Scalable Topic Detection Approaches from Twitter Streams

by

Rania Ibrahim

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Real time topic detection in Twitter streams is an important task that helps discovering natural disasters in a real time from users' posts and helps political parties and companies understand users' opinions and needs. In 2014 the number of active users on Twitter is reported to be more than 288 million users who are posting around 500 million tweets daily. Therefore, detecting topics from Twitter streams in a real time becomes a challenging task that needs scalable and efficient techniques to handle this large amount of data. In this work, we scale an Exemplar-based technique that detects topics from Twitter streams, where each of the detected topics is represented by one tweet (i.e, exemplar). Using exemplar tweets to represent the detected topics, makes these topics easier to interpret as opposed to representing them by uncorrelated terms as in other topic detection algorithms. The approach is implemented using Apache Giraph and is being extended here to efficiently support sliding windows. Experimental results on four datasets show that the optimized Giraph implementation achieves a speedup of up to nineteen times over the native implementation, while maintaining good quality of the detected topics. In addition, Giraph Exemplar-based approach achieves the best topic recall and term precision against K-means, Latent Dirichlet Allocation (LDA), Non-negative matrix factorization (NMF) and Latent Semantic Analysis (LSA), while maintaining a good term recall and running time. The approach is also deployed for detecting topics from real-time Twitter streams and its scalability is demonstrated.

Moreover, another clustering technique called Local Variance-based Clustering (LVC) is proposed in this thesis for detecting topics from Twitter streams. Local Variance-based Clustering (LVC) defines the data points densities based on their similarities. The proposed local variance measure is calculated based on the variance of the data points similarity histogram and is shown to well distinguish between core, border, connecting and outliers points. Experimental results show that LVC outperforms spectral clustering and affinity propagation in clustering quality using control charts, Ecoli and images datasets, while maintaining a good running time. In addition, results show that LVC can detect topics from Twitter with higher topic recall by 15% and higher term precision by 3% over DBSCAN.

Acknowledgements

I would like to thank all the people who have helped me in completing this thesis and in my research career in general. Moreover, I would like to specifically thank the following:

- My Waterloo supervisors, Prof. Mohamed Kamel and Prof. Fakhri Karray for their support and guidance during my master degree.
- Prof. Mohamed Abdelhady, Prof. Noha A. Yousri, Prof. Nagwa ElMakki and Prof. Mohamed Ismail for their help during my first research projects.
- Prof. Ali Ghodsi for his informative courses that deeply helped me in developing this thesis research.
- Ahmed Frahat for his advises and help during my research.

Dedication

This is dedicated to my husband, Ahmed Elbagoury and my parents Soad and Mohamed.

Table of Contents

List of Tables

List of Figures

List of Acronyms

BSP Bulk Synchronous Programming

CLARANS Clustering Large Applications based on RANdomized Search

DBSCAN Density-Based Spatial Clustering of Applications with Noise

HDFS Hadoop Distributed File System

LDA Latent Dirichlet Allocation

LSA Latent Semantic Analysis

LSI Latent Semantic Indexing

LVC Local Variance-based Clustering

NMF Non-negative Matrix Factorization

PAM Partitioning Around Medoids

R1D Rank-One Downdate

SNN Shared Nearest Neighbors

SVD Singular Value Decomposition

Chapter 1

Introduction

1.1 Motivation

Recently Twitter has become one of the most popular social networks, where users can express themselves by *tweeting* their thoughts in a post of 140 characters at most. The increasing number of users - that reached more than 288 million users in 2014 - who are producing more than 500 million tweets daily[1], motivates a lot of celebrities and organizations to post their updates on Twitter. This large number of users and organizations who are producing large amount of data motivates researchers from different areas like machine learning, data mining and database to develop scalable techniques that can analyse these data and infer useful information. One of the most important tasks is *real time topic detection*, which is the task of discovering the underlying topics that are discussed by a set of tweets in a real time manner. This can be useful in discovering natural disasters as early as possible, market analysis, political events like elections or revolutions and understanding users' sentiments towards some products. Although many other techniques have been developed for detecting topics in Twitter, they represent each topic by a set of terms which could be uncorrelated and may result in topics that cannot be easily understood. This motivated the development of an Exemplar-based topic detection approach [8], where each topic is represented by one tweet instead of using keywords from different tweets that can be unrelated to each other.

Although it was shown that the Exemplar-based topic detection approach is superior to other approaches (like K-means and non-negative matrix factorization), its scalability needs to be enhanced to handle the large number of daily generated tweets. One factor that affects both the

[1] http://www.statisticbrain.com/twitter-statistics/ [Last visit 18/2/2016]

quality of the detected topics and the response time is the size of the time window. On one hand, small time windows will have small number of tweets and hence a lower response time, however, there will not be enough time for topics to be shaped; on the other hand, using large time windows will be enough for topics to be formed but the number of tweets to be processed will be large and the system will have a large delay. It was shown in [32] that 10 minutes is a suitable time window size for topics to be formed and in the same time to be able to detect topics as soon as they occur. However, Exemplar-based topic detection doesn't scale to work with the number of tweets that arrive at 10 minutes time window. So, the objective of this work is to propose a scalable Exemplar-based approach in order to handle large-size windows efficiently while preserving a high quality of the detected topics. This can be achieved by scaling out the computations to be executed on multiple machines instead of one machine. One way to scale out the computations is using MapReduce. However, MapReduce is not suitable for real time applications as it needs to perform multiple read and write operations from and to Hadoop Distributed File System (HDFS). To alleviate this problem, we propose a distributed version of the Exemplar-based topic detection approach using Apache Giraph- which is an open source graph processing system. Experimental results on four different datasets show that efficient Giraph-based implementation of the algorithm achieves a speedup up to a factor of 19X while preserving good quality of the discovered topics. There are two versions of time windows which are non-overlapping windows and sliding windows. In sliding windows, the window shifts with a step smaller than the window size. As non-overlapping windows cannot detect topics that span multiple windows, we use a sliding windows model and extend Giraph to handle it efficiently by keeping the same Giraph instance running and modifying the graph structure to delete the old tweets and incorporate the new tweets in each time slot.

Additionally, this thesis addresses another important task which is clustering and utilizes clustering to identify important topics in Twitter streams. Clustering is an unsupervised learning process to group similar data points of unknown classes into one group without any prior knowledge of the data. In the last decades, clustering approaches had proved their importance by being used in many diverse fields like Bioinformatics [39], computer vision [28] and text analysis [20]. As the importance of clustering problem increased, many clustering techniques with different objective functions had been proposed in the literature to conquer this problem.

Many of the traditional objective functions used in K-means, average and complete linkage, Partitioning Around Medoids (PAM) [22] and Clustering Large Applications based on RANdomized Search (CLARANS) [35] are restricted to detecting clusters with specific shapes like spherical shapes clusters detected by K-means. To alleviate this problem, several clustering techniques proposed different clustering criteria which is the clusters' densities, as in Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [12] and Shared Nearest Neighbors (SNN) [11] and thus they were able to detect arbitrary shaped clusters. However, a new prob-

lem emerged with density-based clustering techniques which is their inability to detect clusters with different densities. Hence, there is an important need to develop clustering techniques that can detect both arbitrary shape and arbitrary density clusters as real datasets don't have uniform shapes and densities.

To be able to detect arbitrary density clusters, we need to define another clustering criteria that takes into account that clusters of one dataset can vary in their densities. The problem with density-based clustering techniques like DBSCAN is that it defines two thresholds (ϵ and min point) that globally define the density of the data points and therefore blocks the fact that clusters can have different densities. In this work, a local variance metric is proposed to calculate the density of each data point locally instead of depending on global thresholds. The metric achieves that by calculating the density of data points with respect to their similarity histogram, which captures the data points neighborhood characteristics and helps distinguishing points density in a local manner. By using this new local variance metric, we propose a new clustering technique called LVC that is able to detect arbitrary shape and arbitrary density clusters.

Several techniques like [21] were proposed to detect arbitrary shape and density clusters by defining objective functions that minimize the variance between data points within the same cluster. However, these approaches only focused on grouping data points with less variations in their values and didn't consider the data points neighborhood relations and similarities. In this thesis, we develop a new model that greedily group data points that are similar to each others in values along with data points that are similar to each others in their relations and neighborhood similarities. Finally, we apply this model in clustering Twitter streams.

1.2 Summary of Contributions

The contributions of the thesis can be summarized as follows:

- Proposing a scalable version of Exemplar-based topic detection approach [8] using Apache Giraph and extending it to handle sliding windows efficiently.
- Empirically evaluating the proposed distributed Giraph approach and comparing it against the centralized implementation and related work techniques (K-means, Latent Dirichlet Allocation (LDA), Non-negative Matrix Factorization (NMF) and Latent Semantic Analysis (LSA)) using four datasets.
- Deploying the system to detect topics from real-time Twitter streams and showing its scalability.

- Proposing LVC and evaluating it in detecting arbitrary shape and arbitrary density clusters compared to DBSCAN, spectral clustering, affinity propagation and mean shift. In addition, LVC technique is evaluated in different fields like control charts, Ecoli, pen digits and topic detection.

1.3 Thesis Organization

The rest of this chapter describes the notations used throughout the thesis. The first part of Chapter 2 discusses topic detection related work, while the second part gives the neccerarry background used to scale out Exemplar-based topic detection. The proposed scalable Exemplar-based topic detection approach is explained in Chapter 3, along with experimental results that show empirically the efficiently of the proposed technique. Furthermore, Chapter 4 shows the proposed Local Variance-based technique (LVC) and its evaluation in clustering different data types including Twitter data. Finally, Chapter 5 concludes the thesis and discusses a set of possible future extensions.

1.4 Notations

In this section, we define the notations that are used in the thesis. First, small letters like m, n are used to denote scalars, script letters are used for defining sets like $\mathcal{H}$, vectors are expressed using small bold italic letters like $\boldsymbol{f}$, and finally capital letters like X are used for denoting matrices. Additionally, the following notations are used:

$\|\mathcal{H}\|$	size of the set $\mathcal{H}$.
$\boldsymbol{f}_i$	i-th element of $\boldsymbol{f}$.
$X_{i,j}$	(i, j)-th entry of X.
$X_{i,:}$	i-th row of X.
$X_{:,j}$	j-th column of X.
X^T	transpose of X.

1.5 Summary

In this chapter, we have explained the motivations of our work, the contributions of this thesis and the thesis organization and notations. In the next chapter, we will discuss the necessary background and the related work for detecting topics from Twitter streams.

Chapter 2

Background and Related Work

2.1 Topic Detection Related Work

This section explains the various types of topic detection techniques that are used in the literature. These types can be classified into four different categories which are:

- Matrix Factorization like LSA, NMF and Rank-One Downdate (R1D).
- Probabilistic Topic Detection like LDA.
- Clustering Approaches, which include:
  - Traditional Clustering Approaches like K-means and online Kmeans.
  - Distributed Clustering Approaches.
  - Clustering over Dynamic Graphs.
- Sketch-based Topic Detection.

2.1.1 Topic Detection using Matrix Factorization

Several techniques were proposed in the literature for matrix factorization. The problem of matrix factorization is defined in the context of topic detection as factorizing a data matrix X, where rows of X represent documents and columns of X represent terms. Each entry (i, j) of X corresponds to how related document i to term j. Matrix factorization techniques like

Latent Semantic Indexing (LSI) factorizes the matrix X into the multiplication of three matrices $U\Sigma V^T$, which are the Singular Value Decomposition (SVD) of matrix X. This can be interpreted as matrix U representing the relations between the documents and the topics, Σ representing the relations between the topics and V^T representing the relations between the topics and the terms. Therefore, documents can be assigned to topics based on matrix U and topics can be visualized according to their terms relations using matrix V^T. However, LSI has two problems, which are: (1) U and V^T can have negative values and thus interpreting them is difficult and (2) The detected topics are latent and don't correspond to real documents or have clear meaning.

To overcome the aforementioned problem, another category of matrix factorization techniques were proposed like non-negative matrix factorization (NMF), where it factorizes the matrix X into the multiplication of two matrices W and H. Additionally, NMF enforces a restriction on the values of W and H to ensure that the values are non-negative and thus H and W can be easily interpreted. Still, NMF didn't solve the problem of using latent topics.

2.1.2 Probabilistic Topic Modeling

Probabilistic topic models are generative models that consider each document as a weighted mixture of a set of topics, where each of these topics has a distribution over terms [4]. Each document d_i has a distribution over the set of topics, and each word in the document d_i is generated from one of the topics, based on the document over topics and topic over words hidden distributions. LDA proposed in [4] is one of the well known probabilistic topic modeling approaches that infers the hidden distributions by observing the terms in each document. However, it was reported in [33, 31] that LDA has a problem with the data sparsity in short text.

