improve the cohesion of clusters in a cluster partition produced by an arbitrary clustering method. To achieve this, we remove outliers from each cluster and reassign (i.e., reclassify) them to clusters with which they have greater similarity. This is repeated several times until the cluster partitions stabilize. We demonstrate that this approach produces more accurate cluster partitions than computationally more costly state-of-the-art short text clustering methods based on neural networks.

In our first and second work, we outperformed the state-of-the-art methods on clustering static collections of short texts where the number of texts to be clustered and number of clusters to be produced are known. In real world, texts are continuously being generated from different sources (e.g., such as Facebook, Twitter, question-answering sites, and so on). Organizing this growing number of texts becomes an inevitable task to solve various kinds of text mining problems. Motivated by this, we developed clustering methods that can adapt to the changes in the data over time, aiming for a substantial performance improvement in terms of accuracy and running time on the clustering of dynamic collections of short texts.

In our third paper Rakib et al. (2020b), we developed a dynamic clustering method based on a clustering method called MStream Yin et al. (2018) which clusters texts batch[1] by batch as they arrive. Let us briefly describe the main difference between our method and MStream. MStream assigns a text to a cluster based on the number of texts in clusters and the number of common words between the text and existing clusters. Therefore, the probability of a text choosing a large cluster is higher than choosing a small or new cluster. As a result, a text can be assigned to a large cluster with which it does not share any common words, that is, it should be removed as an outlier and should be reassigned to the appropriate cluster. Therefore, we remove outliers from the clusters and reassign them to appropriate clusters using semantic similarity as discussed in Section 4.1.

In our fourth work Rakib et al. (2021), we developed an efficient short text stream

[1] A Batch is defined as a collection of texts Yin et al. (2018). The terms "Batch" and "Stream" are interchangeably used in our work. "Text stream" and "dynamic collection of texts" are also interchangeably used in this work.

clustering method (called EStream) that outperformed our previous short text stream clustering method Rakib et al. (2020b) and other existing methods (e.g., MStream Yin et al. (2018), DP-BMM Chen et al. (2020)) in terms of running time and clustering accuracy. Prior to EStream, most of the existing short text stream clustering methods assign a text to a cluster by computing similarity between the text and all clusters. However, there may be some clusters with which a future text may not share any common feature, thus we can ignore similarity computations between that text and those clusters. Therefore, EStream assigns a text to a cluster by computing similarity between a text and a selected number of clusters instead of all clusters and thus significantly reduces the running time of clustering of short text streams. To improve clustering accuracy, EStream enhances the distributions of texts of the selected clusters using our clustering enhancement method based on iterative classification Rakib et al. (2020a).

In this thesis, we also show an application of our short text stream clustering method in the context of finding duplicate questions in Stack Overflow. Stack Overflow is a popular question-answering site where questions are continuously being posted on various programming problems. Despite detailed guidelines to prevent posting duplicate questions (i.e., questions that have already been answered), duplicate questions are frequently being posted Ahasanuzzaman et al. (2016). To handle this problem, Stack Overflow employs users with high reputations (called moderators) to detect duplicate questions, which is a labor-intensive job and may lead to some duplicate questions remaining undetected. An automatic duplicate detection system can alleviate this problem by recommending possible duplicates of a question Zhang et al. (2015b). Then, the moderators can focus on a smaller set of questions to find duplicate question for the given question.

Clustering can be a viable tool to provide a limited set of questions to the moderators by selecting questions only from the close clusters (in terms of similarity) of the given question Manning et al. (2008). Then the question arises, why we are using stream clustering to cluster Stack Overflow questions. The intuition is that questions are continuously being posted in Stack Overflow; as a result the number of questions to be clustered and number of clusters to be produced are unknown. Therefore by adopting stream clustering algorithm we can cluster the continuously growing number of questions since the stream clustering algorithm does not require neither the number of questions to be clustered nor the number of clusters to be produced in advance. Motivated by this we cluster Stack Overflow questions using our dynamic clustering method (i.e., EStream) to recommend potential duplicates of a newly posted question using the clusters of questions as described in Section 5.2.2.

The major contributions of this research are as follows.

- We developed two static clustering methods Rakib et al. (2018, 2020a) for short texts that outperformed state-of-the-art short text clustering methods in terms of clustering quality and running time.

- We developed two dynamic clustering methods for short text streams. The first one Rakib et al. (2020b) significantly outperforms the state-of-the-art short text stream clustering methods in terms of clustering quality. The second one Rakib et al. (2021) significantly outperforms the state-of-the-art methods in terms of running time on the datasets built on Stack Overflow questions.

- We have created four datasets for finding duplicate questions in Stack Overflow using four programming languages (which are R, C#, Python, and Java) comprising of 312,829, 1,304,920, 1,347,471 and 1,592,884 questions respectively.

- Experimental study demonstrates that by clustering Stack Overflow questions using our dynamic clustering method along with static clustering, we can find more duplicate questions than an existing duplicate finding system.

The remainder of our research is organized as follows. We briefly describe the work related to our study in chapter 2. The detailed methodology of our static and dynamic clustering methods of texts and experimental results are described in chapter 3 and chapter 4 respectively. In chapter 5, we discuss how we apply our clustering method to find duplicate question as well as the experimental results of finding duplicate questions. Finally we conclude our research and discuss future related work in chapter 6.

# Chapter 2

# Related Work

In this chapter, we discuss related work about static and dynamic clustering of short texts and finding duplicate questions in Stack Overflow.

## 2.1 Static Clustering of Short Texts

A major challenge in short text clustering is the sparseness of the vector representations of these texts resulting from the small number of words in each text. Several clustering methods have been proposed in the literature to address this challenge, including methods based on text augmentation Banerjee et al. (2007); Zheng et al. (2018), neural networks Xu et al. (2017); Hadifar et al. (2019), topic modeling Cheng et al. (2014), Dirichlet mixture model Yin and Wang (2016), and similarity matrix sparsification Kumar (2000); Gollub and Stein (2010); Rakib et al. (2018).

A recent method based on text augmentation Zheng et al. (2018) uses topic diffusion to augment each short text by finding words not appearing in the text that are related to its content. To find related words, this method determines possible topics for each text using the existing words. Then new words are added to each text; these new words are closely related to the text's topics based on the posterior probabilities of the new words given the words in the text. An earlier text augmentation method Banerjee et al. (2007) finds Wikipedia articles using the short text as query string and uses the articles' titles as features.

A short text clustering method based on word embedding and a convolutional neural network called STC2-LE was proposed in Xu et al. (2017). It uses a convolutional neural network to learn a text representation on which clustering is performed. Another short text clustering method based on weighted word embedding and autoencoder was proposed in Hadifar et al. (2019). For each text, it calculates the average of the weighted embeddings Mikolov et al. (2013) of its words. The weight of a word is calculated based on its inverse frequency in the corpus Hadifar et al. (2019)

which is then multiplied with its embedding to obtain weighted word embedding. After that, the embeddings of the texts are feed into an autoencoder to obtain the low dimensional representation of the texts on which clustering is performed.

A method based on locality-sensitive term weighting is introduced in Zheng et al. (2017). Distances between texts are calculated using weights of terms (e.g., words); the weights are obtained based on the property that, similar terms are tight together in terms of their own locality and well-separated from other localities.

Biterm topic modeling (BTM) Cheng et al. (2014) is a topic modeling approach for short texts that learns topics from word co-occurrence patterns (i.e., biterms). Given a topic distribution produced by BTM for each text, clustering is performed by assigning a text to its most probable topic.

Several short text clustering methods were proposed in the literature based on similarity matrix Sparsification Gollub and Stein (2010) such as global threshold Kumar (2000), nearest neighbors Kumar (2000), and center vectors Gollub and Stein (2010). Sparsification of the text similarity matrix keeps the association between a text and its most similar (nearest) texts while breaking associations with less similar ones by setting the corresponding similarity scores to 0.

Similarity matrix sparsification based on global threshold is the simplest sparsification method. It removes all similarity values that are below a given threshold Kumar (2000). The problem with this method is that some real clusters may be destroyed or merged because different clusters may have different similarity levels between the texts they contain. For example, the range of the similarity values between the texts in one cluster may be between 0.2 and 0.4 while the similarity values in another cluster may range from 0.5 to 0.8. If we set the global threshold to 0.5, then the similarity values in the first cluster are set to 0 and the cluster is destroyed. A lower threshold, such as 0.15, may result in the inclusion of additional documents in the second cluster.

Nearest neighbors' based methods for similarity matrix sparsification include $k$-nearest neighbor Kumar (2000) and shared nearest neighbor Kanj et al. (2016). $k$-nearest neighbor sparsification keeps only the $k$ highest similarity scores for each text; the shared-nearest neighbor approach adds a condition that texts retaining similarity values with a particular text should share a prescribed number of neighbors. The $k$-nearest neighbors sparsification strongly assumes that each cluster contains the same number of texts. Therefore this sparsification method may not produce expected clustering result for imbalanced dataset (i.e., some clusters contain large number of texts and some contain much less).

A similarity matrix sparsification method based on the center vector was proposed in Gollub and Stein (2010). Texts are represented by *tf-idf* (term frequency-inverse document frequency) vectors and a center vector is computed by averaging these vectors. The sparsification of the similarity matrix is performed by removing similarities between all pairs of texts that are not more similar to each other than the maximum similarities of these two texts to the center vector.

In our first paper on short text clustering Rakib et al. (2018), we sparsify similarity matrix using the similarity thresholds computed based on similarity distribution. In particular, the similarity scores we keep for each text are determined based on a dynamically computed similarity threshold for that text. Therefore we do not require user defined similarity threshold to retain similarity scores in the matrix. In addition, in similarity distribution based sparsification, we do not need to strongly assume that each cluster contains approximately the same number of texts as the $k$-nearest neighbors. Thus similarity distribution based sparsification helps us to cluster imbalanced dataset more accurately than $k$-nearest neighbor sparsification.

In our second paper on short text clustering Rakib et al. (2020a), we improve the clustering result obtained from our first clustering method by removing outliers from the clusters and reassign them to proper cluster using iterative classification. We demonstrate that this approach produces more accurate cluster partitions than computationally expensive state-of-the-art short text clustering methods based on neural networks.

## 2.2 Dynamic Clustering of Short Texts

A detailed survey of dynamic clustering of texts can be found in Mahdiraji (2009); Silva et al. (2013); Nguyen et al. (2015); Aggarwal and Reddy (2013); Carnein and Trautmann (2019). We can categorize the methods of dynamic clustering of texts (i.e., text stream clustering) into two categories which are similarity-based stream clustering and model-based stream clustering.

### 2.2.1 Similarity-based Stream Clustering

In general, similarity-based text stream clustering methods use the vector space model Erk (2012) to represent the documents. A document is assigned to a new or one of the existing clusters based on the similarity threshold which needs to be manually determined by the user Yin et al. (2018).

CluStream Aggarwal et al. (2003) is a stream clustering method consisting of an online micro-clustering phase and an offline macro-clustering phase. In online phase, it assigns a data point to a new or an existing micro-cluster based on a manually determined similarity threshold. In offline phase, it applies $k$-means clustering on the micro-clusters and obtain $k$ user specified macro-clusters. Similar to CluStream our method EStream consists of online and offline clustering phases. However, in the online phase of EStream, we use the dynamically computed similarity threshold (in contrast to manually determined similarity threshold) for each text to assign it to a cluster; and in the offline phase we enhance the distributions of the $k$ number of clusters where the value of $k$ is dynamically determined.

FuzzStream De Abreu Lopes and De Arruda Camargo (2017), a fuzzy data stream clustering method consists of online and offline clustering phases. In the online phase FuzzStream produces micro-clusters using a fuzzy set of examples from data stream. In the offline phase it applies Weighted Fuzzy C-Means clustering algorithm Li et al. (2018) to the micro-clusters to obtain a set of macro-clusters based on a user defined similarity threshold.

Sumblr Shou et al. (2013) is a tweet stream summarization prototype. It consists of a text stream clustering module that compresses tweets into tweet feature vectors (TCVs) and assigns future tweets to the clusters based on the statistics of TCVs.

An efficient text stream clustering method using term burst information was proposed in Kalogeratos et al. (2016). Bursty terms are the terms that appear in many documents during a short period of time. This approach considers the fact that the documents that are published on a particular topic within a certain time period contain a particular set of bursty terms. An user-defined threshold for the number of occurrences of terms in documents is used to identify bursty terms.

In general, similarity-based text stream clustering methods use user defined similarity threshold to assign a text to a new or to one of the existing clusters Yin et al. (2018). On the contrary, we dynamically calculate the similarity threshold Rakib et al. (2020b) for each text based on statistical measure and use this similarity threshold to assign the text to a new or to one of the existing clusters. Thus our method efficiently handles the *concept drift* problem Zhang et al. (2017b) (the problem that topics of the text streams may change over time).

### 2.2.2 Model-based Stream Clustering

Several model-based text stream clustering methods were proposed based on multinomial mixture model Yin et al. (2018); Chen et al. (2020); Kumar et al. (2020). Generally, these algorithms use Gibbs sampling Ishwaran and James (2001) to estimate the parameters of the mixture model so as to obtain the clustering of text streams Yin et al. (2018).

A recent short text stream clustering algorithm based on Dirichlet process multinomial mixture model was proposed in Yin et al. (2018) which uses two Dirichlet priors $\alpha$ and $\beta$. $\alpha$ refers to the prior probability of a text choosing a new cluster and $\beta$ corresponds to the prior probability of a text choosing a cluster with which the text shares more similar content than other clusters. This algorithm has two variants: one is by retaining all previous clusters (called MStream) and other one is by removing old clusters (called MStreamF).

A biterm based mixture model for short text stream clustering was proposed in Chen et al. (2020). Similar to MStream(F) algorithm Yin et al. (2018), the biterm based clustering method developed two variants: one is by retaining the clusters obtained in previous batches (called DP-BMM) and other is by discarding the clusters obtained in previous batches (called DP-BMM-FP). The main difference between MStream(F) and DP-BMM(-FP) is that DP-BMM(-FP) represents the texts using biterm features instead of unigrams. In particular, DP-BMM(-FP) represents a text of $n$ words using $n * (n - 1)/2$ biterm features.

OSDM Kumar et al. (2020) is a semantic-enhanced Dirichlet model for short text stream clustering. OSDM extends the MStream Yin et al. (2018) algorithm by integrating the word to word co-occurrence based semantic information obtained from the common words between a text and a cluster and uses this semantic information to compute similarity between a text and a cluster.

DCT-L Liang et al. (2016) is a dynamic clustering topic model for short text streams based on Dirichlet process multinomial mixture model. It assigns a single topic (i.e., cluster) to each short text at a particular timestamp and uses the resulting topic distribution as priors for inferring the topics of subsequent documents.

DTM Blei and Lafferty (2006) is a dynamic topic model that analyzes the topics of a collection of documents over time. This method assumes that a document is rich enough to contain multiple topics. However, this assumption does not work well for short texts, which results low quality performance on short text streams.

The Dirichlet process mixture model based clustering algorithms (e.g., MStream(F), DP-BMM(-FP)) assign a text to a new or an existing cluster based on two factors: the number of texts in each cluster and the number of common words between the text and the texts already in the clusters. Therefore, the probability of a text choosing a large cluster is higher than choosing a small or new cluster. As a result, a text can be assigned to a large cluster with which it does not share any common words. In this case, the text may be more meaningfully assigned to a different cluster, that is, it should be removed as an outlier and should be reassigned to the appropriate cluster. To alleviate this problem, in our first short text stream clustering method Rakib et al. (2020b), we remove outliers from the clusters and reassign them to appropriate clusters using the semantic similarity between the outliers and clusters.

The Dirichlet process mixture model based clustering algorithms require tuning the parameters (i.e., $\alpha$ and $\beta$) to obtain the desired clustering performance. For example, MStream(F) uses $\alpha = 0.03$ and $\beta = 0.03$ to obtain optimal clustering performance. The DP-BMM(-FP) and OSDM performed grid search to obtain the optimal values for $\alpha$ and $\beta$ and set ($\alpha = 0.6$ and $\beta = 0.02$) and ($\alpha = 2e^{-3}$ and $\beta = 4e^{-5}$) respectively. On the contrary, our second short text stream clustering method (called EStream) Rakib et al. (2021) does not require this kind of parameter tuning, instead it uses the dynamically computed similarity threshold Rakib et al. (2020b) (based on statistical measure) to assign a text to a new or an existing cluster.

The Dirichlet process mixture model based clustering algorithms assign a text to a cluster by computing similarities between the text and the existing clusters based on the common features (i.e., word, bigram, biterm (defined in Section 4.2.1)). However, there may be some clusters with which a text may not share any common feature, thus the similarities between that text and those clusters are zero. Hence, those similarity computations can be ignored. Motivated by this, the EStream algorithm adopts inverted index based searching technique Ilic et al. (2014) and selects a specific set of clusters for a text that share common features with that text. Then EStream algorithm computes similarity between the text and the selected clusters to assign that text to a cluster. By limiting the number of similarity computations, we significantly reduce the running time of our method and thus the running time of our method is several orders of magnitude faster than that of the state-of-the-art methods.

## 2.3 Finding Duplicate Questions in Stack Overflow

Several studies have been performed to find duplicate questions in Stack Overflow which are of mainly two kinds: unsupervised and supervised.

DupPredictor Zhang et al. (2015b) is an unsupervised duplicate question detection system that identifies potential duplicates of a given question by considering the title, body and tag similarity of the questions. It computes similarity between the given question and all the other questions to search the potential duplicates of the given question. In contrast to computing similarity between the given question and all the other questions, our proposed system computes similarity between the given question and the representations of the clusters which helps us to perform limited number of similarity computations.

CROKAGE da Silva et al. (2020) is a relevant solution finding system from Stack Overflow. Given a programming task as a query to CROKAGE, it finds relevant code examples along with description for that programming task. The searching for relevant code examples by CROKAGE is modeled as an Information Retrieval (IR) problem da Silva et al. (2020). To search relevant code examples, CROKAGE indexed solutions by an inverted index Ilic et al. (2014) based search engine (called Lucene). Then, for a given query, it retrieves top ranked solutions using the index created by Lucene. After that, it applies word embedding based semantic similarity between the query and top ranked solutions so as to retrieve the ultimate desired solutions.

Dupe Ahasanuzzaman et al. (2016) is a supervised duplicate question detection system that finds potential duplicates of a given question by detecting the top ranked duplicate question pairs where a pair consists of the given question, the probable duplicate question and the level of duplication. Dupe is trained using a number of duplicate and non-duplicate question pairs. Based on the trained model it detects the possible duplicate question pairs of the given question.

PCQADup Zhang et al. (2017a) is another supervised duplicate question detection system based on word embedding and topic modeling. By using word embedding and topic modeling it captures the semantic relationship between two questions that helps to efficiently detect whether two questions are duplicate or not.

Another supervised duplicate question detection system based on deep learning model was proposed in Wang et al. (2020). This method uses Word2Vec representation for the words in question. It explores various kinds of deep learning models (e.g., CNN, RNN, LSTM) and other generalized machine learning models (e.g., Support Vector Machine, Logic Regression, Random Forest and eXtreme Gradient Boosting) along with Word2Vec representation to learn semantic relationship between questions. Based on the experimental results it shows that deep learning based models can detect more duplicate questions than the generalized machine learning models.

Most of the duplicate question detection systems in literature are supervised which require manually labelled duplicate questions to predict the future duplicate questions. On the contrary, our duplication question finding system is unsupervised. In addition, our method can adapt to continuously growing number of questions over time and can help to find duplicate questions by searching a limited set of questions selected from the close clusters of the given question.

# Chapter 3

# Static Clustering of Texts

In this chapter, we discuss the proposed methods of static clustering of texts and the experimental results obtained from these methods. Our first method is based on similarity matrix sparsification. Our second method is based on iterative classification.

## 3.1 Short Text Clustering by Similarity Matrix Sparsification

In this work Rakib et al. (2018), we cluster a collection of texts into a given number of clusters. At first we generate a similarity matrix for all text pairs, then we sparsify the similarity matrix. After that we cluster the sparsified matrix by hierarchical agglomerative clustering Müllner (2013).

### 3.1.1 Generating Similarity Matrix

We generate similarity matrix by computing similarity between all pairs of short texts. To compute similarity between the texts we use different types of text similarity measures such as word embedding, word co-occurrence, and Tf-Idf based. Word Embedding based similarity is the cosine similarity for text vectors, which are averages of embeddings of words of a given text. We use three types of pretrained word embeddings: Glove embeddings[1] trained by Glove method Pennington et al. (2014) on Wikipedia dumps, Word2Vec embeddings[2] trained by Word2Vec method Mikolov et al. (2013) on Google News, and BioASQ embeddings[3] trained by Word2Vec method on the abstracts of biomedical publications. BioASQ similarity is applied only to BioMedical dataset. GTM Islam et al. (2012) is a text similarity based on a word pair similarity that utilizes information of co-occurrence of the given two words in Google tri-gram corpus Brants and Franz (2006). Tf-Idf similarity is cosine similarity

[1]http://nlp.stanford.edu/data/glove.42B.300d.zip
[2]https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
[3]bioasq.lip6.fr/tools/BioASQword2vec/

for text vectors in the bag-of-words vector space model with term frequency-inverse document frequency weights.

### 3.1.2 Sparsify Similarity Matrix

Sparsification of text similarity matrix keeps association between a text and its most similar (nearest) texts, while breaking associations with less similar ones Kumar (2000). The major objective of sparsification in the context of clustering is to reduce noise in similarity matrix so as to improve clustering quality Kumar (2000); Gollub and Stein (2010).

We investigate the potential of various methods of sparsification (e.g., Similarity Distribution based sparsification ($SD$) Rakib et al. (2018), k-Nearest Neighbors sparsification ($k$-NN) Kumar (2000), and Center based sparsification Gollub and Stein (2010)) for improving short text clustering. A square $n \times n$ symmetric similarity matrix $S = (s_{ij})$ is an input to each sparsification method ($n$ is the number of texts, $s_{ij}$ is the similarity score between texts $t_i$ and $t_j$). Some methods require additional inputs or parameters. Each sparsification method listed below retains some original values of similarities, while replacing remaining ones by zeros (self-similarities on the diagonal are always retained). A sparsification criterion may render a matrix not symmetric. Such a matrix requires symmetrization: we follow the "sparsification with exclusion" Kumar (2000) approach, namely in the final matrix an element $s_{ij}$ is set to zero only if the sparsification criterion retains neither $s_{ij}$ nor $s_{ji}$. In the following we discuss the three similarity matrix sparsification techniques.

#### Similarity Distribution based Sparsification

***Definition*** The Similarity Distribution based ($SD$) sparsification is a new sparsification method that we propose. In contrast to the $k$-nearest neighbors method, the number of similarities to keep for each text is not fixed, instead it is based on the distribution of the similarity values between the text and all other texts. The parameter of the method, called $l$, is the average number of retained similarities with other texts per text (i.e., the average number of non-zero matrix elements outside of the diagonal per row) in the final, symmetric sparsified matrix.

For each text $t_i$, we calculate mean $u_i$ and standard deviation $\delta_i$ of similarities between $t_i$ and all other texts, and we sparsify similarities between $t_i$ and other texts based on these statistics. In particular, we define the retaining criterion as follows: a similarity $s_{ij}$ is to be retained if and only if

$$s_{ij} > u_i + \alpha\delta_i, \tag{3.1}$$

for some global factor $\alpha$, otherwise it is to be replaced by zero.

Factor $\alpha$ is such that after applying the criterion and symmetrization of the matrix, the average number of non-zero elements outside of the diagonal per row is equal to the value of parameter $l$.

***Algorithm*** We design an algorithm to perform the $SD$ sparsification through a search for an exact value of factor $\alpha$.

For each similarity value $s_{ij}$ in matrix $S$, we use an auxiliary value $a_{ij} = \frac{s_{ij}-u_i}{\delta_i}$. Using this notation, our criterion from Eq. 3.1 can be stated as follows: a similarity $s_{ij}$ is to be retained if and only if $a_{ij} > \alpha$. Since we follow "sparsification with exclusion" approach for symmetrization, we will keep $s_{ij}$ in the final symmetric matrix if the retaining criterion is fulfilled either for $s_{ij}$ or for $s_{ji}$ (or for both). It follows that exactly when $max(a_{ij}, a_{ji}) > \alpha$, then both $s_{ij}$ and $s_{ji}$ are retained in the final sparsified matrix.

Let us also notice that the condition of the average number of non-zero elements outside of the diagonal per row in the final, symmetric matrix is to be $l$, is equivalent to the condition that the total number of non-zero elements above the diagonal is $\lfloor \frac{n\times l}{2} \rfloor$.

Based on these observations, it can be seen that the following algorithm performs the $SD$ sparsification. Auxiliary values $a_{ij}$ are calculated for all elements in $S$. Then for each matrix element above the diagonal, i.e., for each pair of indices $(i, j)$ such that $j > i$, the maximum of $a_{ij}$ and $a_{ji}$ is found, and stored in a list $L$; each value in $L$ is associated with the corresponding pair of indices $(i, j)$. List $L$ is sorted and $\lfloor \frac{n\times l}{2} \rfloor$ largest values from $L$ are retrieved. For each pair of indices $(i, j)$ associated with the retrieved values, both $s_{ij}$ and $s_{ji}$ are retained in the final matrix; the remaining elements outside of the diagonal are set to 0. If factor $\alpha$ is needed explicitly, a value that is less than the smallest retrieved value from list $L$, and greater than or equal to the largest not retrieved value from list $L$, is a correct value of $\alpha$.

#### k-Nearest Neighbors sparsification

The $k$-nearest neighbors (k-NN) method uses the number of nearest neighbors $k$ as a parameter. The method criterion is to retain, for each text, exactly $k$ highest similarities with this text outside of the diagonal. After this criterion is applied, symmetrization is required.

#### Center based sparsification

The *Center* method ($\bar{\phi}$) and the *Modified Center* method ($\hat{\phi}$) Gollub and Stein (2010) are not parameterized. The Center method uses the mean vector $c$ of all text vectors. A similarity between two texts $t_i$, $t_j$ is retained if and only if it is higher than the maximum of the similarities between $t_i$, $c$, and $t_j$, $c$. The Modified Center method uses a modified version of vector $c$. No additional symmetrization is needed.

### 3.1.3 Clustering Sparsified Matrix

We use hierarchical agglomerative clustering to cluster sparsified matrix. The clustering method starts with each document in its own cluster and repeatedly merges pairs of most similar clusters until only $k$ (the desired numbers of clusters) clusters remain. To perform hierarchical clustering we use Ward criterion using fastcluster implementation Müllner (2013).

### 3.1.4 Experimental Results of Short Text Clustering by Similarity Matrix Sparsification

In this section, we discuss the datasets and experimental results obtained by our short text clustering method based on similarity matrix sparsification Rakib et al. (2018).

#### Datasets

We used five different datasets of short texts in our experiments. The basic properties of these datasets are shown in Table 3.1. **SearchSnippet** is a dataset of search results from Google's search engine, containing 12340 snippets distributed into 8 groups Xu et al. (2017). **SearchSnippet-test** is a subset of the SearchSnippet dataset consisting of 2280 search snippets distributed into 8 groups. **AgNews** is a subset of a dataset

Table 3.1: Summary of the short text datasets

| Dataset | #Clusters | #Texts | Average #words/text |
|---|---|---|---|
| SearchSnippet | 8 | 12340 | 17.03 |
| SearchSnippet-test | 8 | 2280 | 17.18 |
| AgNews | 4 | 8000 | 22.61 |
| StackOverflow | 20 | 20000 | 8.23 |
| BioMedical | 20 | 20000 | 12.88 |

of news titles Zhang et al. (2015a). It consists of 8000 texts in 4 topic categories (for each category, we randomly selected 2000 texts). **StackOverflow** is a subset of the challenge data published on Kaggle[4], where 20000 question titles from 20 groups were randomly selected Xu et al. (2017). **BioMedical** is a subset of the challenge data published on the BioASQ's website[5], where 20000 paper titles from 20 groups were randomly selected Xu et al. (2017).

### Evaluation Metrics

We evaluate whether considered sparsification methods improve clustering results, using accuracy (ACC) and normalized mutual information (NMI) as evaluation measures (as in Xu et al. (2017)).

Let $c_i$, $t_i$ be the cluster label and human annotated label of a text $t_i$ respectively, then clustering accuracy is defined in Equation 3.2 as follows:

$$ACC = \frac{\sum_{i=1}^{n} \delta(t_i, map(c_i))}{n} \tag{3.2}$$

where, $n$ is the total number of texts, $map(c_i)$ is the mapping function that maps each cluster label $c_i$ to the equivalent human annotated label, and $\delta$ is the indicator function that equals to one if both values are same otherwise equals to zero.

Normalized mutual information Chen et al. (2011) between human annotated label set T and cluster label set C is defined in Equation 3.3 as follows:

$$NMI(T, C) = \frac{MI(T, C)}{\sqrt{H(T)H(C)}} \tag{3.3}$$

where, $MI(T, C)$ is the mutual information between $T$ and $C$, $H(\cdot)$ is entropy and the denominator $\sqrt{H(T)H(C)}$ is used for normalizing the mutual information to be in the range of 0 to 1.

[4]https://www.kaggle.com/c/predict-closed-questions-on-stack-overflow/download/train.zip
[5]http://participants-area.bioasq.org/

### Impact of Similarity Matrix Sparsification

We consider a sparsification method to improve clustering for a given combination of dataset and similarity, if both evaluation measures increase after the method is applied to the similarity matrix. Results of the sparsification methods are in the top part of Table 3.2. The bottom part of Table 3.2 contains the results of state-of-the-art short short text clustering methods.

Table 3.2: Accuracy and NMI for short text clustering. The top part is for hierarchical clustering using similarity matrix, with the best result for each combination of similarity and dataset bold. The bottom part is for state-of-the-art previous methods for short text clustering. The best overall result for each dataset is shaded.

| Method | | | SearchSnippets-test | | SearchSnippets | | StackOverflow | | AgNews | | BioMedical | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Similarity | Sparsification | ACC | NMI | ACC | NMI | ACC | NMI | ACC | NMI | ACC | NMI |
| Hierarchical Agglomerative Clustering | Glove | Without | 0.770 | 0.709 | 0.765 | 0.594 | 0.616 | 0.544 | 0.765 | 0.528 | 0.353 | 0.290 |
| | | *SD* | **0.894** | **0.787** | **0.826** | **0.637** | **0.648** | **0.594** | **0.818** | **0.545** | 0.369 | 0.307 |
| | | *k*-NN | 0.871 | 0.764 | 0.790 | 0.605 | 0.581 | 0.540 | 0.768 | 0.524 | **0.396** | **0.315** |
| | Word2Vec | Without | 0.665 | 0.620 | 0.644 | 0.472 | 0.299 | 0.258 | 0.742 | 0.457 | 0.238 | 0.173 |
| | | *SD* | **0.809** | **0.686** | **0.761** | **0.549** | 0.340 | **0.287** | **0.788** | **0.525** | 0.263 | 0.209 |
| | | *k*-NN | 0.753 | 0.661 | 0.720 | 0.506 | **0.347** | 0.279 | 0.784 | 0.493 | **0.280** | **0.212** |
| | Tf-Idf | Without | 0.271 | 0.268 | 0.420 | 0.298 | 0.373 | 0.290 | 0.438 | 0.237 | 0.267 | **0.243** |
| | | *SD* | **0.476** | **0.434** | **0.518** | **0.361** | **0.438** | **0.319** | **0.662** | **0.371** | **0.304** | 0.241 |
| | | *k*-NN | 0.444 | 0.375 | 0.486 | 0.340 | 0.419 | 0.291 | 0.604 | 0.347 | 0.294 | 0.226 |
| | GTM | Without | 0.784 | 0.685 | 0.712 | 0.488 | **0.633** | 0.586 | 0.557 | 0.324 | 0.351 | 0.292 |
| | | *SD* | **0.815** | **0.693** | **0.713** | **0.490** | 0.628 | **0.594** | **0.719** | **0.384** | **0.375** | **0.304** |
| | | *k*-NN | 0.806 | 0.680 | 0.683 | 0.455 | 0.626 | 0.560 | 0.608 | 0.373 | 0.358 | 0.293 |
| | BioASQ | Without | | | | | | | | | 0.388 | 0.310 |
| | | *SD* | | | | | | | | | **0.401** | **0.335** |
| | | *k*-NN | | | | | | | | | 0.397 | 0.321 |
| state-of-the-art | STC2-LE | | | | 0.770 | 0.631 | 0.511 | 0.490 | | | 0.436 | 0.380 |
| | Topic Diffusion | | 0.900 | 0.500 | | | | | | | | |
| | Loc.-Sens. Term Weight. | | 0.875 | 0.582 | | | | | | | | |
| | BTM | | 0.726 | 0.590 | 0.508 | 0.352 | 0.335 | 0.244 | 0.536 | 0.311 | 0.288 | 0.209 |

The best result (ACC, NMI) for each combination of dataset and similarity is denoted bold. We can observe that the *SD* method and the k-NN method improve clustering performance in most cases (in all except for 2 and 6 dataset/similarity combinations, respectively). For datasets other than BioMedical, the best hierarchical clustering results are obtained with the *SD* sparsification on Glove similarity. For BioMedical data, the best hierarchical clustering results are achieved also with the *SD* sparsification and BioASQ similarity. We further investigated why we obtain better result by clustering a sparsified matrix than by clustering a dense matrix using hierarchical agglomerative clustering as discussed in Appendix A.

**Comparison with State-of-the-Art Methods**

Table 3.2 (the bottom part) shows clustering performance (ACC and NMI) by four state-of-the-art methods for short text clustering, described in Section 2.1: STC2-LE Xu et al. (2017), Corpus-based Topic Diffusion Zheng et al. (2018), Locality-Sensitive Term Weighting Zheng et al. (2017) and BTM Cheng et al. (2014). For BTM, we perform experiments ourselves, and we report the best results over parameter search for two parameters of the method, $\alpha$ and $\beta$. For the other three state-of-the-art methods, we list previously published results Xu et al. (2017); Zheng et al. (2018, 2017) for datasets, for which they are available, because an implementation is either unavailable or (for STC2-LE) it can not be run on a new set.

For datasets other than BioMedical, results by the $SD$ and k-NN methods with Glove similarity are competitive with state-of-the-art methods. Namely, the $SD$ method with Glove similarity produces the best NMI on these sets, and it produces ACC that is best on three sets (on SearchSnippets-test it is slightly lower than that by the Topic Diffusion method). The results of the k-NN method with Glove similarity are on those sets also usually higher than those of state-of-the-art methods (the exceptions are ACC on SearchSnippets-test and NMI on SearchSnippets).

For BioMedical data, the $SD$ and k-NN sparsification methods perform best with BioASQ similarity. With respect to both ACC and NMI, they are outperformed on this set by STC2-LE.

We compare the run-time of clustering using $SD$ and k-NN sparsification with that of STC2-LE. The runt-time of clustering of a sparsification based method is computed by the run-time of sparsification of similarity matrix plus the run-time clustering the sparse matrix. The sparsification based methods are much faster than STC2-LE by several orders of magnitude, as shown in Table 3.3.

Table 3.3: Run-time in seconds of sparsification based clustering (using Glove embedding) and STC2-LE

| Method | SearchSnippets | StackOverflow | BioMedical |
|---|---|---|---|
| $SD$ | 137 | 648 | 613 |
| $k$-NN | 54 | 67 | 63 |
| STC2-LE | 8277 | 8401 | 8594 |

## 3.2 Short Text Clustering by Iterative Classification

In this work Rakib et al. (2020a), we improve the clustering result obtained by our similarity matrix sparsification based method Rakib et al. (2018) by iterative classification[6]. Given a collection of short texts and a partition of these texts into clusters, iterative classification modifies the given cluster partition by detecting outliers in each cluster and changing the clusters to which they are assigned. This is repeated several times, hence the term iterative in the method's name.

In each iteration, we generate training and test sets containing non-outliers and outliers respectively. Then we train a classification algorithm using the training set and classify the test set using the trained model. This iterative process repeats until the stopping criterion discussed in Section 3.2.2 is satisfied. The details are shown in Algorithm 1 and are described next.