2.1.3 Topic Detection using Clustering

Several proposed solutions handle scalable topic detection in Twitter using clustering where the assumption is that the tweets that fall in one cluster talk about the same topic. [47] belongs to this category, where its objective is to develop a system that performs distributed online clustering in cloud settings. The system starts by routing the new arriving tweets to a free processing unit, then the processing unit clusters the new tweets either by creating new clusters or by assigning them to existing clusters. Finally, an aggregator adjusts the results of the processing units, for example in K-means adjusting the results is done by re-computing the centroids of the clusters. The system measures the deviation in the clustering quality between two consecutive time slots, and if there is much deviation in the quality, the system re-clusters all the tweets. The deviation in the clustering quality is estimated by comparing the tweets from the two time slots using Hungarian

algorithm to solve a bipartite matching problem. If the tweets from the two consecutive time slots are matched with a high score then there is no much deviation in the clustering quality and there is no need to re-cluster the tweets. Other techniques model the topic detection task in Twitter as a clustering problem over a dynamic graph as in [1], [24] and [46]. [1] models the system as a graph where each node represents a keyword. Two keywords are connected if they co-occur together in at least k tweets. The goal of the system is to find dense clusters in this dynamic graph where each cluster represents the keywords of a topic. To reduce the computation, the system works over a subset of the graph called active graph. Active graph contains only the nodes from the original graph that represent the terms appeared in at least γ tweets. To identify dense clusters, the system identifies 0.5 cliques. In 0.5 clique, each node is connected to at least half of the nodes in this clique. Finally, the paper proposes a ranking algorithm to rank the detected clusters based on their density and correlations between the cluster's edges. The objective of [24] is to build a system that detects events in a dynamic streaming data and keeps track of the events' changes. The system constructs a network where each node represents a user's tweet, and two tweets are connected with an edge weighted with their similarity if this similarity value is above a certain threshold. The system uses a time-based Jaccard similarity in order to take the difference in the posting time of the two tweets into account, where each tweet is represented by the nouns in it which are extracted using a part of speech tagger. In addition, the paper keeps a node priority value to eliminate the noisy nodes' effect and categorizes the nodes into three categories: core, border or noise. The approach uses a DBSCAN like clustering technique to cluster the graph where the events are identified as clusters of tweets. Moreover, the paper keeps track of the changes in the network/cluster, which could be: adding or deleting a node, creating or deleting a cluster, a post that joins or leaves a cluster, splitting a cluster or merging clusters. These operators help in tracking the events' changes. However, as it uses DBSCAN like clustering approach, it cannot detect arbitrary density clusters. [46] proposes a clustering technique for streaming graphs that can track cluster's evolution. The system architecture has two main components, which are a window manager and a graph manager. The window manager receives graph updates and generates insertion/deletion operations that are sent to the graph manager. After that, the graph manager updates the graph based on the send operations and also replies to the users' queries about the graph. The graph manager maintains for each time t, two hash tables, one that contains the cluster ID for each vertex ID and another one that contains for each cluster ID, its size and its edges. To capture the clustering evolution, the two tables in two different time slots are compared to find out which clusters have grown, shrunk, created or removed.

Traditional techniques like K-means, K-medoids [18] and PAM [22] restrict the shapes of their detected clusters into spherical shapes. Putting restrictions on the shape of the detected clusters tend to either divide a cluster into smaller spherical shape clusters or tend to produce clusters containing inhomogeneous data points. Therefore, several techniques were proposed to

eliminate the shapes restrictions like DBSCAN [12] and SNN [11]. DBSCAN and SNN both use global thresholds to define the data points density. By doing that, density-based clustering techniques had added another restriction which is limiting their detection to specific clustering densities. Our proposed LVC approach mitigates this problem and is able to detect arbitrary shape and arbitrary density clusters by developing a local measure for the data points density.

Another class of clustering technique is based on mode finding like mean shift [6]. Mean shift [6] represents a general nonparametric mode finding and clustering technique. It doesn't have any assumption on the shape of the clusters nor the number of modes/clusters. The main disadvantages of the mean shift approach is that it needs high data density with clear gradient to locate the cluster centers, which is not true in most of the datasets.

Several clustering techniques had highlighted the importance of taking the similarities between data points into account like spectral clustering [34] and affinity propagation [13]. Spectral clustering first constructs the similarity matrix between the data points and then performs a dimensionality reduction of the similarity matrix using its eigenvalues and eigenvectors. Finally, spectral clustering uses K-means to cluster the new lower dimension representation of the data points. Additionally, affinity propagation is an exemplar-based clustering technique where it tries to identify a small set of data points that best represents the dataset clusters. It achieves that by taking into consideration the points similarities and by exchanging messages between data points such that data points nominate which points they think are the best representative of the clusters. LVC also exploits the data points similarities by measuring the points density in terms of their similarities and relations with their neighborhood region.

Other clustering techniques were proposed that consider the variance as an objective function to be minimized as in [21] and [27]. The work in [21] tries to group data points such that the variance between the data points in one cluster and its centroid is minimized. However, this technique didn't take into account the data points relations and similarities and has measured the variance relative to a virtual data point (centroid) which may mislead the clustering results. Moreover, the work in [27] also tries to minimize the variance of each cluster by first dividing the data points into clusters such that the variance of each cluster is greater than a certain threshold and then by building dendrogram using an agglomerative approach. Finally, the appropriate level of dendrogram is selected so that the variance of each clusters satisfies the selected variance threshold. The main disadvantage of the approach is that it identifies a global variance threshold and thus produces clusters with the same variance level, which is not true in the real datasets.

2.1.4 Sketch-based Topic Detection

Other scalable topic detection techniques for Twitter are based on maintaining a sketch of the streaming data and utilizing this sketch to detect topics in Twitter like [42], [5], [26] and [25]. [42] detects bursty topics in a real time manner. The approach starts by maintaining a sketch of the data, which contains the total number of tweets, the number of occurrence of words and the number of co-occurrence of words pairs. The paper adapts exponential moving average to measure these quantities and uses the change in these quantities to capture the topics' burst. Each tweet is modeled as a mixture of multiple latent topics, where each topic has a fixed distribution over words. Using the maintained values, the paper infers these distributions. Also the paper proposes a dimensionality reduction technique by hashing distinct values into buckets and drawing the topic distribution over buckets instead of words. In addition, [5] adds the geographic constraint, as it wants to detect trending topics related to geographic locations. It starts by describing the characteristics of geographical trending topics (geo-trends) it wants to detect, which are: 1) geo-trend's location has to be a popular location in the current sliding window, 2) probability of topic x occurring in location y needs to be above a threshold θ and 3) probability of location y containing topic x needs to be above a threshold σ. To do this efficiently, the paper constructs a Location-StreamSummary-Table which keeps track of popular locations' counts, where each location cell has $StreamSummary_{y_i}$ list which contains topics that occur in location y_i with a probability above θ. Also, to keep track of the probability mentioned in point (3), the system keeps another table Topic-StreamSummary-Table, which keeps track of each topic's count, and the locations associated with the topic that have probabilities above σ. The paper also drives memory and time requirements, where the memory requirement is shown to be sublinear and the time requirement is shown to have an amortized running time. Moreover, [26] monitors a certain topic by finding the keywords related to it. The used sketch in this paper is the topic's keywords. The paper assumes that topics' classifiers exist, and we need to construct a sampling tweets technique and then run the topic's classifier on the sampled tweets. The sampling process is composed of two steps either using the streaming API, or sampling keywords from tweets and using the filtering API (Twitter API that retrieves tweets containing the provided keywords) with the sampled keywords. After that, the paper extracts the significant keywords of the tweets that belong to this topic and uses these keywords to monitor the topic. The previous steps are done in an iterative manner to detect new keywords that are related to the topic. Finally, [25] also uses keywords as a sketch of the topic. The paper starts by crawling tweets that have crime keywords. It identifies the crime keywords iteratively by first selecting a small set of keywords manually and then automatically observing the coming tweets to find more related terms. After that, the tweets are passed to a crime classifier, which decides if the tweet is about crime or not using its features. If the tweet contains hashtag, url, time or location, then the approach extracts the events related features from it like time and location. The approach also performs user location

prediction from the tweet's location, the tweet's text or the user's profile location. Finally, the paper ranks the tweets using content and user features.

2.2 MapReduce

MapReduce [7] is a programming framework which eases the development of parallel applications that process large volume of data. Usually, these applications run on a cluster of commodity machines, where MapReduce handles data partitioning, task scheduling and fault tolerance. In MapReduce framework, the computations are expressed using two methods, which are called `map` and `reduce`. First, the input data are divided into chunks, then the same `map` function is executed on each chunk in parallel by a mapper task, which outputs key-value pairs. These pairs are shuffled and the pairs with the same key are sent to the same reducer task, which in parallel executes the same `reduce` function on the pairs it has and produces the final output.

2.3 Apache Giraph

Giraph[1] is an open source implementation of Google's Pregel system which employs "Think like a vertex" model, where the computations are defined in terms of what each vertex has to do using a user defined *compute* function. Each vertex is associated with a user defined state, and can be connected to other vertices through directed edges which are associated with a modifiable user defined state too. Computations are executed in a sequence of iterations called supersteps, which are separated by a global synchronization barrier that is known as Bulk Synchronous Programming (BSP). Each vertex can modify the state associated with it or with its edges, receive messages sent to it in the previous superstep $S - 1$, send messages to other vertices which will be received at superstep $S + 1$ and mutate the topology of the graph. Messages are typically sent from one vertex to its neighbors through outgoing edges, however, a message can be sent to any vertex if its ID is known. Each vertex can vote to halt after finishing its work, and it is reactivated after receiving a message. A program terminates if all vertices are inactive and there are no transit messages. The output of a program is the set of values reported by the vertices. To reduce the overhead, user defined combiners can be used to fuse multiple messages into one value. Global computations can be performed using aggregators, where a vertex can provide a value to the aggregator at superstep S and the final result will be available to all vertices at superstep $S + 1$. Local and global topology mutations are supported, where in local mutations

[1] http://giraph.apache.org/

a vertex can add or remove its outgoing edges or remove itself, which introduces no conflicts. On the other hand global mutations let each vertex change the graph topology and may introduce conflicts. Conflicts are resolved by partial ordering, where the removal of edges is performed first then the removal of vertices, adding vertices and finally adding edges. The remaining conflicts are resolved using user defined handlers.

2.4 Other Graph Processing Systems

There are many other graph processing systems like Spark GraphX [16], Cassovary [17] and GraphLab [29]. In this section, we shed the light on these systems and discuss the reasons behind choosing Giraph for scaling the Exemplar-based topic detection.

Spark is an open source framework for processing big data. It provides a unified framework that manages big data with variety of representations (text or graph) and variety of sources (batch and real time). Spark GraphX is a Spark API for graph processing. However, it is still an alpha version and it doesn't support graph mutations, which are needed to support sliding windows efficiently as will be shown. Another graph processing system is Cassovary, which is a graph processing framework for processing large graphs that is implemented in Scala. It is developed by Twitter and contains common graph representations and graph traversal algorithms. However, it doesn't work in a distributed manner. Another remarkable graph processing system is GraphLab, which is a distributed graph processing system that is implemented in C++. GraphLab uses asynchronous model, where there is no global synchronization barrier and vertices can access the latest values of the other vertices and edges. However, the open source GraphLab version doesn't support graph mutations, which is needed for the sliding windows part. Therefore, our choice of Giraph is based on two factors. First, it supports graph mutations which are needed by the Exemplar-based approach as will be shown. Second, it is an open source project that is supported by an active community.

2.5 Exemplar-based Topic Detection

The idea of this approach [8] is to represent each of the detected topics using one tweet which is the most descriptive one for this topic. Using a tweet (exemplar) to represent the topic makes the topics easier to be understandable by the user as the structure of the sentence is preserved and thus the user can make sense of the tweet text instead of providing the user with keywords that are unrelated and not connected in a sentence as produced by other approaches.

The intuition behind the approach is that a tweet which is similar to a set of tweets and dissimilar to the rest of the tweets is a good representative for the topic it discusses. To capture this intuition, the approach constructs the similarity matrix between the tweets and classifies the behavior of the similarity distribution of each tweet into two categories, which are:

- The similarity distribution of tweet i has a low sample variance and thus tweet i is similar to many tweets, or tweet i is not similar to the most of the other tweets.
- The similarity distribution of tweet i has a high sample variance and thus tweet i is similar to a set of tweets and less similar to the others, which will be a good representative for the topic it discusses.