### 3.2.1 Algorithm for Enhancement of Clusters by Iterative Classification

In each iteration, we choose a number $P$ that roughly corresponds to the fraction of texts selected for the training set. $P$ is chosen uniformly at random from an interval $[P_1, P_2]$ determined in Section 3.2.3. To generate the training set, we remove outliers from each of the $K$ clusters defined by the current cluster labels $L$. To remove outliers, we use an outlier detection algorithm called Isolation Forest Liu et al. (2008), which is applied to the *tf-idf* vector representations of the texts. The algorithm isolates the texts that exist in the low density region of the *tf-idf* feature space. If after removing outliers, a cluster contains more than $\frac{n}{K} \times P$ texts, then we remove texts from that cluster uniformly at random to reduce the number of texts in the cluster to $\frac{n}{K} \times P$. The reason of removing texts from each cluster is that we want each cluster to consist of roughly the same number of texts so as to reduce the bias of the classification algorithm. We add the removed texts to the test set and add the other texts to the training set. We train a classifier (i.e., Multinomial Logistic Regression Umaña-Hermosilla et al. (2020)) using the non-outliers and their cluster labels. Then we classify the texts in the test set using the trained classifier. This defines a new set of cluster labels of the texts in the test set and thus produces an

[6]https://github.com/rashadulrakib/short-text-clustering-enhancement

**Algorithm 1** Enhancement of Clustering by Iterative Classification

```
Require: $D$ = set of $n$ texts, $L$ = initial cluster labels of the texts in $D$, $K$ = number
    of clusters
Ensure: Enhanced cluster labels of the texts
 1: $maxIteration = 50$
 2: $avgTextsPerCluster = n/K$
 3: for $i = 1$ to $maxIteration$ do
 4:     Choose a parameter $P$ uniformly at random from the interval $[P_1, P_2]$. ($P_1$ and
        $P_2$ are parameters determined in Section 3.2.3. $P$ bounds the fraction of texts
        kept per cluster.)
 5:     Remove outliers from each of the $K$ clusters defined by $L$ using an outlier
        detection algorithm called Isolation Forest Liu et al. (2008)
 6:     If a cluster contains more than $avgTextsPerCluster \times P$ texts, remove texts from
        that cluster uniformly at random so that exactly
        $avgTextsPerCluster \times P$ texts remain in the cluster.
 7:     $testSet$ = texts removed in Steps 5 and 6
        $trainingSet$ = all the texts not in $testSet$
 8:     Train a classifier (i.e., Multinomial Logistic Regression Umaña-Hermosilla et al.
        (2020)) using the $trainingSet$ and classify the texts in $testSet$. This assigns a
        new cluster label $L(t)$ to each text $t \in testSet$.
 9:     Stop iterative classification if the per cluster text distribution becomes stable
        (as described in Section 3.2.2).
10: end for
11: return $L$
```

updated cluster partition. The reason to use Multinomial Logistic Regression is that this classification model performs better when the amount of noise (i.e., outliers) in data is relatively lower Couronné et al. (2018). Since we remove outliers from clusters before training the classification model, we choose Multinomial Logistic Regression as the classification model in our proposed clustering enhancement method.

### 3.2.2 Stopping Criterion for Iterative Classification

Iterative classification stops when it reaches the maximum number of iterations (i.e., 50) or the sizes of the clusters become stable. Let $C_1, ..., C_k$ and $C'_1, ..., C'_k$ be the clusters before and after an iteration, respectively. We consider the cluster sizes to be stable if

$$\frac{1}{k}\sum_{i=1}^{k} ||C'_i| - |C_i|| \leq 0.05\frac{n}{k}$$

For example, consider the problem of partitioning 100 texts into two clusters. Then the average cluster size is 50. If one iteration assigns 48 texts to the first cluster and 52 texts to the second cluster and the next iteration assigns 49 and 51 texts to these clusters, respectively, then the average absolute change of the cluster size is $\frac{1}{2}(|48-49|+|52-51|) = 1$. Since this is less than 5% of the average cluster size (50), we consider the cluster sizes to have stabilized.

### 3.2.3 Experimental Results of Short Text Clustering by Iterative Classification

In this section, we discuss the experimental results obtained by our clustering enhancement method based on iterative classification Rakib et al. (2020a). We used five short text datasets that were used in our previous short text clustering method Rakib et al. (2018) as shown in Table 3.1. The datasets we used are SearchSnippet, SearchSnippet-test, AgNews, StackOverflow, and BioMedical.

#### Experimental Setup for Iterative Classification

We preprocessed the texts by removing stop words and converting them to lowercase. Then we transformed each text into the *tf-idf* vector representation for a given text collection.

Each iteration of the iterative classification algorithm picks some percentage $P$ of each cluster as the training set and reassigns the remaining texts to clusters based on a classifier trained using this training set; $P$ is chosen uniformly at the random from some interval $[P_1, P_2]$. To justify this approach and to determine optimal choices for $P_1$ and $P_2$, we ran preliminary experiments using a representative dataset (SearchSnippet-test). Specifically, we considered choosing $P$ uniformly at random from the interval $[P_1, P_2]$ or choosing a fixed percentage $P$ in every iteration. For the former method, we determined the optimal combination of $P_1$ and $P_2$ ($P_1 = 0.5$ and $P_2 = 0.95$). For the later, we determined the optimal choice of $P$ ($P = 0.6$). Choosing $P$ uniformly at random from the interval [0.5, 0.95] resulted in cluster accuracy of 82.21 for the representative dataset. Choosing a fixed percentage $P = 0.6$ in every iteration resulted in cluster accuracy of 80.25. Thus we chose $P_1 = 0.5$ and $P_2 = 0.95$ and chose $P$ uniformly at random from this interval in all experiments.

## Experimental Setup for Clustering

To perform clustering, we used the preprocessed texts described in Section 3.2.3. Then, texts were represented as vectors using pretrained word embeddings (i.e., Glove Pennington et al. (2014) and BioASQ Du et al. (2018)). The Glove embedding[7] was trained using the Glove method Pennington et al. (2014) on Wikipedia dumps. The BioASQ embedding[8] was trained using the Word2Vec method Mikolov et al. (2013) on abstracts of biomedical publications. We used the Glove embedding for all datasets except the biomedical dataset since these datasets contained terms related to general domains such as search snippets. For the biomedical dataset, the BioASQ embedding was more appropriate due to its specific focus on biomedical terms.

We represented each text by the average of the vectors of all words in the text. Then, we applied different clustering methods (i.e., k-means, k-means-- Shekhar et al. (2003), hierarchical clustering) to the text vectors. For the k-means and k-means-- clustering algorithms, we used the text vectors as the points to be clustered. For hierarchical clustering, we constructed the dense similarity matrix by computing similarities between the vectors using cosine similarity for all the text pairs. After that,

[7]http://nlp.stanford.edu/data/glove.42B.300d.zip
[8]bioasq.lip6.fr/tools/BioASQword2vec/

we sparsified the dense similarity matrix using the $k$-NN and similarity distribution based ($SD$) sparsification methods as described in Section 3.1.2. Then we applied hierarchical agglomerative clustering using dense (HAC) and sparse similarity matrices (HAC_$k$-NN and HAC_$SD$).

### Results

We used accuracy (ACC) and normalized mutual information (NMI) as the evaluation measures for different clustering algorithms (as in Xu et al. (2017)). The clustering results (ACC, NMI) of these datasets are shown in Table 3.4. The last two rows of Tables 3.5 and 3.6 show the ACC and NMI scores obtained using the state-of-the-art short text clustering methods STC2-LE Xu et al. (2017) and SIF-Auto Hadifar et al. (2019). The ACC and NMI scores of five clustering algorithms both before and after iterative classification for the five datasets are shown in these two Tables. The results with or without the _$IC$ suffix are the results with or without iterative classification. The best result (ACC, NMI) for each dataset is shown in bold.

To compensate for the dependence of k-Means, k-Means-- on the choice of cluster seeds, we ran the k-Means and k-Means-- clustering algorithms 20 times on the same dataset and performed iterative classification on the clustering obtained in each run. After that, we calculated the mean and standard deviation of the 20 clustering results (ACC, NMI) obtained by k-Means, k-means--, k-Means_$IC$ and k-means--_$IC$ for each dataset. We ran hierarchical agglomerative clustering (HAC), HAC_$k$-NN, and HAC_$SD$ only once since HAC is deterministic. However, the enhancement of the clustering obtained by iterative classification varies between runs since the training and test sets are chosen randomly in each iteration. So, we ran iterative classification 20 times on the clustering obtained using HAC, HAC_$k$-NN and HAC_$SD$, and again calculated the mean and standard deviation of each of the 20 clustering results obtained by HAC_$IC$, HAC_$k$-NN_$IC$ and HAC_$SD$_$IC$ for each dataset.

### Impact of Iterative Classification

We evaluated whether iterative classification improves the initial clustering obtained using different clustering algorithms. We consider iterative classification to improve the clustering for a given dataset if both ACC and NMI are increased using iterative classification.

Table 3.4 shows that iterative classification improves the initial clustering of short texts in terms of both ACC and NMI. For most of the datasets, the best clustering ACC and NMI were obtained by applying iterative classification to the clustering obtained by HAC with SD sparsification (HAC_$SD$ Rakib et al. (2018)). The reason is that HAC_$SD$ produces better initial clustering than other clustering methods for these datasets and the enhancement of clustering depends on the initial clustering.

### Comparison with State-of-the-Art Methods

Our second comparison aims to assess how the results of iterative classification in conjunction with the different clustering methods compare to state-of-the-art short text clustering methods, specifically STC2-LE Xu et al. (2017) and SIF-Auto Hadifar et al. (2019). Table 3.5 and 3.6 show that HAC_$SD$_$IC$ and HAC_$k$-NN_$IC$ outperform STC2-LE[9] for the SearchSnippet, StackOverflow and BioMedical datasets in terms of ACC and NMI. It is also shown that HAC_$SD$_$IC$, HAC_$k$-NN_$IC$, HAC_$IC$, k-Means_$IC$, and k-means--_$IC$ outperform SIF-Auto for the SearchSnippet and StackOverflow datasets in terms of ACC and NMI. However, on the Biomedical dataset, the performance of SIF-Auto is better than any clustering method and its corresponding enhancement by iterative classification.

### Statistical Significance Testing of Clustering Performance

Our third comparison aims to investigate whether the clustering improvements achieved by iterative classification are statistically significant. In particular, we perform two investigations: a) whether the improved results achieved by iterative classification are statistically significantly better than the results of their corresponding clustering methods. b) whether the improved results achieved by our best clustering method HAC_$SD$_$IC$ are statistically significantly better than the results of different clustering methods (with or without iterative classification and state-of-the-art methods). For significance testing, we performed a two-tailed paired t-test (with significance level $\alpha = 0.05$) using the pairwise differences of clustering results (ACC, NMI) of 20

[9]We were unable to reproduce the clustering for other short text datasets using STC2-LE and SIF-Auto.

Table 3.4: ACC and NMI of different clustering methods, their corresponding enhancements by iterative classification, and state-of-the-art methods for short text clustering. $^{\Delta}$ indicates that this method is statistically significantly inferior to its corresponding enhancement obtained by iterative classification. * indicates that this method is statistically significantly inferior to HAC_*SD*_*IC*.

| Clustering Methods | Datasets | | | | |
|---|---|---|---|---|---|
| | Search Snippet | Search SnippetTest | AgNews | Stack Overflow | Bio Medical |
| | ACC(%) | ACC(%) | ACC(%) | ACC(%) | ACC(%) |
| HAC_*SD* | 82.69$^{\Delta}$ | 89.47$^{\Delta}$ | 81.84$^{\Delta}$ | 64.80$^{\Delta}$ | 40.13$^{\Delta}$ |
| HAC_*SD*_*IC* | **87.67**±0.63 | **92.16**±0.85 | **84.52**±0.50 | **78.73**±0.17 | 47.78±0.51 |
| HAC_*k*-NN | 79.08$^{\Delta}$* | 87.14$^{\Delta}$* | 76.83$^{\Delta}$* | 58.11$^{\Delta}$* | 39.75$^{\Delta}$* |
| HAC_*k*-NN_*IC* | 83.19*±0.61 | 90.76*±1.79 | 81.83*±0.35 | 70.07*±0.11 | 46.17*±1.10 |
| HAC | 76.54$^{\Delta}$* | 77.06$^{\Delta}$* | 76.56$^{\Delta}$* | 61.64$^{\Delta}$* | 38.86$^{\Delta}$* |
| HAC_*IC* | 80.63*±0.69 | 83.92*±2.66 | 81.13*±1.22 | 67.69*±2.12 | 46.13*±0.92 |
| k-Means | 63.89$^{\Delta}$*±1.15 | 63.22$^{\Delta}$*±1.79 | 58.17$^{\Delta}$*±1.87 | 41.54$^{\Delta}$*±2.16 | 36.92$^{\Delta}$*±0.81 |
| k-Means_*IC* | 83.13*±0.69 | 82.84*±2.32 | 78.06*±3.13 | 69.89*±1.52 | 43.50*±1.38 |
| k-means-- | 47.42$^{\Delta}$*±1.13 | 61.96$^{\Delta}$*±1.98 | 62.48$^{\Delta}$*±2.13 | 43.77$^{\Delta}$*±0.39 | 39.95$^{\Delta}$*±1.21 |
| k-means--_*IC* | 79.77*±2.67 | 75.29*±2.79 | 77.45*±3.49 | 69.25*±1.88 | 45.61*±3.19 |
| STC2-LE | 78.29*±2.72 | | | 53.81*±3.37 | 44.81*±1.72 |
| SIF-Auto | 79.13*±1.27 | | | 59.85*±1.81 | **55.73**±1.97 |

Table 3.5: ACC results

| Clustering Methods | Datasets | | | | |
|---|---|---|---|---|---|
| | Search Snippet | Search SnippetTest | AgNews | Stack Overflow | Bio Medical |
| | NMI(%) | NMI(%) | NMI(%) | NMI(%) | NMI(%) |
| HAC_*SD* | 63.76$^{\Delta}$ | 78.73$^{\Delta}$ | 54.57$^{\Delta}$ | 59.48$^{\Delta}$ | 33.51$^{\Delta}$ |
| HAC_*SD*_*IC* | **71.93**±1.04 | **85.55**±1.09 | **59.07**±0.84 | **73.44**±0.35 | 41.27±0.36 |
| HAC_*k*-NN | 60.51$^{\Delta}$* | 76.42$^{\Delta}$* | 52.43$^{\Delta}$* | 54.06$^{\Delta}$* | 32.19$^{\Delta}$* |
| HAC_*k*-NN_*IC* | 65.49*±0.97 | 83.17*±1.17 | 56.02*±0.86 | 68.88*±0.43 | 38.78*±0.53 |
| HAC | 59.41$^{\Delta}$* | 70.99$^{\Delta}$* | 52.82$^{\Delta}$* | 54.46$^{\Delta}$* | 31.01$^{\Delta}$* |
| HAC_*IC* | 63.61*±1.09 | 77.49*±1.11 | 56.57*±1.23 | 61.76*±1.35 | 38.50*±0.61 |
| k-Means | 43.75$^{\Delta}$*±1.31 | 51.54$^{\Delta}$*±0.92 | 35.26$^{\Delta}$*±2.01 | 38.01$^{\Delta}$*±2.12 | 33.71$^{\Delta}$*±0.29 |
| k-Means_*IC* | 66.27$^{\Delta}$±1.00 | 76.88$^{\Delta}$±2.64 | 52.32$^{\Delta}$±2.47 | 69.84$^{\Delta}$±0.66 | 38.08$^{\Delta}$±0.81 |
| k-means-- | 47.43$^{\Delta}$*±1.65 | 49.73$^{\Delta}$*±2.15 | 39.68$^{\Delta}$*±1.15 | 41.89$^{\Delta}$*±0.86 | 34.49*±1.93 |
| k-means--_*IC* | 63.01*±1.69 | 71.11*±2.40 | 51.05*±3.63 | 69.64*±1.28 | 35.63*±2.82 |
| STC2-LE | 64.72*±1.37 | | | 49.51*±1.63 | 38.42*±0.87 |
| SIF-Auto | 57.72*±1.43 | | | 55.59*±1.23 | **47.21**±1.19 |

Table 3.6: NMI results

runs obtained by different pairs of clustering methods.

On all datasets except the BioMedical dataset, and for all clustering methods tested, the enhancement by iterative classification is statistically significantly better than the base clustering method, and the former are statistically significantly inferior to our method HAC_$SD_IC$. For the BioMedical dataset, the ACC and NMI scores achieved by HAC_$SD_IC$ are statistically significantly better than that of STC2-LE. However, SIF-Auto outperforms HAC_$SD_IC$ on the BioMedical dataset.

# Chapter 4

# Dynamic Clustering of Texts

In this chapter, we discuss our two dynamic clustering methods (i.e., stream clustering) of short texts Rakib et al. (2020b, 2021). The principal difference between static and dynamic clustering is that in dynamic clustering, the amount of data to be clustered as well as the number of clusters to be produced are unknown as the data arrives over time; whereas in static clustering both the number of clusters and amount of data are known before clustering is being performed.

In our first dynamic clustering method Rakib et al. (2020b), we assign texts to clusters as they arrive, then remove outliers from the clusters, finally reassign the outliers to appropriate clusters. In our second method (called EStream Rakib et al. (2021)), we further improved the clustering result and running time performance than that of our previous short text stream clustering method. EStream comprises of two modules: online and offline. In online module, we improve running time performance by reducing the number of similarity computations by indexing clusters using inverted index Ilic et al. (2014). In offline module, we improve clustering result by enhancing the distributions of texts of the selected clusters using our clustering enhancement method Rakib et al. (2020a). In the following, we discuss each of these two dynamic clustering methods and experimental results obtained from them.

## 4.1 Short Text Stream Clustering via Frequent Word Pairs and Reassignment of Outliers to Clusters

In this work Rakib et al. (2020b), we cluster short text streams based on the frequent *word pairs*[1] in texts and reassigning outliers to proper clusters. This method clusters texts batch (collection of texts) by batch as they arrive[2].

Given a batch of texts, the proposed method clusters a fraction of texts in the

[1]Word pair is defined as two words in text not necessarily to be consecutive.