Therefore, the sample variance of the similarity distribution can be a good metric to capture how likely this tweet can be a good representative. The sample variance of the similarity matrix is defined as follows:

$$\text{var}(S_{:i}) = \frac{1}{n-1}\sum_{j=1}^{n}(S_{ij} - \mu_i)^2 \,,$$

where μ_i is the mean of the similarities of tweet t_i: $\mu_i = \frac{1}{n}\sum_{j=1}^{n} S_{ij}$.

To select the tweets using the sample variance metric, the approach starts by sorting the tweets according to their variances and then selects the tweet with the highest variance as the representative of the first topic. After that, the approach removes the tweets that are ϵ similar to the selected tweet to guarantee the diversity in the selected topics and then the approach selects the next remaining tweet with a high variance as the representative of the second topic and so on until retrieving k topics where k is a parameter for the approach.

2.6 Summary

In this chapter, we have discussed the necessary background and the related work for detecting topics from Twitter streams. In the next chapter, we will propose a new method to scale out the Exemplar-based topic detection approach.

Chapter 3

Scalable Exemplar-based Topic Detection Approach

This chapter shows the details of the scalable Exemplar-based topic detection approach. We start by discussing different options and justify our choice of Apache Giraph, then we explain our Giraph implementation in detail and finally, we show the experimental results of comparing the scalable Exemplar-based topic detection against other scalable topic detection techniques using four datasets. In addition, the scalable Exemplar-based topic detection is deployed to detect topics from Twitter streams in real time and its response time is shown while changing the size of the cluster.

3.1 MapReduce Approach

MapReduce is a programming framework which eases the development of parallel applications that process large volume of data. Usually, these applications run on a cluster of commodity machines, where MapReduce handles data partitioning, task scheduling and fault tolerance. The Exemplar-based approach can be modeled as a MapReduce workflow that consists of four jobs. First job_1 reads the tweets text from HDFS as an input and it outputs two files. The first one contains key-value pairs, where each key represents tweet id and the value contains the tf-idf weight of each term in this tweet and the ids of the tweets that share this term with this tweet. The second file contains key-value pairs too, where each key consists of the ids of two tweets that share at least one term, and the value is the similarity between these two tweets based on one shared term only (as both tweets have this term only). Then job_2 reads the first output of

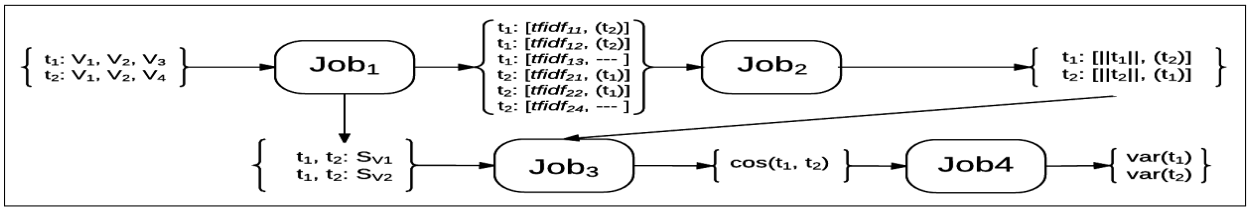


Figure 3.1: MapReduce Workflow

job_1 and outputs the norm of each tweet and the list of tweets that share at least one term with this tweet. After that, job_3 reads the second output of job_1 and the output of job_2 and outputs the cosine similarity between all pairs of tweets that share at least one term. Job_4 reads the output of job_3 as its input and computes the variance of the similarities of each tweet as its output. Finally a java component that runs on one machine, reads the outputs of job_3 and job_4 and selects the exemplar tweets. Figure 3.1 shows the workflow of MapReduce jobs that compute the variance of the similarities for two tweets t_1, that contains terms v_1, v_2 and v_3, and tweet t_2 that contains terms v_1, v_2 and v_4. The figure also shows the inputs and the outputs of each job.

Although this modeling is similar to the modeling in [10]. However, the approach in [10] computes the similarity using term frequency weighting scheme only without the document frequency which was shown to be inefficient for down-weighting the less important terms that occur in many documents [38]. In addition, we don't only provide a model that calculates the similarity but that also calculates the variance of the similarities.

3.1.1 MapReduce Limitations

Although MapReduce approach is simple, it has some limitations. First, there are four jobs in the workflow, where each job reads its input from the HDFS then writes its output to the HDFS so that it can be read by subsequent jobs. Reading and writing from and to HDFS are costly operations, which result in slowing down the computations. Second, MapReduce jobs suffer from the stragglers problem, where the whole job has to wait for one straggler machine to finish that is shown in [32] to slow down the computations. Due to these limitations, it is obvious that MapReduce approach will not be able to detect topics from a large number of tweets in a real time. That is why we decided to use a graph-based system that keeps everything in memory.

3.2 Giraph Approach

This section shows the details of using Giraph system to scale out the Exemplar-based topic detection approach. We first show how to model the problem as a general graph problem and we show how to solve it using Giraph.

The Exemplar-based topic detection approach is built based on calculating the variance of the similarities of each tweet t_i with the rest of the tweets, which is calculated as:

$$\text{var}(S_{:i}) = \frac{1}{n-1}\sum_{j=1}^{n}(S_{ij} - \mu_i)^2 \tag{3.1}$$

Let $\mathcal{N}_i$ be the set of tweets that have non-zero similarities with tweet t_i and $\overline{\mathcal{N}_i}$ be the set of tweets that have zero similarities with tweet t_i, then:

$$\text{var}(S_{:i}) = \frac{1}{n-1}(\sum_{j\in\mathcal{N}_i}(S_{ij} - \mu_i)^2 + \sum_{j\in\overline{\mathcal{N}_i}}(S_{ij} - \mu_i)^2)$$

As $S_{ij} = 0, \forall j \in \overline{N}_i$ and $|\mathcal{N}_i| + |\overline{\mathcal{N}_i}| = n$. Therefore,

$$\text{var}(S_{:i}) = \frac{1}{n-1}(\sum_{j\in\mathcal{N}_i}(S_{ij} - \mu_i)^2 + (n - |\mathcal{N}_i|)\mu_i^2) \tag{3.2}$$

Where, $\mu_i = \frac{1}{n}\sum_{j\in\mathcal{N}_i}(S_{ij})$. Based on equation 3.2, to calculate the variance of similarities of each tweet t_i, it is only needed to calculate the similarities with the tweets in the set $\mathcal{N}_i$. To calculate the set $\mathcal{N}_i$, let $\mathcal{W}_i$ be the set of terms that occur in tweet t_i. As S is the cosine similarity measure, we note that:

$$S_{ij} \neq 0 \leftrightarrow \mathcal{W}_i \cap \mathcal{W}_j \neq \emptyset$$

And by definition: $j \in \mathcal{N}_i \leftrightarrow S_{ij} \neq 0$.

Then: $j \in \mathcal{N}_i \leftrightarrow \mathcal{W}_i \cap \mathcal{W}_j \neq \emptyset$. Based on the previous definition, for each tweet t_i, the set $\mathcal{N}_i$ is the set of tweets that share at least one term with t_i, and can be computed as follows:

$$\mathcal{N}_i = \cup_{v_j\in\mathcal{W}_i}\delta(v_j) \tag{3.3}$$

where $\delta(v_j)$ is the set of all tweets that contain the term v_j.

Based on equations 3.2 and 3.3, computing the variance of the similarities of each tweet can be mapped to a graph that contains two types of vertices. The first type represents tweets and the

second represents terms, and there is a weighted edge between a tweet vertex t_i and a term vertex v_j, $\forall v_j \in \mathcal{W}_i$, where the weight is initially set to the frequency of occurrence of the term v_j in the tweet t_i. Figure 3.2 shows the graph representation of three tweets, t_1 that contains the terms v_1, v_2 and v_3, tweet t_2 that contains the terms v_1 and v_4 and tweet t_3 that contains the term v_4, we will use this example through the rest of the paper to illustrate the steps of the computations. Based on this representation, each term vertex v_j can compute the set $\delta(v_j)$ locally by using its edges, then it can compute its inverse document frequency idf_{v_j} and update the weights on the edges between it and all the tweets in the set $\delta(v_j)$ with the tf-idf values. After that, each term vertex v_j broadcasts the set $\delta(v_j)$ to each tweet vertex that belongs to the set $\delta(v_j)$, so that each tweet vertex can know the rest of the tweets that share the term v_j with it. As each tweet t_i knows $\delta(v_j)$, $\forall v_j \in \mathcal{W}_i$, it can compute the set $\mathcal{N}_i$ and calculate its similarity values with tweets t_j, $\forall j \in \mathcal{N}_i$. Finally, after computing the similarities, each tweet t_i can calculate the variance of these similarities var($S_{:i}$). Next subsections show the details of implementing this model using Giraph and further optimizations. In the rest of the paper, we will refer to the tweet vertex and the term vertex as tweet and term respectively when it is not ambiguous to do so.

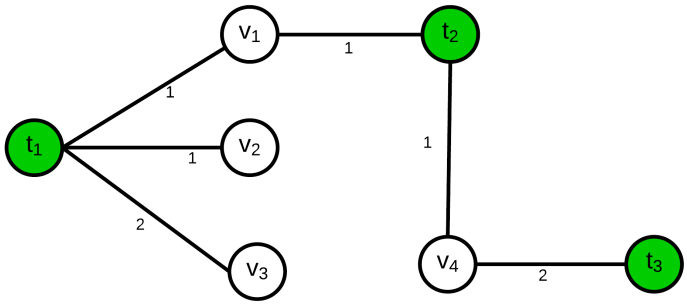


Figure 3.2: Graph Modeling

3.2.1 Giraph Modeling

To implement the aforementioned graph modeling in Giraph, we first need to have two types of vertices while Giraph has only one `Vertex` class. So, to distinguish between tweet vertices and term vertices, positive ids are assigned to tweets, while terms are assigned non-positive ids. The computation of the similarities and the variance of tweets are performed in three supersteps.

- Superstep 0:

- Term vertices compute the set $\delta(v_j)$ which contains all the tweets vertices that are connected to the vertex v_j, then it broadcasts this list to each member in this list (all the tweets vertices it is connected to). This list is customized before being sent to each tweet t_i by removing the vertex t_i from it in order to reduce the size of the message. Figure 3.3 illustrates superstep 0 on the example shown in figure 3.2, where term v_1 computes the list $\delta(v_1) = \{t_1, t_2\}$, then it broadcasts it to each member.

- Superstep 1:

  - Based on messages sent in superstep 0, each tweet vertex t_i calculates inverse document frequency $\text{idf}_{v_j} = \frac{n}{|\delta(v_j)|}$, where n is the number of tweets. Then it updates the edge's weight between it and each term vertex v_j to be the $\text{tf-idf}_{ij} = \text{tf}_{ij} \times \text{idf}_{v_j}$. Each tweet vertex t_i can determine the set of its neighboring tweets $\mathcal{N}_i$, and sends its tf-idf feature vector ($\boldsymbol{x_i}$) to all elements in the set $\mathcal{N}_i$. As shown in figure 3.4, tweet t_1 sends a message to tweet t_2 containing term id and tf-idf weight of each term that appears in t_1.

- Superstep 2:

  - Each tweet node t_i receives the tf-idf vector $\boldsymbol{x_j}$, $\forall j \in \mathcal{N}_i$, then it calculates the cosine similarity S_{ij}, $\forall j \in \mathcal{N}_i$. Finally it calculates the variance of its similarities using equation 3.2 and writes it to HDFS. In addition, for each tweet t_i the ids of all tweets that have similarity values with t_i above certain ϵ are written in the HDFS as conflict tweets to this tweet. Conflict tweets are tweets that are similar to this tweet, which means they cannot both be detected as two different topics.

Finally, a centralized component sorts the tweets based on the variance of their similarities and selects the top tweets with the highest variances that don't conflict with each other as the topics.

3.2.2 Giraph Optimizations

Running this implementation on one time slot containing $20,000$ tweets results in 282 seconds which is a large delay, therefore, one optimization is applied in superstep 1. In order to calculate the cosine similarity between a pair of tweets, each tweet needs to know from the other tweet only its tf-idf weights of the terms shared between them and its norm. Therefore, each pair of tweets doesn't need to share the tf-idf weights of non-shared terms. After applying this optimization, the running time of Giraph approach on $20,000$ tweets is reduced to 163 seconds.

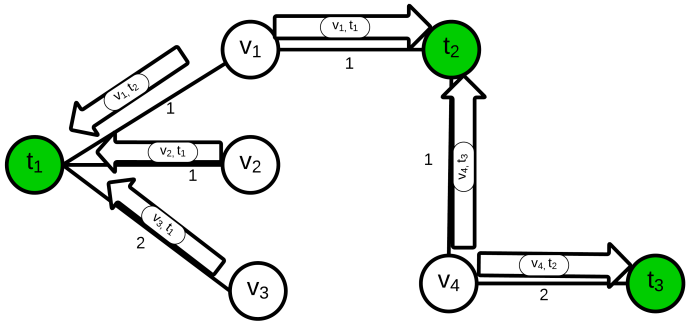


Figure 3.3: Giraph Superstep 0

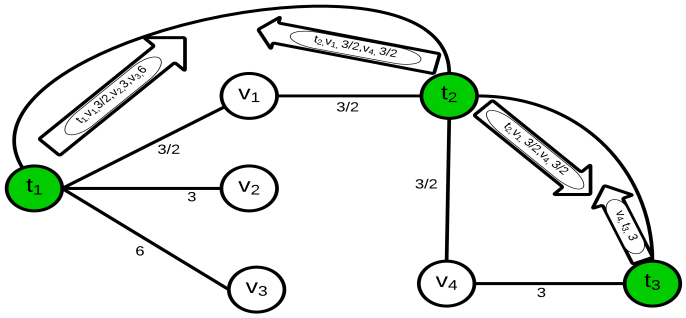


Figure 3.4: Giraph Superstep 1

Another optimization is to let each term vertex v_j send its tf-idf weights with all the tweets in the set $\delta(v_j)$ along with the set itself in superstep 0 instead of sending the set alone. By doing this there is no need to send the tf-idf weights in superstep 1 which saves 4 bytes integer of term id during the communication between pairs of tweets. Applying this optimization, the running time on $20,000$ tweets is reduced from 163 seconds to 84 seconds.

The final optimization is to complete all the computation within Giraph without the need to write an intermediate output to the HDFS, read this output and process it in a centralized component. We use Giraph aggregator for this purpose, where Giraph has a two-level hierarchy of aggregators. The first level is on each worker where the values are partially aggregated on each machine then the partially aggregated value is forwarded to the second level on the master machine to obtain the final aggregated value. The aggregator approach is described as follows:

- When a worker aggregator receives a new tweet's variance message, it scans its sorted list of candidate tweets, if the new tweet conflicts with a tweet with higher variance, then it will

Table 3.1: Running Time, Storage and Communication Bounds of Girpah Algorithm

	Running Time	Communication Cost	Memory Usage
Superstep 0	$O(d_v)$	$O(d_v^2)$	$O(d_v^2)$
Superstep 1	$O(d_{tt})$	$O(d_{tt})$	$O(d_{tt})$
Superstep 2	$O(d_{tt})$	$O(1)$	$O(d_{tt})$

neglect the new tweet and keep the list unchanged. However, if it doesn't, the aggregator will add the new tweet to the list in the right position that maintains the list sorted then it will remove the tweets that have a lower variance and conflict with the new tweet, if there are any. If the size of the list exceeds m, it will only keep the top m tweets.

- When the master aggregator receives worker aggregators' lists, it will combine and sort them. Finally it will choose the top variance tweets that don't conflict with each other as topics.

The worker aggregator runs in $O(\frac{n}{w}m)$ and the master aggregator runs in $O(wm\log(wm))$ where w is the number of workers. Therefore, the total aggregator runs in $O(\frac{n}{w}m+wm\log(wm))$. The parameter m should be chosen carefully, because a small value of m could decrease the topic quality, as each worker aggregator makes a local decision by reporting its m tweets as candidate exemplars. However, its m tweets could conflict with other worker aggregators' tweets and therefore redundant topics will be detected as different topics, which harms the quality of the detected topics. On the other side, large values of m increases the message size that worker aggregators send to the master aggregator. Therefore, the parameter m has a great impact on the results and needs to be tuned carefully.

Giving that d_v is the average degree of each term vertex, d_{tt} is the average degree of each tweet vertex with the other tweets and d_{tv} is the average degree of each tweet vertex with term vertices. Due to the tweet length limitation (140 characters) d_{tv} is a small constant number. Table 3.1 shows the running time, storage and communication bounds on Giraph algorithm per vertex, where the memory usage is calculated as the messages size generated by the vertex plus the memory used by the vertex compute method. As shown in the table, the running time is linear. However, the communication cost and the memory usage are quadratic.

3.2.3 Memory Optimized Giraph Approach

The memory requirement of the previous approach is quadratic in the average term degree $O(d_v^2)$ which does not scale for a large number of tweets. Thus, in this section, another Giraph approach

is proposed to reduce the memory requirement to be linear so that the approach can handle large volume of tweets. The quadratic order of memory requirements appeared due to sharing the set of neighboring tweets by each term vertex among the neighboring tweets. This step can be avoided by reformulating equation 3.1 to be:

$$\text{var}(S_{:i}) = \frac{1}{n-1}\left(\sum_{t_j \in \mathcal{N}_i} S_{ij}^2 - 2\mu_i \sum_{t_j \in \mathcal{N}_i} S_{ij} + n\mu_i^2\right) \tag{3.4}$$

Where $\mu_i = \frac{1}{n}\sum_{v_j \in \mathcal{N}_i} S_{ij}$. As the set $\mathcal{N}_i$ can be expressed as $\cup_{v_k \in \mathcal{W}_i}\delta(v_k)$. Then, the term $\sum_{t_j \in \mathcal{N}_i} S_{ij}$ in equation 3.4 can be computed as the sum of $\sum_{t_j \in \delta(v_k)} S_{ij}, \forall v_k \in \mathcal{W}_i$ and with considering that if two tweets t_i and t_j share more than one term, then only the term vertex with highest id, v_k, calculates the $\sum_{t_j \in \delta(v_k)} S_{ij}$. The terms $\sum_{t_j \in \mathcal{N}_i} S_{ij}^2$ and μ_i can be computed in a similar manner. Thus, to compute equation 3.4, some term vertices can compute parts of it ($\sum_{t_j \in \delta(v_k)} S_{ij}$) and then the tweet vertex adds these parts together without the need to send messages of quadratic order. To achieve this every term vertex v_k needs the feature vector of the tweets $t_i \in \delta(v_k)$ and this information is obtained as follows:

- Each term vertex broadcasts its degree to its neighboring tweets.
- Using the term vertex degree, the tweet vertex calculates each term tf-idf weight, constructs its tf-idf vector and sends it to all its neighboring term vertices.
- Each term vertex v_k calculates the similarity between each pair of tweets tf-idf vectors, and then it sends for each tweet the values $\sum_{j \in \delta(k)} S_{ij}$ and $\sum_{j \in \delta(k)} S_{ij}^2$. Also, term vertex sends to each tweet its C top conflicting tweets (the ones with ϵ close similarity to it). If two tweets share more than one term, then only the term vertex with highest id calculates their pairwise similarity.
- Each tweet vertex adds the sum of similarities and the sum of square of similarities it receives from its term vertices and using these two values, it can compute its variance of similarities using equation 3.4. In addition, each tweet vertex unions the conflict lists it receives from its term verticies. Finally, the tweet sends its variance and conflict list to the worker aggregator.
- Each worker aggregator accumulates the tweets it receives and then sends them to the master aggregator. Finally, the master aggregator performs the same logic explained in the previous section.

Table 3.2: Running Time, Storage and Communication Bounds of Memory Optimized Girpah Algorithm

	Running Time	Communication Cost	Memory Usage
Superstep 0	$O(d_v)$	$O(1)$	$O(d_v)$
Superstep 1	$O(1)$	$O(1)$	$O(1)$
Superstep 2	$O(d_v^2)$	$O(Cd_v)$	$O(Cd_v)$
Superstep 3	$O(1)$	$O(1)$	$O(Cd_{tt})$

Table 3.2 shows the bounds on running time, communicate cost and memory usage of the memory optimized Giraph approach. As shown in the table, the communication cost and the memory usage are linear, however the running time is quadratic. As Giraph is paralleling the running time of the vertices across the distributed machines, the overall running time is reduced to $O(\frac{n}{w}d_v^2)$.

3.2.4 Sliding Windows

Time windows that shift with a step smaller than the window size are called sliding windows. By using sliding windows, topics that span multiple windows and are not frequent in any of them can still be detected. In the aforementioned Giraph model, the sliding windows are handled by running different instance of Giraph for each time slot. However, one drawback of handling sliding windows in such a way is the overhead of initializing multiple Giraph instances and also the tweets that exist in two consecutive time slots are loaded twice. This section describes how to extend the Giraph model to handle the sliding windows without the need to run different instances for each time slot.

The basic idea to handle sliding windows is to keep the Giraph instance running and only load the part of the graph that changes between time slots. Given a time window of size l and a shifting of sliding window of size s. At each time slot, s tweets are removed and s tweets are inserted. Instead of removing vertices from Giraph and adding new ones, the identities of s vertices that are supposed to be removed will be changed so that they can represent the new tweets instead of the old ones. Changing the identity of a tweet vertex means updating its connections with the term vertices such that it becomes connected only to the terms that are contained in the new tweet. This can be done by removing the connections between the tweet vertex and the term vertices and adding connections to the terms that are in the new tweet. Applying this model will save the cost of initializing Giraph multiple times and the cost of deleting and inserting vertices.

In order to maintain load balancing between worker machines, the file containing the new s tweets is divided into w parts, where w is the number of worker machines, and then each worker

arbitrary selects one vertex from the vertices it has to read the worker corresponding part of the input file, then this vertex will update the topology of the graph.

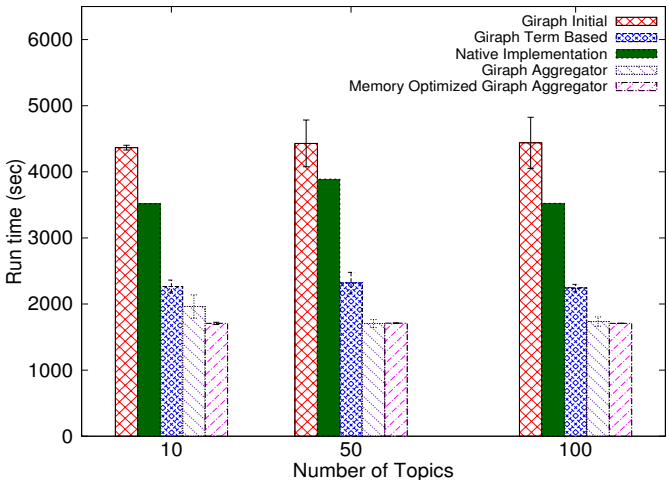


Figure 3.5: US Election Running Time Comparison

3.3 Experimental Results

3.3.1 Experimental Setup

Experiments are conducted using Giraph 1.0.0, Hadoop 1.0.4 and GraphLab v2.2. Four datasets are used which are:

- **US Election Dataset:** which contains $578,837$ tweets. It is divided into 26 time slots, where each time slot has around $20,000$ tweets.
- **Super Tuesday Dataset:** which contains $353,650$ tweets. It is divided into 8 time slots, where some time slots contain around $20,000$ tweets and others contain around $60,000$ tweets. However, there is one time slot that contains around $80,000$ tweets, we have removed this time slot from our evaluation as the native implementation's memory footprint for it is 25GB.
- **120M Tweets Dataset:** which contains around 120 million tweets covering more than 500 events and collected by [30]. As we were able to retrieve only portion of the tweets, we were unable to use the ground truth and therefore only the running time is reported.

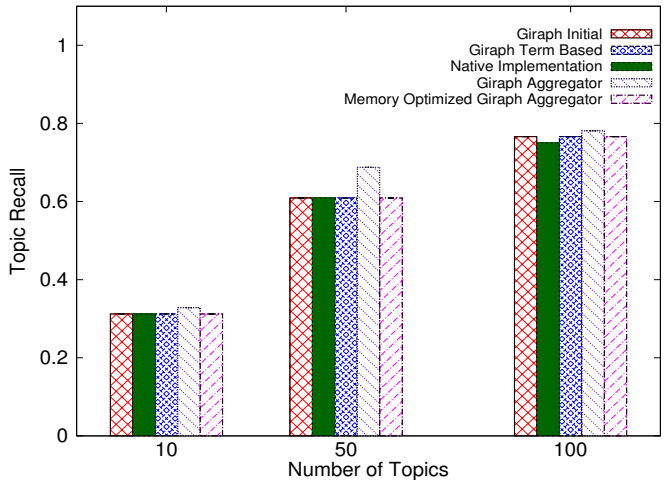


Figure 3.6: US Election Topic Recall Comparison

- **2M Streaming API Tweets:** which contains 2 million English tweets collected using the streaming API.