[2]https://github.com/rashadulrakib/short-text-stream-clustering/tree/master/BatchClustering

batch that contain frequently occurred *word pairs*. Then it builds a lexical clustering model using the cluster assignments obtained by frequently occurred *word pairs* in texts and clusters rest of the texts in the batch using the lexical clustering model. After that it removes outliers from the clusters, then computes semantic clustering model of the clusters and reassigns the outliers to clusters using the semantic similarity between the outliers and clusters. Then it deletes the outdated clusters. Finally it updates the lexical and semantic clustering models to reflect the new composition of clusters and uses the updated clustering models to cluster the next batch of texts.

### 4.1.1 Clustering Texts by Frequent ***Word Pairs***

We cluster a fraction of the texts in each batch using the frequent *word pairs* based on the assumption that each cluster can be represented by a single frequent *word pair*. At first, the word pairs are extracted from texts. Then, for each word pair, we create a list of document indices[3] where the word pair appears. We compute the mean ($\mu$) and standard deviation ($\sigma$) of the number of occurrences of the word pairs. The word pairs with the number of occurrences greater than the $\mu + \sigma$ are the word pairs that frequently appear in the texts.

The frequent word pairs are used to form the clusters based on the assumption that a frequent word pair will not co-exist with another frequent word pair in a cluster. In particular, a document should contain only one frequent word pair. However some documents may contain several frequent word pairs. Therefore we remove the indices of those documents from the lists of document indices of corresponding word pairs. Thus the texts in a cluster contain a single frequent word pair. If a frequent word pair in the texts of the current batch exists in the texts of a previous batch, then we merge these two lists of document indices.

### 4.1.2 Building the Lexical Clustering Model

To build the lexical clustering model, we construct cluster feature ($CF$) vector for each cluster obtained in Section 4.1.1. The $CF$ vector is constructed using the words with frequencies in texts, number of texts, and number of words in a cluster. We use the statistics of $CF$ vectors and adopt the MStream(F) Yin et al. (2018) algorithm

[3]We use global unique index for each document.

to calculate the probabilities for the rest of the texts (i.e., not clustered by frequent word pairs) in a batch to assign them to clusters.

The reason to construct $CF$ vector using the cluster assignments of texts obtained in Section 4.1.1 is to minimize the impact of the number of texts in clusters while calculating the probabilities for upcoming texts to assign them to the clusters since we adopt the MStream(F) algorithm for probability calculation and MStream(F) algorithm has the tendency to assign texts to the clusters having larger number of texts which may lead to an improper assignments of future texts to the clusters.

There may be some words that appear in the texts of different clusters (i.e., multi-cluster words) which may cause inaccurate similarity score between the texts and clusters. As a result, some texts may be assigned to inappropriate clusters. Therefore we keep only the most discriminative words in clusters so that the similarity score between a text and the cluster becomes more accurate so that the future texts can be assigned to more appropriate clusters.

To identify the most discriminative words, we compute the entropy Cover and Thomas (2006) of words using their distributions over the clusters. Higher entropy implies that the word occurs in more clusters. In particular, a word occurring in only one cluster has entropy 0. We compute the mean ($\mu$) and standard deviation ($\sigma$) of the entropies of all words and remove the words from clusters (i.e., from $CF$ vectors) whose entropy is greater than $\mu + \sigma$. After removing those high-entropy words, we update the number of words of corresponding $CF$ vectors.

### 4.1.3 Clustering the Rest of the Texts

We cluster the rest of the texts (i.e., not clustered by frequent word pairs in Section 4.1.1) in each batch using our lexical clustering model obtained in Section 4.1.2. We calculate the probability of a text choosing one of the existing clusters or a new cluster using the lexical clustering model and samples a cluster index for the text using these probabilities. When a text is assigned to a new cluster, we construct a new $CF$ vector using its words. Otherwise, we add the text to an existing cluster and update the $CF$ vector of the corresponding cluster, that is, we update the frequencies of words, number of documents, and number of words for that cluster.

### 4.1.4 Removing Outliers in Clusters

We identify outliers in the clusters obtained in each batch and remove them from the clusters. To identify outliers in clusters, we compute the connected components Cormen et al. (2009) of a text graph whose vertex set is the set of texts in the cluster, with an edge between two texts if they share a word. We consider the vertices in the smallest connected components to be outliers.

To compute these connected components more efficiently, without constructing the text graph explicitly, we construct an adjacency matrix representation of the word co-occurrence graph of the cluster. The vertex set of this graph is the set of all words that occur in at least one text in the cluster. There is an edge between two words if they co-occur in a text in the cluster.

Clearly, the words in each text belong to the same connected component of the word co-occurrence graph and two texts belong to the same connected component of the text graph if their words belong to the same connected component of the word co-occurrence graph. Thus, we compute the connected components of the word co-occurrence graph and then associate each text with the connected component of the word co-occurrence graph containing an arbitrary word in that text. The result is the collection of vertex sets of the connected components of the text graph.

To identify outliers, we calculate the mean ($\mu$) size (in number of vertices) of these connected components and their standard deviation ($\sigma$). The texts in a component are considered to be outliers if the component size is less than $\mu - \sigma$. In addition, we consider the text in every cluster containing only one text to be an outlier.

### 4.1.5 Building the Semantic Clustering Model

When we assign outliers to clusters, there may not be a sufficient number of common words between an outlier and its appropriate cluster which may result in inaccurate (even zero) similarity score between them. Therefore we use semantic representations of the clusters to reassign the outliers to their appropriate clusters.

The texts remaining in the clusters after removing outliers are used to compute the semantic clustering model of the clusters. The high-entropy words are determined based on their cluster distributions (described in Section 4.1.2) and removed from the texts. After removing high-entropy words from texts, we compute the representation of a cluster using the remaining words in texts. We obtain a vector representation of each word using the pre-trained Glove word embedding Pennington et al. (2014). We obtain a *document vector* representing each text by summing up the embeddings of the words in this text. Then we sum up the embeddings of the texts in each cluster to obtain a *cluster vector* which is then divided by the number of texts in the cluster called *cluster center*. The *cluster vector* and *cluster center* are used for the semantic clustering model of a cluster.

If the corresponding cluster of a *cluster vector* in the current batch exists in the previous batches, then we add that *cluster vector* with the corresponding *cluster vector* obtained in the previous batches and recompute the corresponding *cluster center*.

### 4.1.6 Assigning Outliers to Clusters

For each outlier removed from a cluster during outlier removal (Section 4.1.4), we compute the cosine similarity between the *document vector* of that outlier and the *cluster centers* of all clusters; then we calculate the $\mu$ and $\sigma$ of the similarities.

We assign the outlier to the cluster with the highest cosine similarity if the highest similarity is greater than the $\mu + \sigma$ of the similarities. Otherwise, we create a new cluster containing this outlier. Thus the outliers are assigned to clusters based on the dynamic similarity thresholds. Why we use $\mu + \sigma$ as the similarity threshold for assigning a text to a new or existing cluster is described in Appendix B. For every outlier added to a new or an existing cluster, we create (or update) the $CF$ vector, *cluster vector* and *cluster center* to reflect the addition of this outlier to a new or an existing cluster.

### 4.1.7 Deleting Outdated Clusters

We remove the outdated clusters based on their update-timestamps and cluster-sizes (i.e., number of texts in clusters) except the clusters being created or updated in the current batch. When a cluster is being created or updated, a new unique *cluster id* will be assigned to it. Thus, the recent clusters (created or updated) will be assigned highers *cluster ids*. We calculate the $\mu$ and $\sigma$ of the cluster ids and cluster-sizes of the clusters created or updated in previous batches. If the *cluster id* is less than the

$\mu - \sigma$ of cluster ids and the cluster-size is less than the $\mu - \sigma$ of cluster-sizes, then we delete the cluster by deleting the corresponding frequent *word pair* (along with document indices), $CF$ vector, *cluster vector*, and *cluster center*.

### 4.1.8 Experimental Results of Short Text Stream Clustering via Frequent Word Pairs and Reassignment of Outliers to Clusters

In this section, we discuss the datasets and experimental results of dynamic clustering of texts obtained by our method based on frequent word pairs in texts and outlier reassignments.

#### Datasets

We used four different datasets of short texts in our experiments. The basic properties of these datasets are shown in Table 4.1.

Table 4.1: Summary of short text datasets

| Dataset | #Clusters | #Texts | Avg. #words/text |
|---|---|---|---|
| Ns-T | 152 | 11,109 | 6.23 |
| Ts-T | 269 | 30,322 | 7.97 |
| NT | 416 | 41,429 | 6.91 |
| NTS | 438 | 61,428 | 8.13 |

The datasets **Ns-T** Yin et al. (2018) and **Ts-T** Yin et al. (2018) consist of 11,109 news titles and 30,322 tweets and are distributed into 152 and 269 groups respectively. The dataset **NT** was constructed by combining the news titles and tweets of the datasets Ns-T and Ts-T respectively. It consists of 41,429 texts distributed into 416 clusters. The dataset **NTS** was created by combining the texts of **Ns-T**, **Ts-T** and **StackOverflow** Xu et al. (2017) datasets consisting of 61,428 texts distributed into 438 clusters.

#### Comparison with State of the Art Methods

We compare the performance of our proposed method with other state-of-the-art short text stream clustering methods, MStream(F) Yin et al. (2018), DP-BMM(-FP) Chen et al. (2020), and OSDM Kumar et al. (2020) using normalized mutual information (NMI) as shown in Table 4.2.

Table 4.2: NMI score of different clustering methods. * indicates that the proposed method is statistically significantly better than other methods on a particular dataset in terms of NMI.

| Datasets | Clustering Methods | | | | | |
|---|---|---|---|---|---|---|
| | Proposed | MStream | MStreamF | DP-BMM | DP-BMM-FP | OSDM |
| Ns-T | 0.862 | 0.859 | 0.852 | **0.878** | 0.838 | 0.857 |
| Ts-T | **0.892*** | 0.867 | 0.874 | 0.862 | 0.875 | 0.842 |
| NT | **0.830*** | 0.798 | 0.743 | 0.762 | 0.741 | 0.791 |
| NTS | **0.756*** | 0.716 | 0.681 | 0.701 | 0.693 | 0.720 |

We divide each dataset into 16 batches where the texts in each batch are mutually exclusive. We perform 20 independent trials for each of the six algorithms on each dataset. The averages of the results (NMI) of these runs are shown in Table 4.2. Our experimental results show that the proposed method performs better than the state-of-the methods in terms of NMI on all datasets except **Ns-T**. Moreover, the performance of our method is statistically significantly better than the performance of other state-of-the methods in terms of NMI on all datasets except **Ns-T**. For significance testing, we performed a two-tailed paired t-test (with significance level 0.05) using the pairwise differences of clustering results (NMI) of 20 trials obtained by different pairs of clustering methods.

The average running times (in seconds) of our proposed method and other state-of-the-art methods are shown in Table 4.7. The running time of our method is less than that of MStream(F), DP-BMM(-FP), and OSDM on all datasets as we keep only the discriminative words of texts that belong to the clusters whereas MStream(F) and OSDM keep all the words of texts and DP-BMM(-FP) keeps all *word pairs* of texts that belong to the clusters; and the running time of similarity computation depends on the number of features (e.g., words, *word pairs*) in the texts and clusters.

Table 4.3: Average running times (in seconds) of the algorithms.

| Datasets | Clustering Methods | | | | | |
|---|---|---|---|---|---|---|
| | Proposed | MStream | MStreamF | DP-BMM | DP-BMM-FP | OSDM |
| Ns-T | 80 | 225 | 175 | 5019 | 881 | 97 |
| Ts-T | 165 | 547 | 297 | 12318 | 2857 | 191 |
| NT | 583 | 819 | 695 | 18217 | 4728 | 690 |
| NTS | 876 | 1637 | 1055 | 23871 | 7325 | 1032 |

## 4.2 Efficient Clustering of Short Text Streams using Online-Offline Clustering

In this work Rakib et al. (2021), we address the two major challenges of clustering short text streams. The first challenge is to cluster text streams within a reasonable amount of time and; the second challenge is to achieve better clustering result. To overcome these two challenges, we propose an efficient short text stream clustering algorithm (called EStream) using online-offline clustering[4].

We measure the efficiency of our method in extent of running time and clustering result. The online module of our method adopts inverted index based data structure to reduce running time. By adopting inverted index, EStream assigns a text to a cluster by computing similarity between a text and a selected number of clusters instead of all clusters and thus significantly reduces the running time of the clustering of short text streams. The offline module of EStream algorithm enhances the distributions of texts in the clusters obtained by the online module so that the upcoming short texts can be assigned to the appropriate clusters. This helps us to achieve better clustering results. Before we describe our short text stream clustering method, we briefly discuss the background information related to our method.

### 4.2.1 Background

In this section, we briefly describe the key concepts used in our short text stream clustering method (i.e., EStream) which are Text Representation, Cluster Representation, Inverted Index, and Concept Drift.

#### Text Representation

We use two kinds of representations of texts: lexical and semantic.

***Lexical Representation of Text*** For lexical representation, we represent each text using three different kinds of lexical features which are unigram, bigram and biterm and use each representation separately to cluster the streams of texts. Different feature ($f$) representation of a text of $k$ words are shown Table 4.4.

[4]https://github.com/rashadulrakib/short-text-stream-clustering/tree/master/OnlineClustering

Table 4.4: Lexical Feature Representation of Text

| Feature type | Feature representation of text of $k$ words | #Features |
|---|---|---|
| unigram | $\{w_i \mid i \in [1, k]\}$ | $k$ |
| bigram | $\{(w_i, w_{i+1}) \mid i \in [1, k-1]\}$ | $k - 1$ |
| biterm | $\{\{w_i, w_j\} \mid i, j \in [1, k]\ and\ i \neq j\}$ | $\frac{k \times (k-1)}{2}$ |

The words of a text are considered as the unigram features of that text. The two consecutive words of a text are considered as the bigram features of that text. For example, the text "ai improves healthcare system" will be represented by the following bigrams: "ai improves", "improves healthcare", and "healthcare system". The biterm feature is defined as an unordered *word pair* constructed using the words in a text Chen et al. (2020). The same text will be represented by the following biterms: "ai improves", "ai healthcare", "ai system", "improves healthcare", "improves system", and "healthcare system".

***Semantic Representation of Text*** For semantic representation, we construct a document vector representing each text by averaging the embeddings of the words in the text. Embeddings of the words are obtained from Glove pre-trained word embedding Pennington et al. (2014).

### Cluster Representation

***Lexical Representation of Cluster*** In our method, each cluster is represented by a cluster feature ($CF$) vector Yin et al. (2018) consisting of 4 tuples $\{n_z^f$, $n_z$, $m_z$, $id_z$ $\}$ where $n_z^f$ refers to the features (unigram, bigram or biterm) along with frequencies in cluster $z$, $n_z$ refers to the number of features in cluster $z$, $m_z$ refers to the number of texts in cluster $z$, and $id_z$ refers to the unique $id$ for cluster $z$.

***Semantic Representation of Cluster*** To obtain the semantic representation of cluster, we compute cluster vector by summing up the document vectors of the texts in each cluster. After that cluster center is computed by dividing the cluster vector by the number of texts in the cluster. Thus the semantic representation of each cluster consists of the cluster vector and the cluster center.

#### Inverted Index

Inverted Index is a hashmap like data structure that creates mapping from document-features (unigram, bigram or biterm) to documents Ilic et al. (2014). To keep track of which clusters are associated with which features, we adopt the inverted index based searching technique and create a vector $F$ for each feature defined as a tuple of $\{l_f^{id}\}$ where $l_f^{id}$ refers to the list of cluster $ids$ associated with a feature $f$.

#### Concept Drift

Concept drift is a phenomenon in which the characteristics of the data changes in an arbitrary way over the course of time Lu et al. (2019). In the context of text stream clustering, concept drift can be defined as the problem that the topics of the text streams may change over time Zhang et al. (2017b). In particular, when a new text arrives, we need to decide whether the text will be assigned to a new cluster (i.e., a new topic has been emerged in the text stream) or the text will be assigned to one of the existing clusters (i.e., the topic of the text is similar to the topic of an existing cluster). To decide, whether a new text will be assigned to a new or one of the existing clusters, we use dynamic similarity threshold Rakib et al. (2020b) as described in Section 4.2.2.

### 4.2.2 Proposed Method

Our proposed short text stream clustering method (EStream) consists of two modules: online and offline as shown in Figure 4.1. In the online module, EStream clusters each short text one by one as it arrives. In the offline module, EStream enhances the distributions of the texts of the selected clusters obtained by online module in every Update-interval ($UI$). Update-interval is the interval when we perform offline clustering after clustering a certain number of texts using online clustering. In the offline module, EStream also deletes the outdated clusters and updates the Lexical and Semantic clustering models defined in Section 4.2.1. Based on the updated clustering models the subsequent texts are clustered.

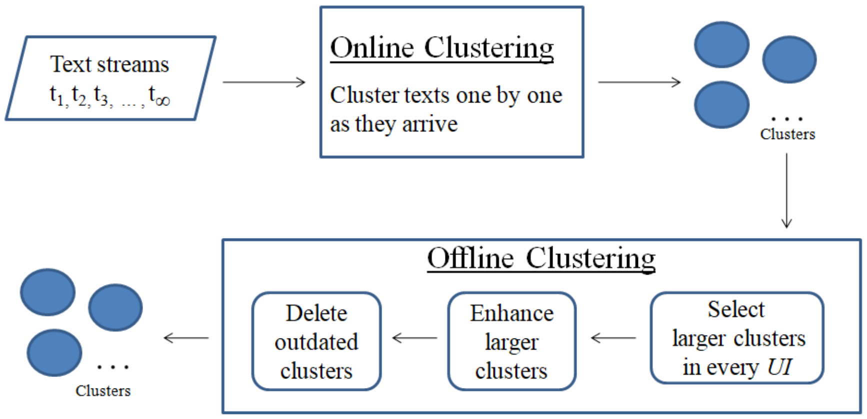


Figure 4.1: Proposed short text stream clustering using Online-Offline clustering.

**Online Clustering**

At first we remove stop words from the text, then we extract features from that text. At a time, we use only one type of feature representation to cluster the streams of texts. That means, we extract only either unigram, bigram or biterm features from the text. After that we select the clusters that contain the features of the text. Then our method computes similarities between the text and the selected clusters using common features. Following that it assigns the text to an appropriate cluster (new or existing) using the dynamically computed similarity thresholds based on statistical measure. After that it builds a clustering model using the cluster assignment of the text to reflect the addition of this text to a new or an existing cluster; and uses the current clustering model to cluster the subsequent text. The details are shown in Algorithm 2 and are described next. The notations used in Algorithm 2 are shown in Table 4.5.

**Algorithm 2** Proposed Online Text Stream Clustering

```
Require: Texts: $t_1 ... t_\infty$
Ensure: Cluster assignments: $z_{t_1 ... t_\infty}$
 1: for $t_i$ in $t_1 ... t_\infty$ do
 2:     Extract features ($f$) from $t_i$
 3:     Select $L$ clusters that share common features with $t_i$ (described in Section 4.2.2)
 4:     Compute similarities ($s_l$) between $t_i$ and the $L$ selected clusters using Equation 4.1
 5:     Compute the maximum ($\max_l$), mean ($\mu_l$) and standard deviation ($\sigma_l$) of the $s_l$ similarities
 6:     if $\max_l > \mu_l + \sigma_l$ then
 7:         $j =$ cluster index for $\max_l$
 8:         Assign $t_i$ to $j^{th}$ cluster
 9:     else
10:         Assign $t_i$ to a new cluster
11:     end if
12:     Build clustering model (described in Section 4.2.2)
13: end for
```

Table 4.5: Notations used in the Algorithm of Online Text Stream Clustering

| Notation | Definition |
|---|---|
| $similarity(t, z)$ | Similarity between a text $t$ and cluster $z$ |
| $FI^f$ | Feature-Importance ($FI$) of a feature ($f$) with respect to cluster $z$ |
| $\|f \in z\|$ | Number of clusters containing feature $f$ |
| $n_z^f$ | Number of occurrences of feature $f$ in cluster $z$ |
| $n_z$ | Number of occurrences of features in cluster $z$ |
| $N_t^f$ | Number of occurrences of feature $f$ in text $t$ |
| $N_t$ | Number of occurrences of features in text $t$ |
| $m_z$ | Number of texts in cluster $z$ |

### Selecting Clusters for the Text

For each text, we select a specific set of clusters (based on inverted index Ilic et al. (2014)) that share common features with the text. For each feature in a text, we obtain the cluster *ids* from the corresponding feature vector $F$. Then we aggregate the cluster *ids*. These aggregated clusters *ids* are considered as the selected clusters for the text.

### Computing Lexical Similarities between the Text and Selected Clusters

In online clustering, EStream computes similarities between the text and the selected clusters based on the common features following the notion of Dice similarity Thada and Jaglan (2013) as shown in Equation 4.1.

$$similarity(t, z) = \frac{\sum_{f \in t \wedge z} \left( N_t^f + n_z^f \right) \times FI^f}{N_t + n_z} \tag{4.1}$$

To compute similarity between a text $t$ and a cluster $z$, we sum the occurrences of each common feature between $t$ and $z$ multiplied by the feature-importance of the corresponding feature denoted by $FI^f$. After that we normalize the value of the summation by the total number of features in text $t$ and cluster $z$ denoted by $N_t$ and $n_z$ respectively. Here $n_z^f$, $N_t^f$ refer to the features ($f$) along with frequencies in cluster $z$ and text $t$ respectively. The feature-importance ($FI$) of a feature ($f$) is calculated following the notion of inverse document frequency Robertson (2004) defined in Equation 4.2. $|z|$ refers to the total number of existing clusters. $|f \in z|$

refers to the number of clusters that contain the feature $f$.

$$FI^f = \log \frac{|z|}{|f \in z|} \tag{4.2}$$

The features that occur in many clusters are less discriminative for choosing the correct cluster than the features that occur in few clusters, ideally a single cluster. If a feature occurs much more often in one cluster than in other clusters, then this feature is also highly discriminative in favour of this single cluster. Therefore by multiplying the feature-importance with the occurrences of features, we obtain more accurate similarity score between the texts and the clusters which in turn helps us to assign the texts to the more appropriate clusters.

### Assigning Text to Cluster by Lexical Similarity

To assign a text to a cluster, we compute the maximum ($max$), mean ($\mu$) and standard deviation ($\sigma$) of the similarities between the text $t$ and the selected clusters. We assign the text to the cluster with the maximum similarity if the maximum similarity is greater than the $\mu$+$\sigma$ of the similarities. Otherwise, we create a new cluster containing this text. Thus the texts are assigned to clusters based on the dynamically computed similarity thresholds Rakib et al. (2020b). The intuition behind using maximum similarity greater than $\mu + \sigma$ is that, this maximum similarity is above the average similarities reflecting that both the text and the target cluster share highly similar content.

### Building Lexical Clustering Model

We build clustering model using the cluster assignment of the text to reflect the addition of this text to an existing or a new cluster. When a text $t$ is added to a cluster $z$, we update the corresponding $CF$ vector by updating its features with frequencies ($n_z^f$), number of features ($n_z$), and number of texts ($m_z$). The addible property of the $CF$ vector is described in the following.

**Addible Property of Text to Cluster:**

$n_z^f = n_z^f + N_t^f \quad \forall f \in t$

$n_z = n_z + N_t$

$$m_z = m_z + 1$$
$$id_z = \begin{cases} id_z;\ if\ successive\ texts\ are\ in\ same\ cluster \\ max(id_z) + 1,\ \forall z \in\ Z;\ otherwise \end{cases}$$

Here, $N_t^f$ and $N_t$ refer to the features with frequencies in text $t$ and the total number of features in text $t$ respectively. $Z$ is the set of all $CF$ vectors (i.e., $Z = \{CF\}$).

A new $id$ is assigned to the cluster $z$ (new or existing) if the cluster assignment of the current text is different from that of the previous text, otherwise the cluster $id$ of the current text remains same as that of the previous text. This implies that the recently created or updated cluster will have the highest cluster $id$. For each feature in the text, we append the cluster $id$ to the corresponding feature vector $F$.

### Offline Clustering

We perform offline clustering in every Update-interval ($UI$). In every Update-interval, we select a set of clusters obtained by the online clustering module of EStream algorithm. Then we enhance the distributions of the texts of those clusters. After that we assign a set of texts to clusters by semantic similarity. Following that we update both the lexical and semantic clustering models of the clusters. Finally we delete the outdated clusters.

***Selecting Clusters*** We select a specific set of larger clusters obtained by the online clustering module based on the cluster-sizes[5]. In particular, we select the clusters whose sizes are greater than the mean ($\mu$) and standard deviation ($\sigma$) of the sizes of the clusters. The reason to select larger clusters is that larger clusters tend to have more outliers than the smaller clusters which may cause improper assignments of the future texts to the clusters. Therefore by improving the distributions of the texts in the larger clusters, we can assign the upcoming texts to the appropriate clusters.

[5]Cluster-size refers to the number of texts in a cluster.

***Enhancement of Clusters*** We enhance the distributions of the texts of the selected larger clusters using a state-of-the-art clustering enhancement method as described in Section 3.2. The clustering enhancement method requires three input parameters which are the collection of texts ($n$), number of target clusters ($k$) and the initial clustering labels of the texts ($L$). We use the texts of the larger clusters as the collection of texts, the $k$ number of larger clusters as the number of target clusters, and the corresponding clustering labels of the texts as the initial clustering labels.

***Assigning Texts to Clusters by Semantic Similarity*** We select the clusters containing only one text and try to reassign the texts of these clusters to the rest of the existing clusters. The reason to select the clusters containing only one text is that the texts of these clusters do not share any common lexical features with other clusters. However these texts may be semantically similar to other existing clusters. Therefore we compute the semantic representations of these texts and the semantic representations of the rest of the existing clusters as described in Section 4.2.1.

For example, the texts "storm boreas barrel east coast leaving wake cancelled flight" and "causing epic thanksgiving storm" form a cluster since they have common feature (e.g., storm) between them. If we want to assign the text "rain snow threaten snarl holiday travel" to this cluster, then we cannot assign it to this cluster since this text does not have any common feature with the cluster. On the contrary, this text is semantically similar to the above cluster. Therefore we use semantic similarity to reassign the texts of clusters with single text to the rest of the existing clusters.

To assign a text to a cluster we compute cosine similarity between the semantic representations of the text and the clusters. Then we compute the maximum (max), mean ($\mu$) and standard deviation ($\sigma$) of the semantic similarities for a particular text. If the maximum similarity is grater than the mean $\mu + \sigma$ of the similarities, then the text will be reassigned to the cluster with the highest cosine similarity otherwise it will be kept in the previous cluster obtained by the online clustering module of EStream algorithm.

***Updating Clustering Models*** We update both the lexical and semantic representations of the clusters after a text is reassigned to a cluster. Lexical representation of the cluster is being updated by updating the corresponding cluster feature ($CF$) vector as defined in Section 4.2.1. Semantic representation of the cluster is being updated by updating the corresponding cluster vector and cluster center as defined in Section 4.2.1.

***Deleting Outdated Clusters*** We remove the outdated clusters based on their update-timestamps (represented by cluster *id*) and cluster-sizes. The recently created or updated clusters will have higher cluster *ids*.

We remove outdated clusters in every Update-interval. To obtain outdated clusters, we calculate the $\mu$ and $\sigma$ of the cluster *ids* and cluster-sizes. If the *cluster id* is less than the $\mu - \sigma$ of cluster *ids* and the cluster-size is less than the $\mu - \sigma$ of cluster-sizes, then we delete the cluster by deleting the corresponding $CF$ vector and remove the corresponding cluster *id* from the feature vectors ($F$) that contain that particular cluster *id*.

### 4.2.3 Experimental Results of Efficient Clustering of Short Text Streams using Online-Offline Clustering

In this section we describe the experimental results obtained from our proposed short text stream clustering method (called EStream). At first we discuss the datasets that we use in our experiments. Then we compare the clustering performance and running time of EStream with other state-of-the-art short text stream clustering methods.

We have used three datasets from our previous work on short text stream clustering Rakib et al. (2020b) which are Ns-T, Ts-T, and NT. In addition to these datasets, we create a new dataset called **SO-T** using the titles of the StackoverFlow questions consisting of 123,342 question titles distributed into 10,573 clusters. The average number of words per text in the datasets **Ns-T**, **Ts-T**, **NT**, and **SO-T** are 6.23, 7.97, 6.91, and 5.57 respectively. The texts in these four datasets were randomly shuffled to examine how EStream and the state-of-the-art methods (MStream(F) Yin et al. (2018), DP-BMM(-FP) Chen et al. (2020), and OSDM Kumar et al. (2020)) perform when dealing with the texts from different topics arriving in random order.

### Construction of the Dataset SO-T

We create a dataset **SO-T** using the titles of the duplicate questions posted in StackOverflow[6] on various topics such as *Java*, *Python*, *JQuery*, *R*, and so on. We consider that duplicate questions are similar to each other and a group of similar questions can form a cluster.

We obtain the question titles from the file Posts.xml[7] and obtain the information about the duplicate questions from PostLinks.xml. Each item in Posts.xml represents a single post which can be of different types (e.g., question, answer, and so on). Each item of the PostLinks.xml contains the information about a pair of duplicate questions. For instance, the PostLinks.xml contains the questions A and B if they are duplicate. There are 20,094,655 questions in Posts.xml and 1,009,249 pair of duplicate questions in PostLinks.xml. Among the 1,009,249 pair of duplicate questions, we randomly select 400,000 pairs[8].

Using the duplicate information in PostLinks.xml, we create a list of directed edges (e.g., $A \rightarrow B$, $B \rightarrow C$) which are then used to create a graph. To obtain the clusters of duplicate questions, we compute connected components[9] using the representation of the graph. In particular, If A and B are duplicate, and B and C are duplicate, then we obtain the connected component $A \rightarrow B \rightarrow C$ which is considered as a cluster of the duplicate questions of A, B, and C.

We compute the length of the connected components defined as the number of questions in the component. After that, we compute the mean ($\mu$) and standard deviation ($\sigma$) of the lengths of the connected components and select the components whose lengths are between the $\mu \pm \sigma$ which in turn produces 10,573 connected components (i.e., clusters) consisting of 123,342 question titles. Sample StackOverflow question titles (with *PostId*) of a cluster in the dataset **SO-T** are shown in Figure 4.2.

[6]https://stackoverflow.com/

[7]https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede

[8]We select this specific number of pairs (400,000) because of the maximum capacity of the computer (Core i5-4200U and 8GB memory) where the experiments were carried out.

[9]We use the library in https://networkx.github.io/ to compute connected components.

– Python Video Framework (1003376)
– Best video manipulation library for Python? (220866)
– Trim (remove frames from) a video using Python (7291653)

Figure 4.2: Sample StackOverflow question titles (with *PostId*) of a cluster in the dataset **SO-T**.

### Optimal Update-interval for Offline Clustering

Our proposed method (EStream) requires only one parameter called Update-interval ($UI$) to perform offline clustering to enhance the distributions of the texts in the clusters and remove the outdated clusters. We set Update-interval to 500 for all the datasets used in our experiments implying that we enhance the cluster-distributions and remove outdated clusters after clustering every 500 texts using our online clustering module. How we choose this value is discussed in the following.

To determine the optimal value of Update-interval, we choose the dataset Ns-T. We determine the value of Update-interval based on the optimal clustering performance of our method for the dataset Ns-T; and use this value to perform offline clustering for other datasets. The clustering performance (in terms of NMI) of our method for different Update-intervals on the dataset Ns-T is shown in Figure 4.3. To perform clustering we use biterm features of the texts of the dataset Ns-T.

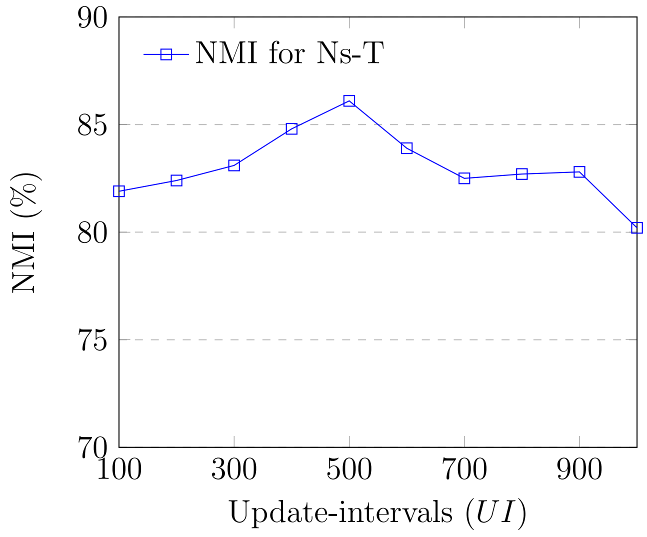


Figure 4.3: NMI results of our short text stream clustering method for different Update-intervals ($UI$) on dataset Ns-T.

Based on the different values of $UI$, we observe that we achieve the highest NMI (0.861) for Ns-T when $UI$=500. Therefore, we choose $UI$=500 for all datasets used in our experiments to perform offline clustering.

### Baseline Clustering Methods

We compare the performance of EStream with the recent state-of-the-art short text stream clustering methods as described in the following.

- **MStream** Yin et al. (2018) algorithm clusters each batch of short texts at a time. It stores all the clusters produced over the course of time. MStream has one pass clustering process and update clustering process of each batch. In update clustering process, it applies Gibbs sampling to the same batch of texts multiple times to improve the initial clustering result obtained in one pass clustering process.

- **MStreamF** Yin et al. (2018) is a variant of MStream algorithm that deletes the outdated clusters of previous batches and only stores the clusters of the current batch.

- **DP-BMM** Chen et al. (2020) is a short text stream clustering algorithm that adopts the similar approach as MStream and clusters each batch of short texts at a time as it arrives. The principal difference between DP-BMM and MStream is that DP-BMM represents the texts using biterm features instead of unigrams (i.e., words). Therefore, DP-BMM stores the clusters of texts using biterm features.

- **DP-BMM-FP** Chen et al. (2020) is a variant of DP-BMM algorithm that deletes the outdated clusters of previous batches and only stores the clusters of the current batch similar to MStreamF.

- **Rakib et al. (2020b)** proposed a short text stream clustering method by adopting the clustering model of MStream algorithm Yin et al. (2018). The main difference between this method and MStream algorithm is that this method removes outliers from the clusters obtained by MStream algorithm and reassigns the outliers to the clusters using dynamically computed similarity thresholds.

- **OSDM** Kumar et al. (2020) is a short text stream clustering algorithm that clusters each short text one by one as it arrives. OSDM deletes a cluster if a cluster becomes outdated, that is, the cluster is not being updated for a while over time.

For MStream(F), we set the parameters $\alpha = 0.03$ and $\beta = 0.03$ for all the datasets as defined in Yin et al. (2018). Likewise, for DP-BMM(-FP), we set $\alpha = 0.6$ and $\beta = 0.02$ as mentioned in Chen et al. (2020). MStream(F) and DP-BMM(-FP) cluster each batch of texts at a time. We set the batch size[10] to 2000 for MStream(F) and DP-BMM(-FP) for all datasets. We set the number of iterations to 10 for MStream(F) and DP-BMM(-FP), and number of saved batches to one for MStreamF and DP-BMM-FP as mentioned in Chen et al. (2020).

For OSDM, we set $\alpha = 2e^{-3}$, $\beta = 4e^{-5}$, and an additional parameter $\lambda = 6e^{-6}$ for all the datasets as defined in Kumar et al. (2020). OSDM used $\lambda$ as the decay rate which is used to set lower weights to the clusters which are not being updated over time and need to be deleted in the future.

**Faster Version of Baseline Clustering Methods**

- **Fast-MStream** and **Fast-MStreamF** are the faster versions of MStream and MStreamF algorithms that we developed respectively. We apply the inverted index Ilic et al. (2014) based searching technique using the words of the clusters produced by the MStream and MStreamF algorithms.

- **Fast-DP-BMM** and **Fast-DP-BMM-FP** are the faster versions of DP-BMM and DP-BMM-FP algorithms respectively. We apply the inverted index based searching technique using the biterms of the clusters produced by the DP-BMM and DP-BMM-FP algorithms.

- **Fast-Rakib et al. (2020b)** is the faster version of Rakib et al. (2020b). We index the clusters produced by Rakib et al. (2020b) using unigrams following the notion of Fast-MStream.

[10]The batch size equal to 2000 was chosen for MStream(F) and DP-BMM(-FP) based on their optimal performance on the datasets used in this paper.