Four types of experiments are conducted, which are:

- **Validating Topic Quality Experiments:** The purpose of this experiment is measuring the quality of the topics detected by the centralized native implementation and Giraph distributed implementations. This experiment is performed on a cluster of three machines, each contains 16GB RAM and i7 3.6GHz CPU using US Election and Super Tuesday datasets. Sliding window is not used in this experiments as these datasets have non-overlapping time slots. In addition, the effect of changing m of the aggregator is measured on the quality of the topics. The quality of the detected topics is measured using:
  - **Topic recall:** Percentage of topics successfully retrieved.
  - **Term precision:** Number of correct keywords in the detected topics over the total number of keywords.
  - **Term recall:** Number of correct keywords over the total number of keywords in the ground truth topics.
  - **Running Time:** Total time needed to detect the topics.

  Topic precision is not used as not all the topics covered by Twitter appear in news sources and the ground truth for these datasets are constructed from news sources. Paper [2] that

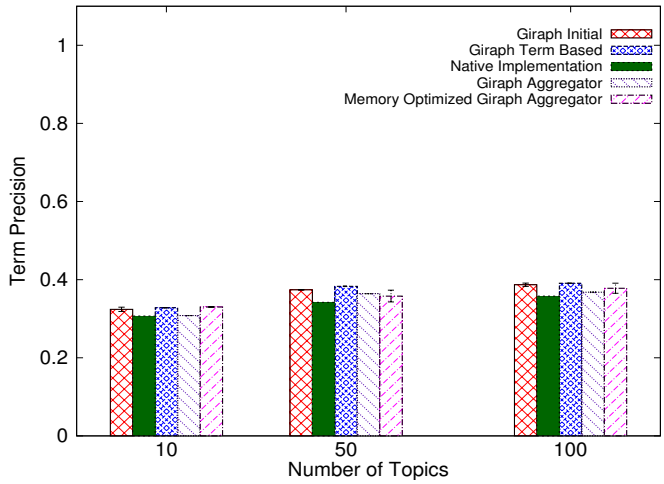


Figure 3.7: US Election Term Precision Comparison

constructed these datasets and their ground truth have also neglected the Topic precision for the same reason.

- **Related Work Comparison Experiments:** The purpose of this experiment is to compare the running time and the topic quality of proposed approach to other topic detection approaches like SVD, LDA, LSA and K-means. This experiment is performed on the same settings as in validating topic quality experiment.
- **Sliding Window Experiment:** The purpose of this experiment is to compare the running time of sliding window Giraph implementation to the Giraph implementation, while increasing the number of tweets. This experiment is performed on the same settings as in validating topic quality experiment.
- **Scalability Experiment:** To measure the scalability of the proposed approach, it is evaluated on clusters of 8, 16 and 32 machines. Each machine contains 30.5GB RAM and 4 cores.

3.3.2 US Election and Super Tuesday Results

Four variations of the distributed implementation are compared to the centralized native Java implementation, which are:

- **Giraph Initial:** Giraph implementation with no optimization.

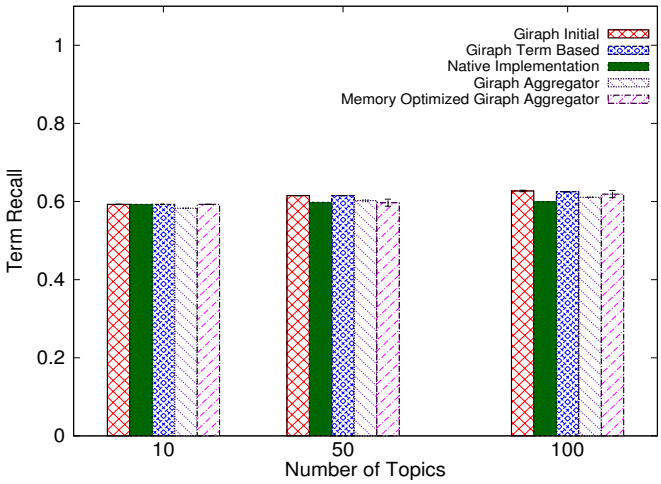


Figure 3.8: US Election Term Recall Comparison

- **Giraph Term-based:** Giraph implementation with the optimizations that let each term vertex v_j send its tf-idf weight in superstep 0 along with the set $\delta(v_j)$.
- **Giraph Aggregator:** Giraph Term-based implementation which uses an aggregator to compute the topics instead of the centralized component, where $m = 5000$.
- **Memory Optimized Giraph Aggregator:** Memory optimized Giraph Algorithm, C is chosen to equal 1000 empirically as it achieves the best topic quality.

All runs are repeated three times and 95% confidence interval is reported in the figures. Figures 3.5, 3.6, 3.7 and 3.8 show the results of running time, topic recall, term precision and term recall respectively on US election dataset. Results show that our proposed Giraph approach is able to speedup the native implementation by a factor of $\sim$ 2X, while maintaining or increasing the topics quality. Also, results show that changing the number of detected topics has a small effect on the running time as the dominating time in the Exemplar-based approach is the time to calculate the similarity matrix between the tweets. Moreover, Figures 3.9, 3.10, 3.11 and 3.12 show the results of running time, topic recall, term precision and term recall respectively on Super Tuesday dataset. Results also show that Giraph approach is able to speedup the native implementation by a factor of $\sim$ 3X with good topic quality.

Table 3.3 shows the average CPU idle time, memory usage and the number of the transmitted bytes for the Native implementation and different variations of Giraph implementation. As shown in the table, memory optimized Giraph Aggregator has the lowest memory footprint in both US election and Super Tuesday datasets. It also achieves the lowest CPU utilization in

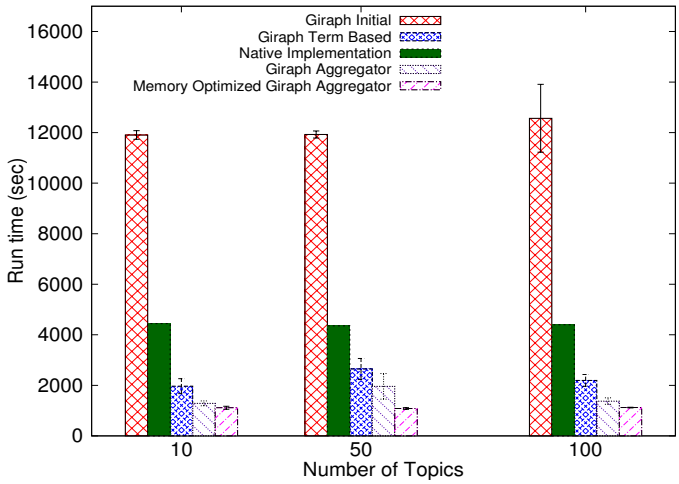


Figure 3.9: Super Tuesday Running Time Comparison

both US election and Super Tuesday. However, the number of transmitted is the largest due to the aggregators usage and that the worker aggregators send all tweets to the master aggregator. For both Giraph Initial and Giraph Term-based, the variance of each tweet is stored by Giraph in the HDFS and then a centralized Java component collects these variances from the HDFS to select the representative exemplars. Thus, the number of transmitted bytes in this case is the sum of the bytes transmitted within Giraph and the bytes sent between all the machines and the machine that runs the centralized component. In the case of US Election dataset the number of bytes transmitted in the Giraph Term-based approach is dominated by the bytes transmitted by Giraph, that is why using the aggregator in the Giraph Aggregator did not reduce the number of transmitted bytes. On the other hand; for the Super Tuesday dataset, the bytes sent between machines by the centralized Java component are the dominating, as a result the cost introduced due to using the aggregator in the Giraph Aggregator approach is out-weighted by eliminating the significant part of the network overhead (moving the data between machines to the centralized Java component). This appeared in Super Tuesday dataset only as Super Tuesday time slots have more tweets than US election, therefore HDFS cost increases while worker aggregators sent data is fixed ($m = 5000$).

For measuring the effect of changing the parameter m on the aggregators, we varied the parameter m from 100 to 5000 and measured the change in the running time and the quality of the detected topics on the US election dataset. As shown in figure 3.14, increasing m increases the running time of Giraph as the aggregator runs in $O(\frac{n}{w}m + wmlog(wm))$. In addition, figure 3.13 shows that increasing m increases the topic recall and as the detected topics increased, the topic's quality (term precision and term recall) decreased.

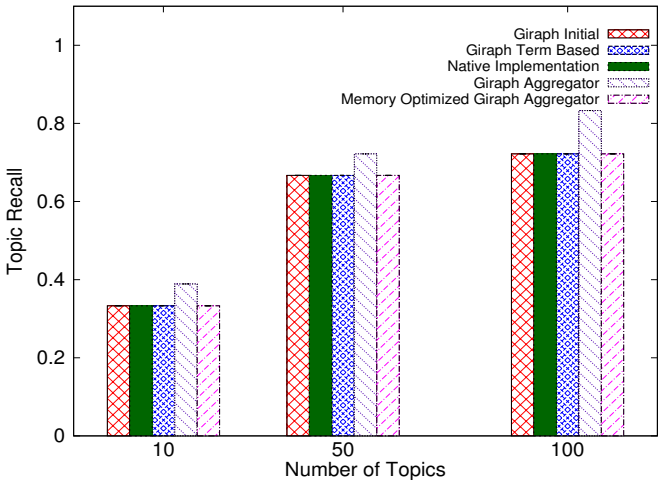


Figure 3.10: Super Tuesday Topic Recall Comparison

3.3.3 Related Work Comparison

We compared our approach to the Graphlab implementations as Giraph has no existing implementation of the related work. It is known in the literature that Graphlab implementations are faster than Giraph, therefore showing that our Giraph approach is faster than Graphlab related work implementations, means that if our approach was implemented in Graphlab it will be even faster. Our approach is compared against LSA, NMF using alternating least squares, Kmeans and LDA. As Graphlab implementation of LDA takes the running time as a parameter, we chose to set the running time of LDA to be equal to the maximum time needed by our approach to finish one time slot which is 75 and 275 seconds for US election and Super Tuesday datasets respectively. Each run is repeated three times and 95% confidence interval is reported. For US election dataset we achieved the second lower running time after Kmeans, while algorithms like NMF and LSA don't scale with increasing the number of topics as shown in figure 3.15. Note that LSA and NMF took more than 6500 seconds. However, we didn't show the complete bar in figure 3.15, so that the figure can be clear. Our approach obtains the highest topic recall except for 10 topics, where LDA was higher with around 6% while our approach was higher than LDA with around 15% and 27% for 50 and 100 topics respectively as shown in figure 3.16. For term precision, our approach achieves the highest value for all number of topics as in figure 3.17, while maintaining good term recall as shown in figure 3.18.

The running time on the Super Tuesday dataset follows the same pattern, where Kmeans achieves the best running time, while LSA does not scale as shown in figure 3.19. Our algorithm achieves the best topic recall and term precision while maintaining a good term recall as shown

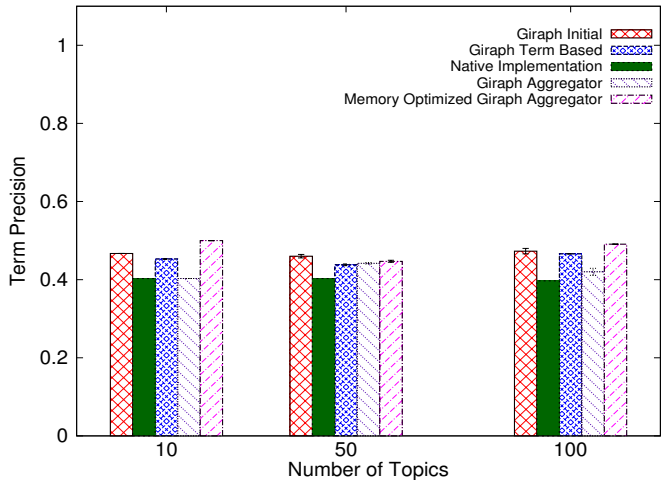


Figure 3.11: Super Tuesday Term Precision Comparison

in figures 3.20, 3.21 and 3.22. It worth noting that LSA achieves the highest term recall as it detects a few number of correct topics which is reflected in its low topic recall.

3.3.4 Giraph Sliding Window

In this section, Giraph sliding windows technique is compared against the Giraph Aggregator. Both systems run using three workers on 10 time slots, which are a subset of 120M tweets dataset. Table 3.4 shows the running time of both systems by varying the window size l and fixing the shifting size s to $10,000$, the number of topics to be detected to 10 and the number of tweets to keep at each aggregator m to 5000. Results show that Giraph Sliding Windows technique is faster than Giraph Aggregator and achieves a speedup of factor 19X over the native implementation. However this gap gets reduced as the window size increases, as the computation time dominates the total running time instead of Giraph setup and vertices' loading time.

3.3.5 Scalability Results

To demonstrate the efficiency of our proposed system in a real-world scenario, we have collected 2 million English tweets using the streaming API, which is the amount of collected English tweets in 10 minutes according to Twitter statistics. In this experiment, we have varied the number of machines from 8 to 16 and 32 machines and measured the running time. Each run is replicated three times and the 95% confidence interval is reported. As shown in figure 3.23 using

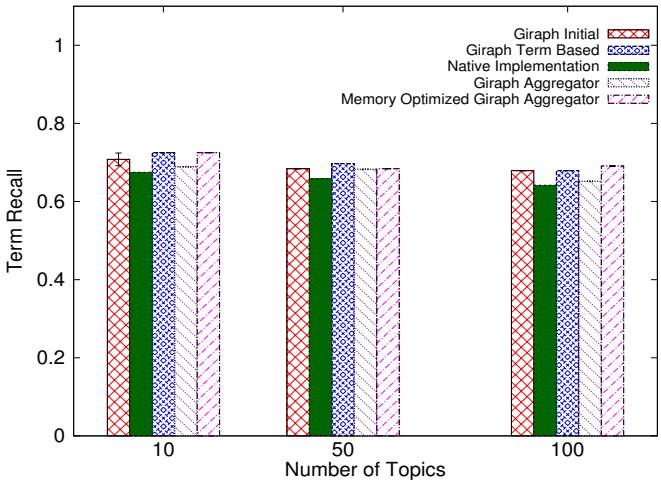


Figure 3.12: Super Tuesday Term Recall Comparison

only 16 machines, the topic detection task is finished in around 7 minutes, which is less than the 10 minute window and therefore using a small cluster of 16 machines is enough for our approach to run in a real settings and to detect topics from real Twitter streams.

3.4 Summary

In this chapter, we have proposed the scalable Exemplar-based topic detection approach and we have shown its efficiency compared to the centralized implementation and compared to other related work like K-means, LDA, NMF and LSA. In the next chapter, we will propose another method for detecting topics from Twitter which is the Local Variance-based Clustering.

Dataset	Approach	Avg CPU Idle Time (%)	Avg Memory Size (GB)	Number of Transmitted Bytes
US Election	**Native Implementation**	86.91%	9.35	—
	Giraph Initial	86.01%	10.07	3.69×10^{12}
	Giraph Term-based	84.75%	7.77	2.13×10^{12}
	Giraph Aggregator	84.56%	7.62	1.18×10^{13}
	Memory Optimized Giraph Aggregator	93.12%	5.30	2.19×10^{13}
Super Tuesday	**Native Implementation**	86.71%	7.74	—
	Giraph Initial	86.54%	9.84	1.23×10^{13}
	Giraph Term-based	83.46%	9.61	1.25×10^{13}
	Giraph Aggregator	81.46%	9.03	5.13×10^{12}
	Memory Optimized Giraph Aggregator	92.57%	4.63	2.19×10^{13}

Table 3.3: CPU, Memory and Network Analysis of Native Implementation and Giraph Distributed Implementations (Number of topics = 100)

Table 3.4: Running Time Comparison (in Seconds) for Giraph Sliding Windows using 120M Tweets Dataset

Approaches \ n	20000	50000	100000
Native Implementation	75.0	518.0	1975.0
Giraph Aggregator	33.1	43.5	107.7
Giraph Sliding Window	24.0	31.9	102.6

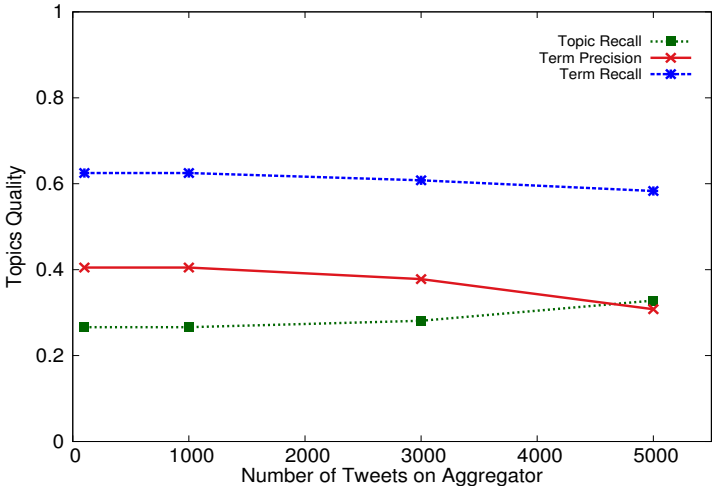


Figure 3.13: Aggregator Parameter m Effect on Topic Detection Quality (Number of detected topics = 10)

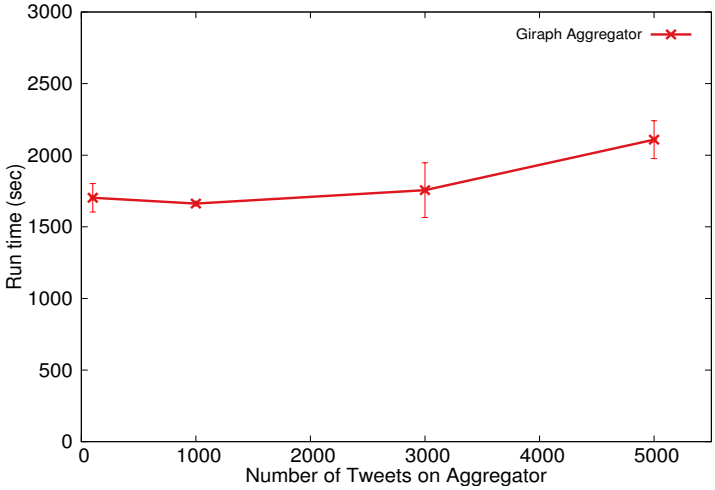


Figure 3.14: Aggregator Parameter m Effect on Running Time (Number of detected topics = 10)