### Comparison with State-of-the-art Methods

***Comparison of Clustering Results*** We compare the performance of our proposed method (EStream) with the state-of-the-art short text stream clustering methods and their corresponding faster versions that we developed. We apply different types of text representations (unigram, bigram, and biterm) to EStream denoted as EStream-unigram, EStream-bigram, and EStream-biterm. We use normalized mutual information (NMI), Homogeneity (Ho.), V-Measure (VM) as the evaluation measures for evaluating the performance of different clustering methods.

We randomly shuffle each dataset 20 times. Then we perform 20 independent trials for each of the methods on each dataset. The averages of the results[11] (NMI, Ho. and VM) of these runs are shown in Table 4.6.

Our experimental results show that EStream performs better than the state-of-the-art methods in terms of NMI, Ho. and VM on the dataset NT and SO-T. Moreover, the performance of our method is statistically significantly better than that of the state-of-the-art methods on these two datasets. For significance testing, we performed a two-tailed paired t-test Dahiru (2008) (with significance level 0.05) using the pairwise differences of clustering results (NMI, Ho. and VM) of 20 trials obtained by different pairs of clustering methods.

The state-of-the-art methods MStream(F), DP-BMM(-FP), Rakib et al. (2020b), and OSDM perform better than EStream on the datasets Ns-T and Ts-T since these methods tune the hyper parameters (e.g., $\alpha$, $\beta$) on these datasets to achieve better assignments of texts to the clusters (new or existing). On the contrary, our method uses dynamic similarity thresholds Rakib et al. (2020b) to assign texts to the clusters (new or existing) and does not require this kind of hyper parameter tuning on a particular dataset.

EStream tunes the only hyper parameter Update-Interval ($UI$) on a single dataset (Ns-T) and uses the same value of $UI$ for rest of the datasets. This signifies that EStream is less sensitive to the hyper parameter. Therefore, the overall performance of our method is comparable to the performance of the state-of-the-art methods on the datasets Ns-T and Ts-T. In addition, our method significantly outperforms the

[11]We could not run DP-BMM(-FP) on the dataset SO-T because of their longer running time that exceeds the capacity of the experimental computer.

Table 4.6: Normalized Mutual Information (NMI), Homogeneity (Ho.), V-Measure (VM) score of different clustering methods. * indicates that the EStream is statistically significantly better than other methods on a particular dataset in terms of NMI, Ho., or VM. The highest result for a particular dataset is denoted bold.

| Clustering Methods | Eva. | Data Sets | | | |
|---|---|---|---|---|---|
| | | Ns-T | Ts-T | NT | SO-T |
| EStream-unigram | NMI | 0.826 | 0.825 | 0.857* | 0.748* |
| EStream-bigram | | 0.851 | 0.848 | 0.873* | 0.781* |
| EStream-biterm | | 0.861 | 0.859 | **0.884*** | **0.794*** |
| Fast-MStream | | 0.858 | 0.864 | 0.799 | 0.609 |
| Fast-MStreamF | | 0.868 | 0.873 | 0.747 | 0.618 |
| Fast-DP-BMM | | **0.879** | 0.868 | 0.758 | – |
| Fast-DP-BMM-FP | | 0.832 | 0.875 | 0.740 | – |
| Fast-Rakib et al. (2020b) | | 0.862 | 0.891 | 0.833 | 0.651 |
| MStream | | 0.859 | 0.867 | 0.798 | 0.608 |
| MStreamF | | 0.869 | 0.874 | 0.743 | 0.619 |
| DP-BMM | | 0.878 | 0.862 | 0.762 | – |
| DP-BMM-FP | | 0.838 | 0.875 | 0.741 | – |
| OSDM | | 0.858 | 0.842 | 0.791 | 0.441 |
| Rakib et al. (2020b) | | 0.862 | **0.892** | 0.830 | 0.654 |
| EStream-unigram | Ho. | 0.858 | 0.851 | 0.863* | 0.799* |
| EStream-bigram | | 0.861 | 0.882 | 0.898* | 0.819* |
| EStream-biterm | | 0.876 | 0.898 | **0.915*** | **0.824*** |
| Fast-MStream | | 0.901 | 0.875 | 0.619 | 0.618 |
| Fast-MStreamF | | 0.873 | 0.895 | 0.663 | 0.647 |
| Fast-DP-BMM | | 0.891 | 0.864 | 0.656 | – |
| Fast-DP-BMM-FP | | 0.848 | 0.855 | 0.644 | – |
| Fast-Rakib et al. (2020b) | | 0.883 | 0.942 | 0.859 | 0.687 |
| MStream | | 0.902 | 0.873 | 0.621 | 0.617 |
| MStreamF | | 0.871 | 0.897 | 0.661 | 0.649 |
| DP-BMM | | 0.892 | 0.863 | 0.659 | – |
| DP-BMM-FP | | 0.847 | 0.853 | 0.642 | – |
| OSDM | | **0.907** | 0.939 | 0.419 | 0.390 |
| Rakib et al. (2020b) | | 0.886 | **0.949** | 0.858 | 0.683 |
| EStream-unigram | VM | 0.824 | 0.821 | 0.858* | 0.742* |
| EStream-bigram | | 0.855 | 0.842 | 0.873* | 0.786* |
| EStream-biterm | | 0.863 | 0.852 | **0.889*** | **0.799*** |
| Fast-MStream | | 0.854 | 0.866 | 0.784 | 0.613 |
| Fast-MStreamF | | 0.864 | 0.867 | 0.747 | 0.618 |
| Fast-DP-BMM | | 0.878 | 0.861 | 0.749 | – |
| Fast-DP-BMM-FP | | 0.836 | 0.872 | 0.744 | – |
| Fast-Rakib et al. (2020b) | | 0.865 | 0.888 | 0.835 | 0.653 |
| MStream | | 0.854 | 0.859 | 0.793 | 0.612 |
| MStreamF | | 0.867 | 0.871 | 0.745 | 0.618 |
| DP-BMM | | **0.879** | 0.865 | 0.768 | – |
| DP-BMM-FP | | 0.836 | 0.866 | 0.747 | – |
| OSDM | | 0.859 | 0.846 | 0.790 | 0.447 |
| Rakib et al. (2020b) | | 0.865 | **0.896** | 0.833 | 0.658 |

Table 4.7: Average running times (in seconds) of different methods.

| Clustering Methods | Data Sets | | | |
|---|---|---|---|---|
| | Ns-T | Ts-T | NT | SO-T |
| EStream-unigram | 6 | 20 | 35 | 138 |
| EStream-bigram | 6 | 18 | 36 | 137 |
| EStream-biterm | 8 | 24 | 51 | 153 |
| Fast-MStream | 75 | 165 | 358 | 1290 |
| Fast-MStreamF | 63 | 143 | 311 | 998 |
| Fast-Rakib et al. (2020b) | 23 | 69 | 179 | 513 |
| MStream | 227 | 541 | 820 | 4289 |
| MStreamF | 173 | 295 | 698 | 2137 |
| DP-BMM | 5016 | 12307 | 18220 | – |
| DP-BMM-FP | 880 | 2854 | 4561 | – |
| OSDM | 31 | 196 | 690 | 1882 |
| Rakib et al. (2020b) | 80 | 165 | 585 | 1389 |

state-of-the-art methods on the datasets NT and SO-T as the hyper parameters of the state-of-the-art methods were not tuned on these two datasets.

Among the three variants of EStream, **EStream-biterm** performs better than **EStream-unigram** and **EStream-bigram** on all the datasets. The reason is that, by using biterm features, we can extract sufficient distinctive features for short texts which in turn help us to partition the texts into proper clusters Chen et al. (2020).

***Comparison of Running Time*** The average running times (in seconds) of EStream, the state-of-the-art methods and their corresponding faster versions (i.e., Fast-MStream, Fast-MStreamF, and Fast-Rakib et al. (2020b)) are shown in Table 4.7.

The running time of EStream is several orders of magnitude faster than that of MStream(F), DP-BMM(-FP), and OSDM on all datasets. In addition, we demonstrate that the faster versions of the state-of-the-art methods (based on inverted index Ilic et al. (2014) of the clusters) require significantly less amount of running time than that of the corresponding state-of-the-art methods. The reason is that we do not compute similarity between a text and all the existing clusters while assigning a text to a cluster. Instead we select a specific set of clusters using the features of the text based on inverted index Ilic et al. (2014) and compute similarities between the text and the selected clusters. We store almost twice the number of features than other methods as we use inverted index to select a specific set of clusters for a particular text. We consider this as a small price to pay for the significant improvement in running time of our proposed method (EStream).

Another reason why the running time of MStream(F) and DP-BMM(-FP) is slower than the running time of EStream is that both MStream(F) and DP-BMM(-FP) perform Gibbs sampling Ishwaran and James (2001) several times on a single text so as to assign the text to a cluster which is a time consuming operation. For instance, if there are $N$ number of texts, $K$ number of clusters, $V$ number of words in each text, and Gibbs sampling is performed $I$ times for each text, then the total running time of clustering will be $\mathcal{O}(IKNV)$.

On the other hand, the online clustering module of EStream computes similarity between a text and a selected number of clusters. Thus the online clustering module of EStream selects a constant ($c$) factor of $V$ number of clusters (i.e., $c \times V$ clusters) for each text since each text contains $V$ words and for each word EStream selects approximately $c$ number of clusters. In addition, the online clustering module of EStream requires only one iteration (i.e., $I = 1$) to find clusters for the texts. Therefore the approximate running time of EStream will be $\mathcal{O}(N \times cV \times V) \approx \mathcal{O}(N \times V^2)$[12]. The running time complexity of EStream shows that EStream is significantly faster than MStream(F) and DP-BMM(-FP) in the context of clustering streams of short texts. The reason is that $V^2$ is significantly less than $I \times K \times V$ (i.e., $V^2 \ll I \times K \times V$) because for each text of $V$ words, EStream selects approximately $V$ number of clusters where as MStream(F) and DP-BMM(-FP) select $K$ number of clusters and $V$ is significantly less than $K$ (i.e., $V \ll K$).

Though the running time complexity of EStream is significantly lower in the context of clustering short texts, it may not perform well in the context of clustering long texts as the number of words per text (i.e., $V$) will be relatively larger than that of the short text.

Both DP-BMM and DP-BMM-FP represent each text of $n$ words using $n \times (n - 1)/2$ biterms and store all the biterms of texts that belong to a particular cluster. Therefore, the running time of DP-BMM(-FP) is quite longer than that of other methods.

[12]We ignore the running time of offline clustering module of EStream algorithm since offline clustering is performed on a small fraction of texts in every update-interval.

# Chapter 5

# Finding Duplicate Questions using Clustering

At first, we discuss data preparation for finding duplicate questions. Then we describe how we can find duplicate questions using our clustering method. After that we discuss the experimental results obtained from our duplicate questions finding system.

## 5.1 Data Preparation for Finding Duplicate Questions

To prepare dataset, we downloaded a publicly available data dump of Meta Stack Exchange from archive.org[1]. The data dump contains all the activities around posted questions between June, 2009, and August, 2021. The data dump is composed of a set of XML files containing data about all questions, associated answers, post histories, post links, comments, and votes. We used two files from this set which are **Posts.xml** and **PostLinks.xml**. Each item in Posts.xml represents a single post which can be of different types (e.g., question, answer, and so on). Each item in the PostLinks.xml contains the information about a pair of duplicate questions.

There are about 20 millions of questions posted in StackOverflow on various programming languages such as R, C++, C#, Python, Java, JavaScript and so on. Among them we extract questions on four different programming languages which are R, C#, Python, and Java. For each question we extract the following contents which are QuestionId, Tag, Title, and Body. We use the whole content of the Tag and whole content of the Title of each question. However, for Body, we use the frequent keywords extracted from the body. The reason to use a list of keywords of body is that our method is a short text stream clustering method and it cannot cluster the questions with whole bodies (i.e. long text). To extract the keywords from the body, we use an existing keyword extraction tool called RAKE[2]. We apply RAKE to the body of each question and extract top 10 frequent keywords based on the frequency

[1] `https://archive.org/download/stackexchange`

[2] https://pypi.org/project/rake-nltk/

of the word and its co-occurance with other words in the text where each keyword consists of 2 to 4 words. The detailed statistics of the questions of four languages are shown in Table 5.1.

Table 5.1: Summary of the StackOverflow Questions

| Language | #Questions | Avg. #words/Tag | Avg. #words/Title | Avg. #words/Body |
|---|---|---|---|---|
| R | 312,829 | 2.66 | 9.37 | 22.30 |
| C# | 1,304,920 | 3.30 | 8.84 | 21.68 |
| Python | 1,347,471 | 3.12 | 9.03 | 21.87 |
| Java | 1,592,884 | 3.45 | 9.67 | 22.14 |

To examine how our proposed duplicate finding system works, we create two datasets called training and test set respectively. The test dataset consists of the duplicate questions and training dataset consists of the duplicate and non-duplicate questions. To create the datasets, at first we extract the groups of duplicate questions described in Section 5.1.1. In Section 5.1.2, we discuss how we construct the training and test sets.

### 5.1.1 Extracting Groups of Duplicate Questions

In this work, we aim to find duplicate questions of a given question on a particular language. We consider that duplicate questions are similar to each other and a group of similar questions can form a cluster. In the following we discuss how we extract the clusters of duplicate questions.

We obtain the QuestionId, Tag, Title, and Body of the questions from the file Posts.xml[3] and obtain the information about the duplicate questions from PostLinks.xml. Each item of the PostLinks.xml contains the information about a pair of duplicate questions. For instance, the PostLinks.xml contains the questions A and B if they are duplicate. There are about 20 million questions in Posts.xml and 1 million pair of duplicate questions in PostLinks.xml.

For each pair of questions in PostLinks.xml, we find the base question and the duplicate question. The base question and duplicate question are defined as the oldest and newest question within a pair respectively based on their creation time.

[3]https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede

We obtain the creation time of a question from Posts.xml. To extract a group of duplicate questions we find the duplicate questions for a single base question. In particular a base question along with the corresponding duplicate questions form a group of duplicate questions. For example, if B, C, and D are the duplicate questions for the base question A (i.e., $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$); then A, B, C, and D form a group of duplicate questions. However, a duplicate question (e.g., D) can be a base question of another duplicate question (e.g., E). In this work, we discard the group containing the base question which becomes a duplicate question of another group so as to make sure that no duplicate question can be used as a base question. Thus we keep the group of duplicate questions A, B, C, and D and discard the group of duplicate questions D and E. After obtaining the groups of duplicate questions we label them. In particular, questions in the same group have the same label. The detail statistics of the groups of duplicate questions for different languages are shown in Table 5.2.

Table 5.2: Statistics of the Groups of Duplicate Questions for Different Languages

| Language | *#Clusters* | *Avg.CSize* | *Min.CSize* | *Max.CSize* | *Median.CSize* |
|---|---|---|---|---|---|
| R | 6832 | 3.06 | 2 | 424 | 2 |
| C# | 18954 | 2.64 | 2 | 168 | 2 |
| Python | 21987 | 3.10 | 2 | 562 | 2 |
| Java | 24308 | 3.18 | 2 | 781 | 2 |

The number of groups of duplicate questions (*#Clusters*), average number of questions (*Avg.CSize*), minimum number of questions (*Min.CSize*), maximum number of questions (*Max.CSize*), and median of the number of questions (*Median.CSize*) in the cluster are shown in Table 5.2. Sample Tag, Title, and Body of a group of duplicate questions for Java language is shown in Table 5.3.

Table 5.3: Sample Content (Tag, Title, and Body) of a Group of Duplicate Questions for Java

| Content Type | Content (PostId) |
|---|---|
| Tag | – <java><lucene><spring-batch> (39701195)<br>– <java><serialization><lucene><spring-batch> (39717584) |
| Title | – Should I keep Lucene IndexWriter open for entire indexing or close after each document addition? (39701195)<br>– Can we make Lucene IndexWriter serializable for ExecutionContext of Spring Batch? (39717584) |
| Body | – create code indexwriter code code indexwriter code code itemprocessor code spring batch job searching please suggest find duplicate documents opening index writer step completion also lucene indexer step document addition slow (39701195)<br>– keep lucene indexwriter open add code indexwriter code code indexwriter code serialize non serializable code executioncontext code add constructor shown code super code code fields access code super class (39717584) |

### 5.1.2 Training and Test Set Generation

We generate training and test set using the Tags, Titles, and Bodies of the questions respectively for a particular language. In particular, for each language we generate three training and test sets; one by Tags and other two by Titles and Bodies respectively.

To generate training and test set, we split each group of duplicate questions. From each group we add the base question to the test set and add the rest of the duplicate questions to the training set. Therefore the final test set contains the base questions from the groups of duplicate questions. The final training set is constructed using the rest of duplicate questions from the groups in conjunction with the ungrouped questions. The detail statistics of the training and test set for each programming language are shown in Table 5.4.

Table 5.4: Statistics of Training and Test Set

| Language | #Training Instances | #Test Instances |
|---|---|---|
| R | 305,997 | 6,832 |
| C# | 1,285,966 | 18,954 |
| Python | 1,325,484 | 21,987 |
| Java | 1,568,576 | 24,308 |

The number of questions in the Training and Test set of R language are 302410 and 6485 respectively as shown in Table 5.4 implying that we use 6485 base questions extracted from 6485 clusters as test set and 302410 questions consisting of duplicate and non-duplicate questions as training set.

## 5.2 Finding Duplicate Question by Clustering

In this section, we discuss how we find a duplicate question for a given question using the clusters obtained by our dynamic clustering method (i.e., stream clustering) along with static clustering Rakib et al. (2021). Since questions are continuously being posted in Stack Overflow, our text stream clustering method clusters them one by one as they arrive. At any given time, if an user provides a question to our duplicate question finding system, it will use the existing clusters obtained by our proposed text stream clustering method and try to find a duplicate question of the given using the clusters. Figure 5.1 shows the architecture of our duplicate question finding system using stream clustering.