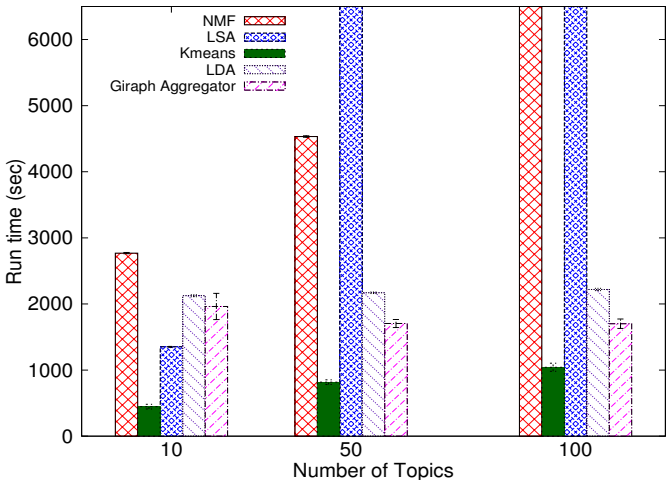


Figure 3.15: US Election Running Time Related Work Comparison

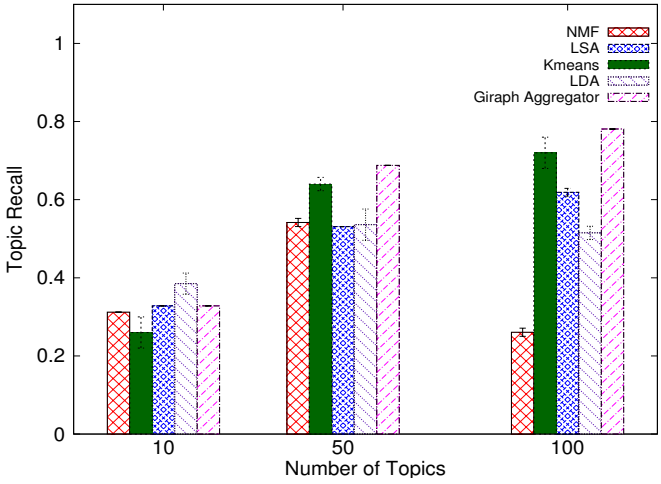


Figure 3.16: US Election Topic Recall Related Work Comparison

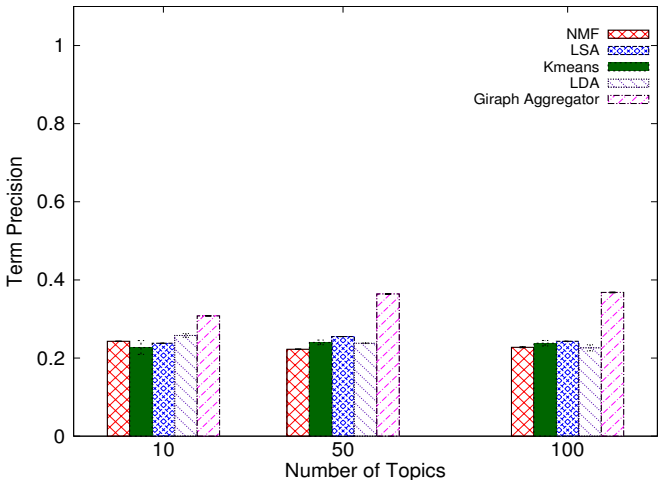


Figure 3.17: US Election Term Precision Related Work Comparison

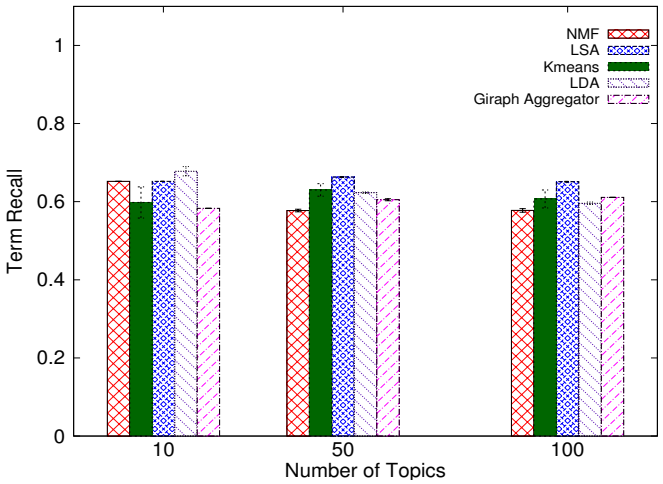


Figure 3.18: US Election Term Recall Related Work Comparison

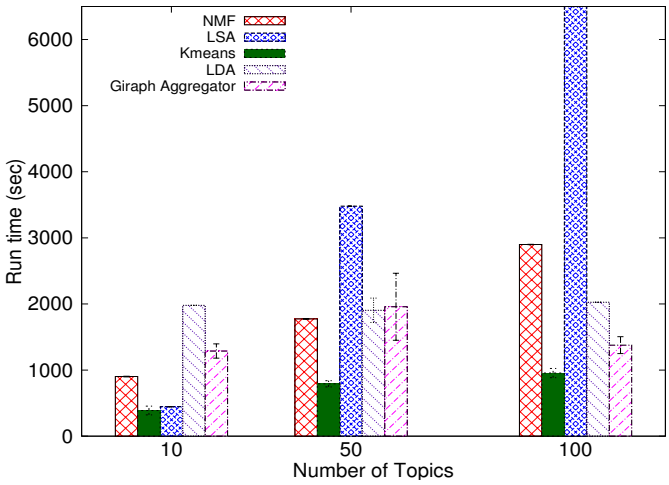


Figure 3.19: Super Tuesday Running Time Related Work Comparison

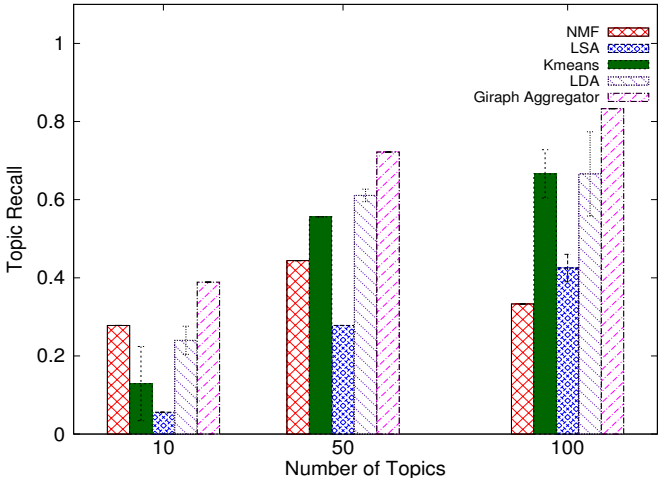


Figure 3.20: Super Tuesday Topic Recall Related Work Comparison

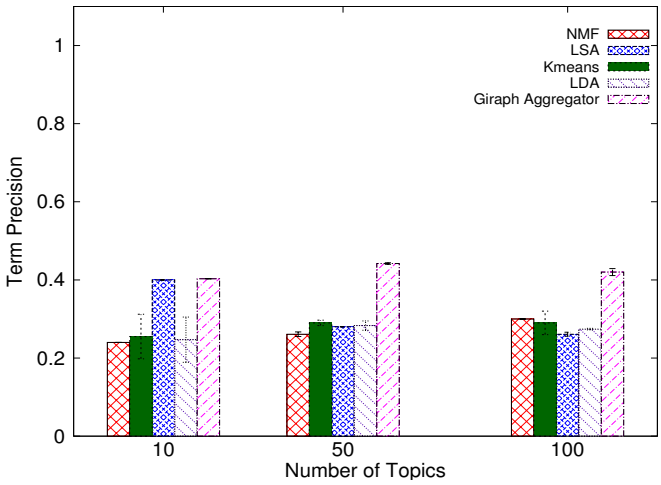


Figure 3.21: Super Tuesday Term Precision Related Work Comparison

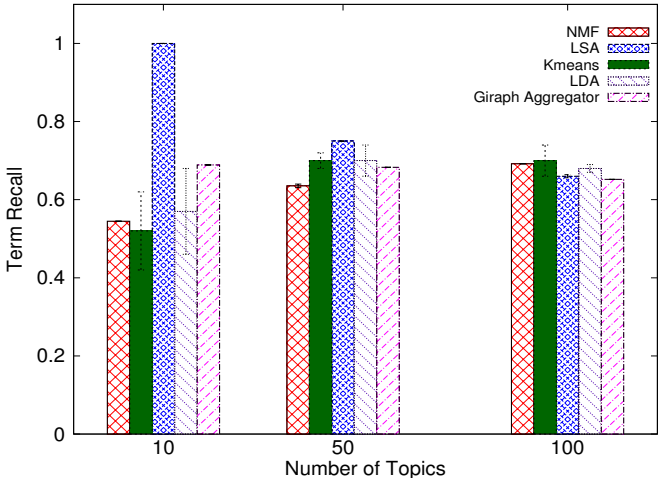


Figure 3.22: Super Tuesday Term Recall Related Work Comparison

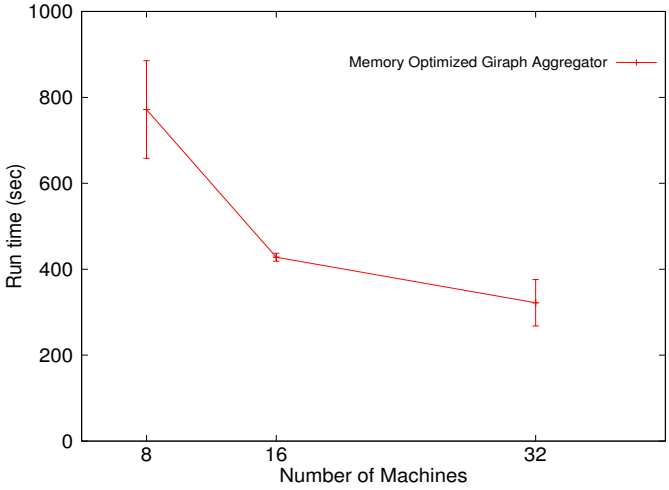


Figure 3.23: The Effect of Increasing the Number of Machines on the Running Time

Chapter 4

Local Variance-based Clustering

In this chapter, we address the problem of topic detection from a different perspective by proposing Local Variance-based Clustering. Local Variance-based Clustering first defines the data points densities based on their similarities. The proposed local variance measure is calculated based on the variance of the data points similarity histogram and is shown to well distinguish between core, border, connecting and outliers points. We demonstrate the effectiveness of the clustering approach in different domains like control charts, protein and images datasets as well as in the topic detection task.

4.1 Problem Definition

Given $X \in \mathbb{R}^{n \times d}$ n data points in d dimensional space, our objective is to group them into arbitrary shape and arbitrary density clusters $\mathcal{C}$, such that:

$$\min_{\mathcal{C}} \sum_{\mathcal{C}_i \in \mathcal{C}} \text{variance}(\mathcal{C}_i)$$

Where $\text{variance}(\mathcal{C}_i) = \frac{1}{|\mathcal{C}_i|-1} \sum_{x_j \in \mathcal{C}_i} (x_j - \mu_{\mathcal{C}_i})^2$ and $\mu_{\mathcal{C}_i} = \frac{1}{|\mathcal{C}_i|} \sum_{x_j \in \mathcal{C}_i} x_j$. The previous objective function is NP-hard and in the next subsection we propose a greedily solution for it.

4.2 Local Variance Measure

While other approaches like DBSCAN use density to distinguish outlier points from the rest of the points. Using the DBSCAN density measure reflects only the number of points within

the neighborhood regardless of their distribution within this neighborhood, therefore DBSCAN is unable to differentiate between points within sparse clusters, points within dense clusters and points connecting two clusters. Figure 4.1 shows a blue point that connects two separated clusters colored by red and green. In DBSCAN, as the density measure only cares about the number of points in the neighborhood, it will consider the blue point as a core point and hence mistakenly merge the two clusters. One way to alleviate this problem is by exploiting the distribution of similarities between the data points within a certain neighborhood region. For example, by noting that the blue neighborhood circle has green points and red points near its end and not within its center, we can understand that the blue point is a connecting point.

Figure 4.2 shows four different types of points taken from one of the Chameleon 2D dataset (10,000 data points), where red region represents the local neighborhood of a core point in a dense cluster, yellow region represents the local neighborhood of a core point in a sparse cluster, green region represents the local neighborhood of a connecting point that connects two different clusters and finally the pink region represents the local region of an outlier. Our objective is to develop a metric that can distinguish between these different types of points. The shown neighborhood region is constructed using $\epsilon = 0.9$. Figure 4.3 shows the local similarity histogram of these points. As shown in the figure, the neighborhood region of the dense cluster point contains all ranges of similarities and thus its similarity histogram is almost uniform. However, the neighborhood region of the core point within a sparse cluster has missing similarity ranges and therefore its similarity histogram tends to be more centered in smaller similarity buckets. Finally, an outlier point has a lot of missing similarity ranges and its histogram tends to be much more centered in smaller similarity buckets.

As we can see the similarity histogram of the data points encodes essential information about the neighborhood points distribution and helps in discriminating between different points types like core points, connecting points and outliers. This leads to the local variance measure, which is a measure associated with each data point x and is calculated as the variance of the similarity histogram of x. The similarity histogram contains only the local similarities; the similarities between point x and its neighboring points (points with similarities $\leq \epsilon$ to x). The local variance of each data point i is given by the following equation:

$$V(i) = \frac{1}{|NB(i)| - 1} \sum_{j \in NB(i)} (Sim(i,j) - \mu_i)^2$$

where $\mu_i = \frac{1}{|NB(i)|} \sum_{j \in NB(i)} Sim(i,j)$, $NB(i)$ is the set of ϵ neighbors (points with similarities $\leq \epsilon$ to i) of point i and $Sim(i,j)$ is the similarity value between point i and point j. The variance of the dense cluster core point is 0.5150, the variance of the sparse cluster core point is 0.3960, the variance of the connecting point is 0.3251 and the variance of the outlier point

is 0.0885. Thus, the variance values confirm our intuition that using variance can accurately differentiate between the different types of points.

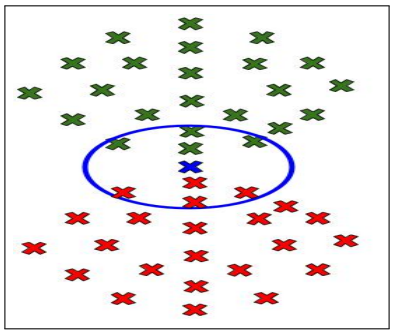


Figure 4.1: Syntactic Data for Illustrating the Difference between DBSCAN and LVC

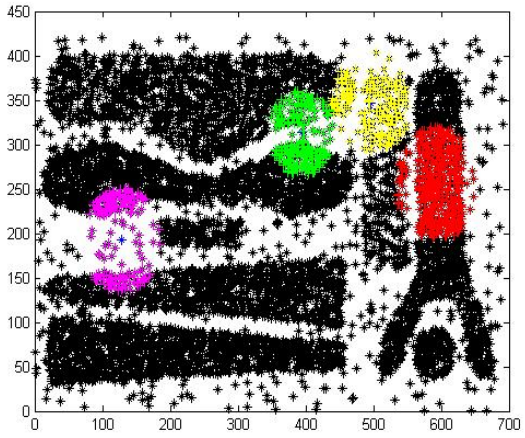
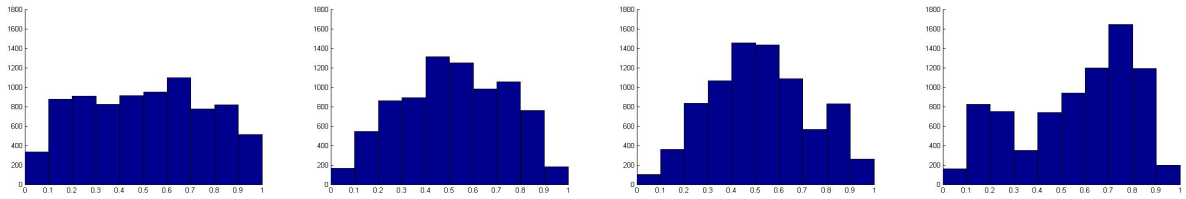


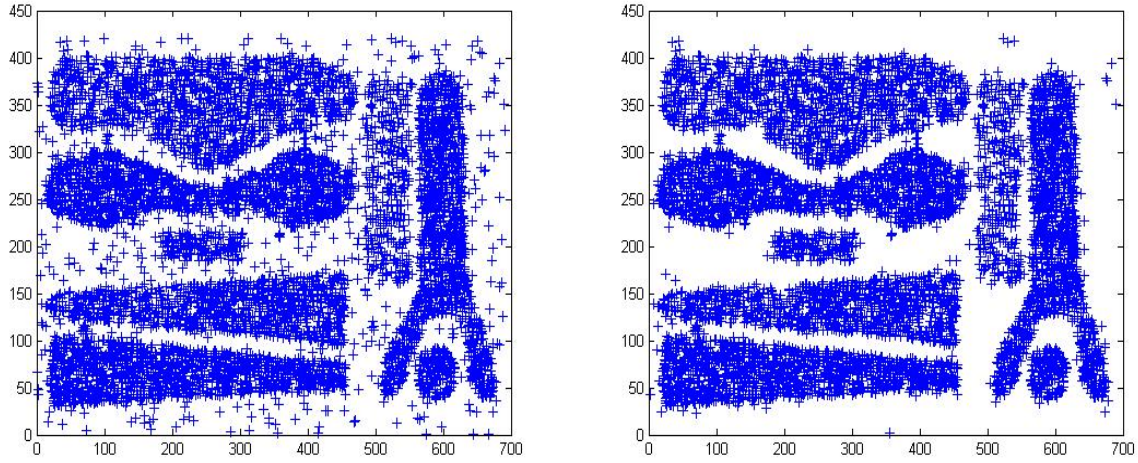
Figure 4.2: Chameleon 2D Dataset Different Points Analysis

To further confirm our analysis that the local variance measure alone can accurately discriminate the outliers, we have used the previous 2D dataset and only kept the points with local variance greater than or equal 0.3. Figure 4.4 shows the original 2D dataset and the 2D dataset after removing the outliers by just thresholding over the local variance measure. As shown in the figure, local variance measure can well distinguish outlier points from other points.



(a) Dense Cluster Point (b) Sparse Cluster Point (c) Connecting Point (d) Outlier Point

Figure 4.3: Similarity Histogram of Different Data Points



(a) Original Chameleon 2D Dataset (b) Dataset after Removing Outliers

Figure 4.4: Local Variance Measure Effect on Removing Outliers

4.3 Variance Model of Flow

By using local variance measure, we can identify core points, connecting points and outliers. Still an important question is how to group these points together? To answer this question, we employ a DBSCAN like algorithm where some points share their cluster labels with some of their neighbors. In order to illustrate how to choose the points that can share their cluster labels and with which points these labels can be shared, we have divided the range of local variance into buckets and assign a unique color to each bucket. Then, the color of each bucket is used to draw the points within this bucket as shown in figure 4.5. Note that outliers using local variance thresholding are removed from the figure. As we can conclude from the figure, interior cluster points have higher local variance than border points in most of the cases. Therefore, interior clustering points can share their labels with their neighbors while border points should have limited sharing privileges to avoid merging two independent clusters. This motivates the usage of a flow model, where interior points propagate their labels to their neighborhood and border points only allowed to share their labels with other border points. This model is called the variance model of flow.

Although discriminating the interior points from the border point is a challenging task, we use a simple local variance thresholding mechanism to achieve that, where points with local variance higher than or equal to t is marked as interior points and other points are labeled as border ones after neglecting outlier points.