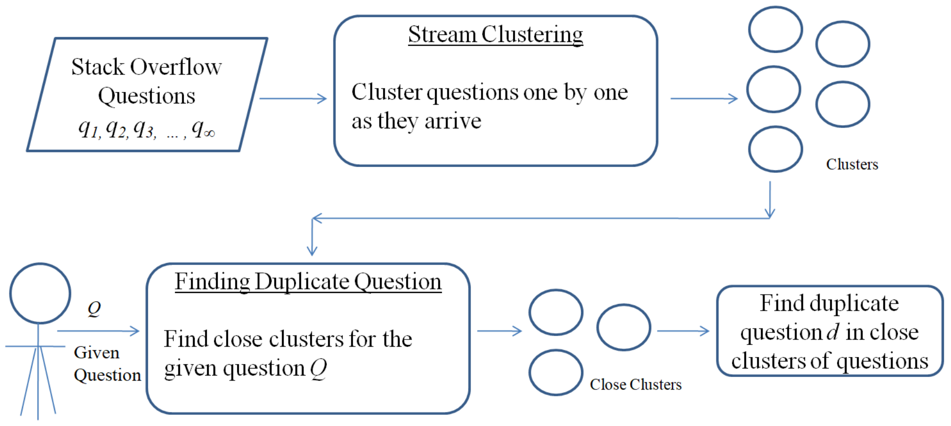

Stack Overflow Questions
$q_1, q_2, q_3, \ldots, q_\infty$
Stream Clustering
Cluster questions one by one as they arrive
Clusters
$Q$
Given Question
Finding Duplicate Question
Find close clusters for the given question $Q$
Close Clusters
Find duplicate question $d$ in close clusters of questions

Figure 5.1: Proposed duplicate question finding system using stream clustering.

It is shown in Figure 5.1 that our stream clustering method clusters questions one by one as they arrive. Whenever a question will be given to our duplicate finding system it will find the close clusters of the given question in terms of similarity between the question and the clusters. Then our duplicate finding system finds the duplicate question by searching a set of questions contained in those close clusters.

### 5.2.1 Finding Close Clusters

In the following Algorithm 3, we discuss how we find close clusters of a given question. The input of the algorithm are the clusters of questions obtained by our stream clustering method and a given question for which we find a duplicate question. The output of the algorithm is the close clusters of the given question.

**Algorithm 3** Find Close Clusters of a Given Question

```
Input: Clusters of Questions obtained by Stream Clustering: $C_1, C_2, C_3, ..., C_m$; Given Question: $Q$
Output: Close Clusters of given question: $D_1, D_2, ..., D_n$
1:  $D = \phi$, list of close clusters
2:  $S = \phi$, list of similarity values
3:  for $C_i$ in $C_1 ... C_m$ do
4:      Compute similarity ($s_i$) between $Q$ and the cluster $C_i$
5:      Add $s_i$ to $S$
6:  end for
7:  Compute the mean ($\mu$) and standard deviation ($\sigma$) of the similarities in $S$
8:  for $C_i$ in $C_1 ... C_m$ do
9:      $s_i = S[i]$
10:     if $s_i > \mu + \sigma$ then
11:         Add $C_i$ to $D$
12:     end if
13: end for
14: return $D$
```

Given $m$ clusters of StackOverflow questions ($C_1, C_2, C_3, ..., C_m$), and a question $Q$. The above algorithm finds the $n$ close clusters ($D_1, D_2, ..., D_n$) for that given question. To obtain the close clusters, the algorithm computes similarity between the question $Q$ and existing $m$ clusters as described in the following.

***Similarity Computation for Finding Close Clusters*** To find close clusters of a given question $Q$, we use either semantic or lexical similarity. We use semantic similarity (as described in Section 4.2.2) to find close clusters, when at least one word of the given question $Q$ (after removing stop words) do not appear in any of the existing clusters $(C_1, C_2, ..., C_m)$. Otherwise we use lexical similarity (as defined in Equation 4.1). For example, "How to make an R object immutable" is a duplicate question of the given question "Declaring a Const Variable in R". After removing stop words from the given question (i.e., "Declaring Const Variable"), we observe that the word "const" does not appear in any of the existing clusters. Therefore, we compute semantic similarity between the given question and existing clusters to obtain close clusters so as to find the corresponding duplicate question.

### 5.2.2 Finding Duplicate Question using Clusters

Using the close clusters obtained from Algorithm 3, we find the corresponding duplicate question for the given question $Q$ as shown in Algorithm 4 and described next.

**Algorithm 4** Find Duplicate Question of a Given Question

```
Input: Clusters of Similar Questions: $D_1, D_2, ..., D_n$;   Question from Test Set: $Q$
Output: Corresponding Duplicate Question
1: for $D_i$ in $D_1 ... D_n$ do
2:     for $T$ in $D_i$ do
3:         $t_{label}$=label of question $T$
4:         $q_{label}$=label of question $Q$
5:         if $t_{label} = q_{label}$ then
6:             return $T$
7:         end if
8:     end for
9: end for
```

For each close cluster $D_i$, we iterate through the questions of that cluster. After that for each question $T$ in $D_i$, we extract the label of the question $T$ (how we obtain label of question discussed in Section 5.1.1) denoted as $t_{label}$. Following that we compare the label of the question $Q$ (denoted as $q_{label}$) with that of $T$. If both labels are same, we consider $T$ as the duplicate question of the given question $Q$.

## 5.3 Experimental Study on Finding Duplicate Questions

In this section, we perform empirical study on finding duplicate questions using our proposed duplicate question finding system. We compare the result of our proposed system with that of an information retrieval system called Lucene Balipa and Ramasamy (2015) which is used as a baseline duplicate question finding system. Lucene is an inverted index Ilic et al. (2014) based search engine that finds the similar documents of a given document within the corpus.

To perform empirical study using our system, at first we cluster the Stack Overflow questions in training set using our short text stream clustering method. Then we try to find the duplicate question of a given question in test set using the duplicate question finding system as described in Section 5.2. To perform empirical study using Lucene, at first we index the Stack Overflow questions using Lucene, then we retrieve similar questions of a given question using Lucene and finally we try to find a duplicate question of the question among the similar questions. In Section 5.3.2, we discuss the experimental results obtained by our proposed duplication question finding system and by Lucene.

### 5.3.1 Statistics of Clustering Stack Overflow Questions in Training Set

We cluster the Stack Overflow questions in training set using our proposed short text stream clustering method using Tag, Title, or Body of each question respectively. In particular, when we use Tag, we cluster the questions using only the Tags of the questions for a particular language. The detail statistics of clustering Stack Oveflow questions of different languages (i.e., R, C#, Python, and Java) using Tag, Title, or Body are shown in Table 5.5.

Table 5.5: Statistics of clustering questions in Training Set using Proposed Stream Clustering Method for Different Languages

| Language | #Training instances | Question Content | #Clusters | Min/Max/Avg. ClusterSize |
|---|---|---|---|---|
| R | 308,567 | Tag | 978 | 12/1906/315.50 |
| | | Title | 15451 | 1/572/19.97 |
| | | Body | 1151 | 1/701/268.08 |
| C# | 1,295,976 | Tag | 1168 | 72/1260/1109.56 |
| | | Title | 23575 | 1/361/54.97 |
| | | Body | 8122 | 1/1167/159.56 |
| Python | 1,336,032 | Tag | 1297 | 89/1493/1030.09 |
| | | Title | 26919 | 1/1084/49.63 |
| | | Body | 9643 | 1/1051/138.54 |
| Java | 1,580,802 | Tag | 1569 | 127/1698/1007.52 |
| | | Title | 45901 | 1/1356/34.43 |
| | | Body | 14701 | 1/1207/107.53 |

The number of training instances used in our experiment for R language is 308,567. The number of clusters (#Clusters), minimum (Min), maximum (Max), and average (Avg.) size of the clusters are shown in Table 5.5. For example, after clustering 308,567 questions using Tags, we obtain 978 clusters where the Min, Max, and Avg. size of the clusters are 12, 1906, and 315.50 respectively. The size of a cluster is determined based on the number of questions in the cluster.

## 5.3.2 Experimental Results on Finding Duplicate Questions

In this section, we discuss the experimental results on finding duplicate questions by our proposed duplicate finding system and by Lucene Balipa and Ramasamy (2015). To find duplicate question by our method, at first, we find the close clusters of the given question using Algorithm 3. The close clusters contain the similar content as the content of the given question. Using the questions of the close clusters, we find duplicate question for the given using Algorithm 4. To find duplicate question by Lucene, the questions in the training set are ranked based on the similarity values (in descending order) between the given question and the questions in training set. Then top ranked ten thousand questions are selected. After that we find a corresponding duplicate question of the given question among those top ranked questions.

The experimental results of finding duplicate question for different languages (i.e., R, C#, Python, and Java) by Our Method and by Lucene are shown in Table 5.6.

Table 5.6: Experimental results of finding duplicate question for different languages by Our Method and Lucene

| Language | #Testing instances | Question Content | Our Method | | | Lucene | | |
|---|---|---|---|---|---|---|---|---|
| | | | #Duplicate Found | Running Time (Secs) | Min/Max/ Median SRank | #Duplicate Found | Running Time (Secs) | Min/Max/ Median SRank |
| R | 1000 | Tag | 502 | 135 | 1/ 4061/ 331 | 504 | 29 | 1/ 9849/ 547.5 |
| | | Title | 763 | 149 | 1/ 6154/ 108 | 692 | 52 | 1/ 9954/ 139.5 |
| | | Body | 533 | 451 | 1/ 7218/ 495 | 538 | 136 | 1/ 9920/ 551.5 |
| C# | 1000 | Tag | 541 | 203 | 1/ 5375/ 309 | 565 | 67 | 1/ 9854/ 355 |
| | | Title | 827 | 187 | 1/ 5257/ 48 | 749 | 54 | 1/ 9751/ 74 |
| | | Body | 552 | 503 | 1/ 7303/ 139 | 545 | 138 | 1/ 9956/ 154 |
| Python | 1000 | Tag | 559 | 197 | 1/ 6495/ 331 | 577 | 65 | 1/ 9792/ 363 |
| | | Title | 755 | 204 | 1/ 4829/ 95 | 708 | 63 | 1/ 9768/ 123 |
| | | Body | 509 | 521 | 1/ 6703/ 314 | 516 | 162 | 1/ 9903/ 352 |
| Java | 1000 | Tag | 639 | 213 | 1/ 6379/ 273 | 657 | 69 | 1/ 9605/ 306 |
| | | Title | 853 | 232 | 1/ 6341/ 68.5 | 755 | 53 | 1/ 9921/ 80 |
| | | Body | 546 | 602 | 1/ 6761/ 163 | 551 | 151 | 1/ 9962/ 205 |

To perform this study, we randomly select 1000 questions from test set of a particular programming language and find duplicate questions of them using the Tags, Titles, and Bodies of the questions. For example, there are 6832 questions in test set for the programming language R. Among these 6832 questions we randomly select 1000 questions to find their duplicate questions. Using the Tags of the questions we are able to find 474 duplicate questions for 1000 given questions using Algorithm 4. Likewise we are able to find 763 duplicate questions for 1000 given questions using the Titles of the questions. Using Lucene, we find 504 duplicate questions using the Tags of the questions. Likewise by Lucene, we find 692 duplicate questions using the Titles of the questions.

The experimental results in Table 5.6 show that our proposed duplicate question finding system outperforms Lucene in the context of finding duplicate questions on all four programming languages using question titles. The reason is that question titles contain discriminative features that help us to organize the questions in proper clusters which in turn allow us to select the clusters that share similar titles as the title of the given question. However, using tags we do not outperform Lucene. The reason is that most of the questions have same tag; therefore our stream clustering method puts the questions with same tags into same cluster although those questions are on different topics. Using body we do not perform better than Lucene as we select a list of keywords from the bodies. Therefore some content in the body might be missing because of not taking into account the whole content of body which in turn may produce improper clusters and may affect on finding a duplicate question of the given question.

The running time (in seconds) for finding duplicate questions by our method and by Lucene is shown in Table 5.6. For example, the running time of our method and Lucene to find duplicate questions for 1000 given questions for R language using Title are 149 and 52 seconds respectively. We observe that the running time of Lucene is several order of magnitude faster than that of our method for any programming language. The reason is that we used the Lucene implemented in Java where as our method is developed in Python and we know that the running time of Java is three to five times faster than that of Python for similar kind of program since Java is a compiler based language and Python is an interpreter based language McMaster et al. (2017).

In Table 5.6, we also show the statistics (minimum (Min), maximum (Max), and median) of Search Ranks (SRank) in the context of finding duplicate questions. ***Search Rank (SRank)*** *is defined as the ranking of a duplicate question for a given question in terms of content similarity between two questions.* The lower the SRank of a duplicate question, the more similar to the given question and vice versa. The Min, Max, and Median SRank of finding duplicate questions (using our method) for R using Title are 1, 6154, and 108 respectively. Likewise the Min, Max, and Median SRank of finding duplicate questions (using Lucene) for R using Title are 1, 9954, and 139.5 respectively.

We observe that the SRank of our duplicate finding method is lower than that of Lucene for any programming language. Let us briefly describe why SRank of our method is lower than that of Lucene. Our proposed duplicate question finding system computes similarity between the given question and the clusters of questions instead of each individual question which limits the number of similarity computation for a given question Manning et al. (2008). On the contrary, Lucene needs to compute similarity between the given question and each individual question. By clustering the questions, we organize the questions with similar content into the same cluster. Our duplicate question finding system selects a set of close clusters for the given question which helps to find a duplicate question by searching a subset of the whole collection of questions. Therefore, the Maximum (Max) and Median Search Rank (SRank) of our method is lower than that of Lucene for any programming language used in our experiments as shown in Table 5.6.

# Chapter 6

# Conclusion and Future Work

In this thesis, we developed static and dynamic clustering methods for short texts. Our first static clustering method improves clustering performance by sparsifying similarity matrix using our proposed similarity distribution based sparsification method. Our second static clustering method is based on iterative classification that enhances the initial clustering of short texts obtained using an arbitrary clustering method (e.g., k-means, k-means–, hierarchical agglomerative clustering) by removing outliers and reclassifying them to appropriate clusters until the cluster partitions stabilize. This method is a generic clustering enhancement approach where various classification algorithms, initial clustering and number of clusters can be easily integrated. Experimental results demonstrate that our static clustering methods outperform state-of-the-art short text clustering methods on several datasets in terms of clustering accuracy and running time.

In our first dynamic clustering method, we have demonstrated that building an efficient clustering model using initial cluster assignments based on frequent word pairs and removing outliers from clusters, outperforms the state-of-the-art short text stream clustering methods in terms of clustering accuracy. Our second dynamic clustering method significantly reduces the running time than that of the state-of-the-art methods by limiting the number of similarity computations between the texts and clusters using inverted index. It also improves clustering accuracy by enhancing cluster partitions using our clustering enhancement method.

Using our static and dynamic clustering methods of short texts, we developed a duplicate question finding system that finds duplicate questions in Stack Overflow based on the clusters produced by our clustering methods. Experimental results demonstrate that using the titles of questions, our duplicate question finding system finds more duplicate questions than Lucene. However using the tags and bodies of questions Lucene finds more duplicate questions than our proposed duplicate finding system.

In the future, we will use context based word embedding (e.g., BERT Devlin et al. (2018)) instead of average-vector based word embedding (as described in Section 4.1.5) for the representation of short texts as context based embedding takes into account the order of words in a text whereas average-vector based embedding ignores the order of words in a text. For instance, we have two short texts: “boat house” and “house boat”. The average-vector based embeddings for these two texts will be the same. However the embeddings of these two texts should be different as the meaning of these two texts are different from each other. The “boat house” means a house for sheltering boats, whereas “house boat” refers to a boat that serves as a house. Therefore we will use context based word embedding for the representations of short texts in our static and dynamic short text clustering methods since context based embedding generates embedding based on the order of words in a text which in turn may produce better clustering results.

We will investigate the impact of hubness in the context of clustering static collection of texts Feldbauer and Flexer (2019). Hubs are the items that frequently appear among the k-nearest neighbors of other items. Since hubs have higher similarity with most of the items, it may be difficult for a clustering algorithm to properly partition the data into expected clusters due to the hubs. Therefore in the future, we will take into account the hubness information of an item while sparsifying a matrix using our similarity distribution based sparsification method Rakib et al. (2018). In the sparsified matrix, we will keep a similarity value if the retaining criteria is fulfilled by our sparsification method in conjunction with the hubness retaining criteria (e.g., the similarity value is less than a certain hubness threshold).

In Stack Overflow, questions are continuously being posted on different topics over time that need to be answered in a timely manner. In order to answer the questions, Stack Overflow needs to find users who have expertise on similar topics Riahi et al. (2012). One simple approach to find these users is to look for the users who already answered similar types of questions. Then Stack Overflow can notify those selected users to answer the questions. Clustering can be a viable tool to recommend a limited set of expert users to answer the questions Roy et al. (2021). Since the questions are continuously being posted over time, we plan to use our dynamic clustering method to cluster the questions Rakib et al. (2021) since dynamic clustering methods are adaptive with the changes in the collection of data (e.g., a new question is posted). Whenever a new question is posted, we will find the clusters containing similar content as the content of the question. After that we can notify the users who previously answered the questions of those clusters.

In this research, we use our dynamic clustering method to cluster a couple of millions of Stack Overflow questions to find duplicate questions. In order to perform clustering, we compute similarity between a text and a number of clusters using a single physical computing unit (i.e., a processor) which prevents us from obtaining the clusters of questions within a reasonable amount of time. Therefore, we plan to use multiple physical computing units (as required) so that we can perform multiple similarity computations at the same time. For instance, if we need to compute similarity between a text and one hundred clusters, then we can use one hundred physical computing units to compute similarity between that text and one hundred clusters at the same time which in turn can significantly reduce the running time of text clustering.

We plan to use our dynamic clustering methods of short texts to find linked questions in Stack Overflow. A question is called linked question if the question is referenced in the solution of a given question. We will also explore more programming languages (e.g., JavaScript, PhP, Perl, Ruby and so on) in the context of finding duplicate questions as well as linked questions to investigate the robustness of our static and dynamic clustering methods of short texts.

# Bibliography

Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 81–92.

Aggarwal, C. C. and Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications*. Chapman & Hall, 1st edition.

Aggarwal, C. C. and Zhai, C. (2012). *A Survey of Text Clustering Algorithms*, pages 77–128. Springer US, Boston, MA.

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., and Schneider, K. A. (2016). Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, page 402–412, New York, NY, USA. Association for Computing Machinery.

Balipa, M. and Ramasamy, B. (2015). Search engine using Apache Lucene. *International Journal of Computer Applications*, 127:27–30.

Banerjee, S., Ramanathan, K., and Gupta, A. (2007). Clustering short texts using wikipedia. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, page 787–788, New York, NY, USA. Association for Computing Machinery.

Blei, D. M. and Lafferty, J. D. (2006). Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 113–120, New York, USA. ACM.

Brants, T. and Franz, A. (2006). Web 1t 5-gram corpus version 1.1. *Linguistic Data Consortium*.

Carnein, M. and Trautmann, H. (2019). Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering*, 61(3):277–297.

Chen, J., Gong, Z., and Liu, W. (2020). A dirichlet process biterm-based mixture model for short text stream clustering. *Applied Intelligence*, 50(5):1609–1619.

Chen, W.-Y., Song, Y., Bai, H., Lin, C.-J., and Chang, E. Y. (2011). Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586.

Cheng, X., Yan, X., Lan, Y., and Guo, J. (2014). Btm: Topic modeling over short texts. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2928–2941.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Couronné, R., Probst, P., and Boulesteix, A.-L. (2018). Random forest versus logistic regression: a large-scale benchmark experiment. *BMC Bioinformatics*, 19(1):270.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA.

da Silva, R. F. G., Roy, C. K., Rahman, M. M., Schneider, K. A., Paixão, K., Dantas, C. E. d. C., and Maia, M. d. A. (2020). Crokage: effective solution recommendation for programming tasks by leveraging crowd knowledge. *Empirical Software Engineering*, 25(6):4707–4758.

Dahiru, T. (2008). P - value, a true test of statistical significance? a cautionary note. *Annals of Ibadan postgraduate medicine*, 6(1):21–26.

De Abreu Lopes, P. and De Arruda Camargo, H. (2017). Fuzzstream: Fuzzy data stream clustering based on the online-offline framework. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. cite arxiv:1810.04805Comment: 13 pages.

Du, Y., Pan, Y., Wang, C., and Ji, J. (2018). Biomedical semantic indexing by deep neural network with multi-task learning. *BMC Bioinformatics*, 19(20):502.

Erk, K. (2012). Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6:635–653.

Feldbauer, R. and Flexer, A. (2019). A comprehensive empirical comparison of hubness reduction in high-dimensional spaces. *Knowledge and Information Systems*, 59(1):137–166.

Gollub, T. and Stein, B. (2010). Unsupervised sparsification of similarity graphs. In Locarek-Junge, H. and Weihs, C., editors, *Classification as a Tool for Research*, pages 71–79, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hadifar, A., Sterckx, L., Demeester, T., and Develder, C. (2019). A self-training approach for short text clustering. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 194–199, Florence, Italy. Association for Computational Linguistics.

Ilic, M., Spalevic, P., and Veinovic, M. (2014). Inverted index search in data mining. In *2014 22nd Telecommunications Forum Telfor (TELFOR)*, pages 943–946.

Ishwaran, H. and James, L. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173.

Islam, A., Milios, E., and Kešelj, V. (2012). Text similarity using google tri-grams. In *Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 312–317, Berlin, Heidelberg. Springer-Verlag.

Kalogeratos, A., Zagorisios, P., and Likas, A. (2016). Improving text stream clustering using term burstiness and co-burstiness. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence*, SETN '16, New York, NY, USA. Association for Computing Machinery.

Kanj, S., Brüls, T., and Gazut, S. (2016). Shared nearest neighbor clustering in a locality sensitive hashing framework.

Kaptein, M. and van den Heuvel, E. (2022). *Statistics for Data Scientists: An Introduction to Probability, Statistics, and Data Analysis.* Undergraduate Topics in Computer Science. Springer International Publishing.

Kumar, J., Shao, J., Uddin, S., and Ali, W. (2020). An online semantic-enhanced Dirichlet model for short text stream clustering. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 766–776, Online.

Kumar, V. (2000). An introduction to cluster analysis for data mining. Technical report, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN.

Li, P., Chen, Z., Hu, Y., Leng, Y., and Li, Q. (2018). A weighted fuzzy c-means clustering algorithm for incomplete big sensor data. In Li, J., Ma, H., Li, K., Cui, L., Sun, L., Zhao, Z., and Wang, X., editors, *Wireless Sensor Networks*, pages 55–63, Singapore. Springer Singapore.

Liang, S., Yilmaz, E., and Kanoulas, E. (2016). Dynamic clustering of streaming short documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 995–1004.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363.

Mahdiraji, A. R. (2009). Clustering data stream: A survey of algorithms. *International Journal of Knowledge-based and Intelligent Engineering*, 13(2):39–44.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK.

McMaster, K., Sambasivam, S., Rague, B. W., and Wolthuis, S. (2017). Java vs. python coverage of introductory programming concepts: A textbook analysis. *Information Systems Education Journal*, 15:4–13.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Müllner, D. (2013). fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, 53(9):1–18.

Nguyen, H.-L., Woon, Y.-K., and Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowledge and Information Systems*, 45(3):535–569.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Rakib, M. R. H., Jankowska, M., Zeh, N., and Milios, E. (2018). Improving short text clustering by similarity matrix sparsification. In *Proceedings of the ACM Symposium on Document Engineering 2018*, New York, NY, USA.

Rakib, M. R. H., Zeh, N., Jankowska, M., and Milios, E. (2020a). Enhancement of short text clustering by iterative classification. In Métais, E., Meziane, F., Horacek, H., and Cimiano, P., editors, *Natural Language Processing and Information Systems*, pages 105–117, Cham. Springer International Publishing.

Rakib, M. R. H., Zeh, N., and Milios, E. (2020b). Short text stream clustering via frequent word pairs and reassignment of outliers to clusters. DocEng '20, New York, NY, USA. Association for Computing Machinery.

Rakib, M. R. H., Zeh, N., and Milios, E. (2021). Efficient clustering of short text streams using online-offline clustering. In *Proceedings of the 21st ACM Symposium on Document Engineering*, DocEng '21, New York, NY, USA. Association for Computing Machinery.

Riahi, F., Zolaktaf, Z., Shafiei, M., and Milios, E. (2012). Finding expert users in community question answering. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, page 791–798, New York, NY, USA. Association for Computing Machinery.

Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for idf. *J. Documentation*, 60:503–520.

Roy, P. K., Jain, A., Ahmad, Z., and Singh, J. P. (2021). Identifying expert users on question answering sites. In Goyal, D., Bălaş, V. E., Mukherjee, A., Hugo C. de Albuquerque, V., and Gupta, A. K., editors, *Information Management and Machine Intelligence*, pages 285–291, Singapore. Springer Singapore.

Shekhar, S., Lu, C.-T., and Zhang, P. (2003). A unified approach to detecting spatial outliers. *GeoInformatica*, 7(2):139–166.

Shou, L., Wang, Z., Chen, K., and Chen, G. (2013). Sumblr: Continuous summarization of evolving tweet streams. In *Proceedings of the 36th International ACM SIGIR Conference on Information Retrieval*, pages 533–542.

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. d., and Gama, J. a. (2013). Data stream clustering: A survey. 46(1).

Thada, V. and Jaglan, D. (2013). Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. *International Journal of Innovations in Engineering and Technology*, 2:202–205.

Umaña-Hermosilla, B., de la Fuente-Mella, H., Elórtegui-Gómez, C., and Fonseca-Fuentes, M. (2020). Multinomial logistic regression to estimate and predict the perceptions of individuals and companies in the face of the covid-19 pandemic in the Ñuble region, chile. *Sustainability*, 12(22).

Wang, L., Zhang, L., and Jiang, J. (2020). Duplicate question detection with deep learning in stack overflow. *IEEE Access*, 8:25964–25975.

Xu, J., Xu, B., Wang, P., Zheng, S., Tian, G., Zhao, J., and Xu, B. (2017). Self-taught convolutional neural networks for short text clustering. *Neural Networks*, 88:22–31.

Yin, J., Chao, D., Liu, Z., Zhang, W., Yu, X., and Wang, J. (2018). Model-based clustering of short text streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2634–2642.

Yin, J. and Wang, J. (2016). A model-based approach for text clustering with outlier detection. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 625–636.

Zhang, W. E., Sheng, Q. Z., Lau, J. H., and Abebe, E. (2017a). Detecting duplicate posts in programming qa communities via latent semantics and association rules. In *Proceedings of the 26th International Conference on World Wide Web*, page 1221–1229.

Zhang, X., Zhao, J., and LeCun, Y. (2015a). Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA. MIT Press.

Zhang, Y., Chu, G., Li, P., Hu, X., and Wu, X. (2017b). Three-layer concept drifting detection in text data streams. *Neurocomputing*, 260:393–403.

Zhang, Y., Lo, D., Xia, X., and Sun, J.-L. (2015b). Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30(5):981–997.

Zheng, C., Qian, S., Cao, W., and Wong, H. (2017). Locality-sensitive term weighting for short text clustering. In *Neural Information Processing*, pages 434–444.

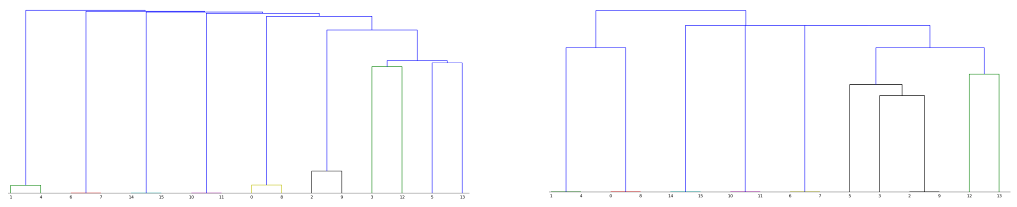
Zheng, C. T., Liu, C., and Wong, H. S. (2018). Corpus-based topic diffusion for short text clustering. *Neurocomput.*, 275(C):2444–2458.

# Appendix A

## Impact of Similarity Matrix Sparsification on Hierarchical Clustering

To investigate the impact of hierarchical clustering on dense and sparse matrix, we took a small subset of the original dataset so as to observe the merging steps through dendrogram Aggarwal and Zhai (2012) during clustering. The dendrogram is a tree diagram showing the hierarchical relationship between objects. The small subset we choose for clustering consisting of 16 texts distributed into 8 clusters (i.e., each cluster contains 2 texts) of the dataset SearchSnippet.

The hierarchical agglomerative clustering starts with each document in its own cluster and repeatedly merges pairs of most similar clusters until only $k$ (the desired numbers of clusters) clusters remain. The dendrograms obtained from clustering dense and sparse similarity matrix for 16 texts respectively are shown in Figure A.1. Dense matrix contains similarity values for each pairs of texts. On the contrary, sparse matrix keeps similarity values for some pairs of texts and removes the rest. To obtain sparse similarity matrix we use our similarity distribution based sparsification method Rakib et al. (2018).



(a) Dendrogram for dense matrix (b) Dendrogram for sparse matrix

Figure A.1: Dendrograms for clustering similarity matrix

The texts (i.e., documents) we cluster are indexed by 0 to 15 where the texts 0 and 1, 2 and 3, ... , 12 and 13, 14 and 15 are in same cluster. Sample Documents with indices and corresponding cluster topics are shown in the following Table A.

Table A.1: Sample texts from small subset with indices and corresponding cluster topics for the dendrogram in Figure A.1

| Index | Text | Cluster topic |
|---|---|---|
| 3 | lucasfilm marin county california lucasfilm independent production companies | culture-arts-entertainment |
| 12 | epidemic sommaire ral art zoyd dumb lepage saup granular synthesis jean michel bruy lydie jean dit pannel dangereuses visions | health |
| 13 | monographs iarc eng monographs volume volume tobacco smoke involuntary smoking tobacco smoking causally cancer oesophagus squamous cell tobacco smoking stomach cancer causal dangereuses | health |

Let us analyze the dendrograms for dense and sparse matrices in Figure A.1 respectively to investigate the impact of sparsification on hierarchical clustering. One major difference between these two dendrograms is the merging of documents 12 and 13 which are in the same cluster in source dataset. When we cluster dense matrix we observe that the text 3 and 12 are being merged together because they have the closest similarity between them in dense matrix. That means, 12 is the most similar document for document 3 in dense matrix. However, in source dataset, the most similar document for document 12 is document 13. When we sparsify the dense matrix using our similarity distribution based method, we keep similarity between document 12 and 13 and remove other similarity values for document 12 and 13 as our sparsification method keeps similarity values for a pair of texts if they are among the most similar texts of each other. By analyzing the similarity values we find that 3 is not within the most similar document of document 12. Therefore in sparse matrix we remove the similarity value between document 3 and 12 which in turn helps hierarchical clustering to merge text 12 and 13 into a single cluster (as shown in Figure A.1 (b)) so as to achieve better clustering performance.

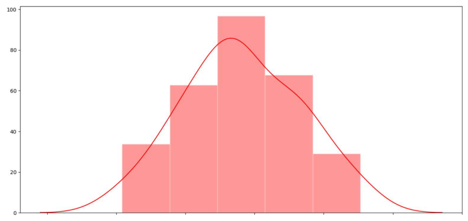
# Appendix B

## Justification of Similarity Threshold ($\mu + \sigma$) for Assigning Texts to Clusters

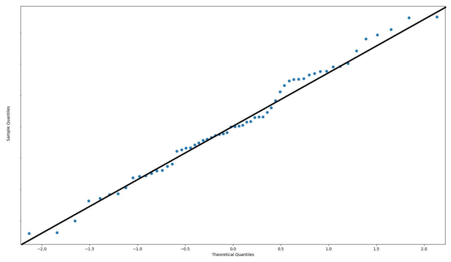
At first we briefly discuss the purpose of similarity threshold in the context of dynamic clustering of texts (i.e., stream clustering). Later, we justify why we use $\mu + \sigma$ as the similarity threshold for assigning texts to clusters.

In stream clustering, data arrives over time. Therefore, the amount of data to be clustered and number of clusters to be produced are unknown. Whenever a new data arrives, either it will be merged with an existing cluster or a new cluster will be created containing the new data based on the similarity between the new data and existing clusters. To quantify how similar a data is with an existing cluster, we use similarity threshold that ultimately determines whether a new cluster will be created or not.

In our short text stream clustering methods Rakib et al. (2020b, 2021), we assign a text to a cluster when the similarity between them is higher comparing to other clusters as well as greater than the similarity threshold. To determine similarity threshold, we compute mean ($\mu$) and standard deviation ($\sigma$) of the similarities between the text and clusters and use $\mu + \sigma$ as the similarity threshold. If the similarity between a text and cluster is greater than $\mu + \sigma$ and the similarity is higher comparing to other clusters, then the text will be assigned to that cluster implying that the text is highly similar to that particular cluster. Now the question arises, why we use $\mu + \sigma$ as the similarity threshold. The reason is that in normal distribution, most of the values tend to be around their center (mean) (i.e., most of them exist within their $\mu \pm \sigma$) and higher values tend to be above $\mu + \sigma$ implying that clusters having similarities greater than $\mu + \sigma$ contain very much similar content with that of an upcoming text. Since we are using $\mu + \sigma$ as similarity threshold, we perform empirical study to investigate whether the similarities between a text and the clusters are normally distributed or not. We observed that approximately for 56 percent of texts the similarities between the texts and clusters are normally distributed[1]. Sample distributions (using histogram and quantile-quantile plot) of normally and not normally distributed similarity values are shown in Figure B.1 and Figure B.2 respectively and described below.

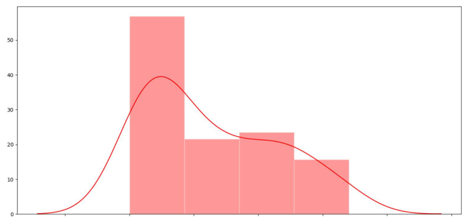


(a) Distribution of similarity values using histogram

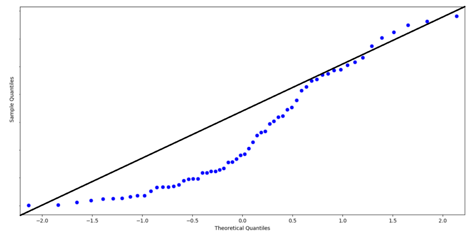


(b) Distribution of similarity values using quantile-quantile plot

Figure B.1: Distribution of normally distributed similarity values



(a) Distribution of similarity values using histogram



(b) Distribution of similarity values using quantile-quantile plot

Figure B.2: Distribution of not normally distributed similarity values

The histogram in Figure B.1(a) shows that the similarity values are normally distributed as most of the values tend to be in the center of the curve (i.e., highest pick at center along y axis) and rest of the values spread equally in the left and right direction from center. In addition to histogram, we have used another type of graphical plot called quantile-quantile[2] plot (q-q plot) Kaptein and van den Heuvel (2022) to show whether the similarity values are normally distributed or not. The q-q plot is a graphical representation for determining if two sets of data come from similar distribution[3]. In our experiment, we consider that our first dataset comes from

[1]We used built-in normality test library (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html) to check whether a set of similarities are normally distributed or not.

[2]By quantile, we mean a fraction of values below the given value.

[3]https://www.itl.nist.gov/div898/handbook/eda/section3/qqplot.htm

an ideal normal distribution (called theoretical distribution) of zero mean and one standard deviation and other dataset comes from our experimental results comprising of similarity values between the text and clusters. If we plot quantiles for a set of normally distributed values, it falls along the 45 degree with x axis which in turn form a straight line called theoretical quantile line.

To verify whether the similarity values are normally distributed, we compute quantiles of similarity values and plot them along y axis against the theoretical quantiles along x axis as shown in Figure B.1(b) and Figure B.2(b) respectively. The quantile-quantile plot in Figure B.1(b) shows that most of the points plotted on the graph lies on a straight line implying that there is small variance between the theoretical quantile line (i.e., at 45 degree with y axis) and sample quantile line, which is the principle concept of normally distributed quantile-quantile plot Kaptein and van den Heuvel (2022). To better understand the difference between normally and not normally distributed similarity values, we plot the histogram and quantile-quantile plot of not normally distributed values as shown in Figure B.2(a) and B.2(b) respectively. The histogram of the similarity values is right skewed as shown in Figure B.2(a) implying that most samples lie on the right side of curve and left side of the curve contains very few samples, which is also reflected in the quantile-quantile plot of Figure B.2(b) since in this plot we observe larger variance between the theoretical quantile line and sample quantile line at the left side of this plot.