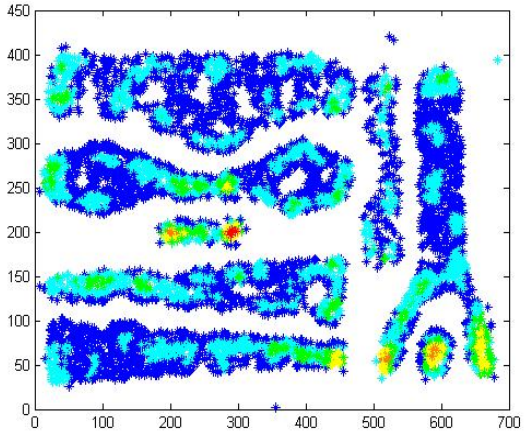


Figure 4.5: Local Variance Distribution

4.4 Local Variance-based Clustering

The previous subsections suggest the usage of the local variance measure with the clustering technique to assist in distinguishing between different types of points. Therefore, LVC technique starts by getting the ϵ neighbors of each data point and then based on these local neighbors, LVC calculates the local variance measure for each data point. Based on the local variance measure, LVC neglects the outlier points, which are defined as points with local variance ≤ 0.1. After that, LVC classifies the remaining points into two types, which are:

- **Core Points:** points with local variance $\geq t$.
- **Border Points:** points with local variance $< t$.

After classifying the points, LVC sorts the points according to their local variance score from the points with high local variance to the points with low local variance, so that it starts expanding the clusters from the most core points to the border ones. LVC utilizes the DBSCAN expand concept. It tries to expand the cluster of each data point i by retrieving its ϵ_{expand} neighborhood points that have the same or lower variance label to i as stated in the variance flow model explained in the previous section. Then, data point i propagates its clustering label to its ϵ_{expand} neighborhood points and then LVC tries to further expand the cluster from the neighborhood points themselves, until no new points are added to the cluster. After that, LVC selects a new unlabeled data point, assigns a new clustering label to it and starts expanding the new cluster from this point. LVC repeats the previous steps until all data points are labeled.

The difference between DBSCAN and LVC is that LVC uses a variance model of flow where points in LVC are examined according to their variance values and interior points propagate their labels to other interior points or border points and border points can only propagate their labels to other border points. To differentiate core points from border points, a threshold t on the value of the local variance is used. Furthermore, LVC uses the local variance measure to determine the points densities, while DBSCAN just identifies the points densities by the number of points in their neighborhood. Algorithm 1 shows the Local Variance-based Clustering pseudo-code. Note that `getPointNeighbors` returns ϵ_{expand} neighbors of point i with $Vlabels$ less than or equal to point i.

Using LVC technique allows us to greedily minimize the within cluster variance objective function, as each cluster is expanded by adding points similar to the points already within the cluster and hence minimizing the within cluster variance. Additionally, each cluster expands until it reaches border points and thus no other core points clusters are merged together in one

cluster which also limits the deviation in the clusters variance.

Algorithm 1: Local Variance-based Clustering Pseudocode

Input: Data Matrix $X \in \mathbb{R}^{n \times d}$, ϵ, ϵ_{expand} and t
Output: $Plabels \in \mathbb{R}^n$, which represents clustering labels of each data point
$NB \leftarrow$ getNeighbors(X, ϵ)
$V \leftarrow$ getVariance(NB)
$V \leftarrow$ removeOutliers(V) //mark outliers as -1
$Vlabels \leftarrow V$
for $i = 1$: Size(V) **do**
  if $V(i) \geq t$ **then**
    $Vlables \leftarrow 2$
  end if
  if $V(i) \neq -1$ **then**
    $Vlables \leftarrow 1$
  end if
end for
$Points \leftarrow$ sort(X, V) // sort X descending based on V
$Plabels \leftarrow$ zeros(n, 1)
$Visited \leftarrow$ zeros(n, 1)
for $i = 1 : n$ **do**
  if $Visited(i)$ or $V(i) == -1$ **then**
    continue
  end if
  $PNB \leftarrow$ getPointNeighbors(X, i, ϵ_{expand}, $Vlabels(i)$)
  $Cnew \leftarrow$ initNewCluster()
  $Plabels(i) \leftarrow Cnew$
  $Visited(i) \leftarrow 1$
  for $j = 1$: Size(PNB) **do**
    if $Visited(PNB(j))$ or $V(PNB(j)) == -1$ **then**
      continue
    end if
    $Plabels(PNB(j)) \leftarrow Cnew$
    $Plabels(PNB(j)) \leftarrow 1$
    $PNB \leftarrow PNB \cup$ getPointNeighbors(X, $PNB(j)$, ϵ_{expand}, $Vlabels(PNB(j))$)
  end for
end for
$Plabels \leftarrow$ removeSmallClusters() //remove clusters with less than 1% of the data points as they represent outlier clusters;

4.4.1 Time and Memory Analysis

In this subsection, we discuss the running time and the memory complexities of the Local Variance-based Clustering technique. For the running time, we can get the neighbors of all the data points in O($dnlogn$) using an indexing data structure that can retrieve the neighbors of each data point in O($logn$). Thus, the time complexity of LVC approach is O($dnlogn$). In addition, the memory complexity of LVC is O(kn), where k is the average number of ϵ neighbors of the data points. Table 4.1 shows time and memory complexities of LVC compared to other clustering techniques, where c is the number of K-means centroids, i is the number of iterations and t is the number of neighbors used in spectral clustering (we use an optimal version of spectral clustering that just stores the similarity between each data point and its t closest neighbors).

4.5 Experimental Results

This section discusses the experimental results of LVC [1] in different domains. In the first subsection, LVC is compared against DBSCAN, spectral clustering and mean shift using Chameleon 2D datasets and in the second subsection, we compare LVC technique to several other techniques, which are K-means, DBSCAN, spectral clustering and affinity propagation using control charts, ecoli and images datasets. In the last subsection, we compare LVC technique to K-means, incremental batch K-means and DBSCAN in the task of topic detection from Twitter streams [2]. Note that, spectral clustering and affinity propagation are not used in the topic detection task as they don't scale with the number of the data points as will be shown in the second subsection and the topic detection task needs scalable techniques.

4.5.1 Chameleon 2D Datasets Clustering

Four 2D Chameleon datasets [3] are used to compare LVC technique with DBSCAN, spectral clustering and mean shift. To evaluate the clustering quality, we draw the 2D datasets and we discuss the images segmentation in this subsection. In addition, euclidean distance is used to measure

[1] You can download LVC Matlab version from https://sourceforge.net/projects/local-variance-clustering/files/LVC/

[2] K-means Matlab implementation was used, spectral clustering implementation was used from http://alumni.cs.ucsb.edu/~wychen/sc.html and affinity propagation implementation was used from http://www.psi.toronto.edu/index.php?q=affinity%20propagation

[3] You can download Chameleon datasets from http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download

the distance between the points. To convert the euclidean distances into similarity values, we normalize the euclidean distances and then let the similarities equal to 1 - euclidean distances.

Clustering parameters were adjusted based on the best data visualization quality for LVC and DBSCAN. For spectral clustering, k was set to the number of clusters. For the first image row (10,000 data points) in figure 4.6, only LVC (using $\epsilon = 0.9, t = 0.37$ and $\epsilon_{expand} = 0.981$) were able to detect the correct clusters, DBSCAN (using $\epsilon = 10$ and $MinPts = 3$) has merged the two upper clusters together and the two lower clusters together as they had connecting points between them and DBSCAN was unable to distinguish connecting points from core and border points. For the second (8,000 data points), third (8,000 data points) and fourth (8,000 data points) rows, both LVC and DBSCAN were able to detect the correct clusters. In the second row, LVC parameters were $\epsilon = 0.9, t = 0.4$ and $\epsilon_{expand} = 0.988$, while DBSCAN parameters were $\epsilon = 5.9$ and $MinPts = 4$ and for the third row, LVC parameters were $\epsilon = 0.9, t = 0.4$ and $\epsilon_{expand} = 0.982$, while DBSCAN parameters were $\epsilon = 5$ and $MinPts = 4$. Finally, for the fourth row, LVC parameters were $\epsilon = 0.9, t = 0.4$ and $\epsilon_{expand} = 0.99$, while DBSCAN parameters were $\epsilon = 3$ and $MinPts = 4$.

Spectral clustering and mean shift were unable to detect appropriate clusters except for the last row image where spectral clustering was able to detect the correct clusters. Spectral clustering problem is raised from using K-means as the clustering algorithm after performing the dimensionality reduction and thus the clusters tend to have spherical shapes.

4.5.2 High Dimensional Datasets Clustering

Four high dimensional datasets are used to assist the performance of LVC, which are Synthetic Control Charts (SCC), ecoli, COIL20 and pen digits datasets [4]. The characteristics of these datasets are shown in table 4.2. In this subsection, LVC technique is compared against K-means, DBSCAN, spectral clustering (SC) and affinity propagation. Two versions of affinity propagation are used, which are affinity propagation median (APM) that sets the preference vector of the affinity propagation to the median vector of the similarity matrix and affinity propagation K (APK) which adjusts the preference vector given the number of clusters. In addition, three measures are used for the comparisons, which are:

- **F-measure:** which is the weighted average of each cluster i F-measure $F(i)$, where $F(i) = \frac{2P(i)R(i)}{P(i)+R(i)}$, $P(i)$ is the precision of cluster i and $R(i)$ is the recall of cluster i.
- **Jaccard coefficient:** which is the distribution of the Jaccard coefficient for every cluster produced by the selected technique compared to the most similar cluster in the ground truth results, as defined in [19].
- **Running time:** which is the total time required to detect the clusters.

Clustering parameters were adjusted based on the best F-measure and Jaccard coefficient for LVC and DBSCAN. For spectral clustering and affinity propagation K (APK), k was set to the number of classes. In addition, euclidean distance is used to measure the distance between points for ecoli, COIL20 and pen digits datasets. To convert the euclidean distances into similarity values, we normalize the euclidean distances and then let the similarities equal to 1 - euclidean distances. For Synthetic Control Charts, we use Pearson correlation as our similarity measure as it was reported to perform better in the control charts datasets.

Table 4.3 shows that LVC (using $\epsilon = 0.75$, $t = 0.5$ and $\epsilon_{expand} = 0.5$) has achieved the best clustering quality in SCC dataset by 10% higher than DBSCAN (using $\epsilon = 0.5$ and $MinPts = 1$)

[4] You can download the used datasets from the following links:

- **SCC:** https://archive.ics.uci.edu/ml/datasets/Synthetic+Control+Chart+Time+Series
- **Ecoli:** https://archive.ics.uci.edu/ml/datasets/Ecoli
- **COIL20:** http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php
- **Pen Digits:** https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

in the Jaccard coefficient and is the third fastest approach with gap of around 0.2 seconds to spectral clustering. In SCC dataset, LVC was able to obtain 3 correct clusters out of the 5 clusters as it was merging the "Up-trend" cluster with the "Up-shift" clusters and was merging "Down-trend" cluster with the "Down-shift" cluster, because of the persistence of strong connecting points. Additionally, DBSCAN was also merging the two "Up" clusters and the two "Down" clusters. DBSCAN was also unable to discover the "Normal" cluster as it has relatively low density as mentioned in [45]. LVC (using $\epsilon = 0.44$, $t = 0.2$ and $\epsilon_{expand} = 0.8$) is also the best in the Ecoli dataset as shown in table 4.4; it had achieved higher Jaccard coefficient over DBSCAN (using $\epsilon = 0.16$ and $MinPts = 9$) by 14% and is the second fastest approach in the running time after spectral clustering with a gap of only 0.02 seconds. Moreover, in COIL20 dataset, LVC (using $\epsilon = 0.78$, $t = 0.9$ and $\epsilon_{expand} = 0.7$) has exceeded spectral clustering and affinity propagation by around 20% in the F-measure and has accomplished the best running time as shown in table 4.5. It is worth noting that DBSCAN (using $\epsilon = 5$ and $MinPts = 1$) has a comparable clustering quality with LVC in COIL20 dataset but LVC is much faster than DBSCAN by around 38 seconds. Finally, for pen digits dataset, affinity propagation was unable to run as it needed more memory than our machine (8 GB RAM). As shown in table 4.6, the best clustering quality is achieved by spectral clustering, while LVC (using $\epsilon = 0.51$, $t = 0.42$ and $\epsilon_{expand} = 0.889$) is the second best with a gap of 2% in the F-measure and 1% in the Jaccard coefficient. However, LVC was able to accomplish that in time much less than spectral clustering with a speedup of around 20X. Furthermore, LVC was better than DBSCAN (using $\epsilon = 28$ and $MinPts = 3$) by 7% in the F-measure and 6% in the Jaccard coefficient.

4.5.3 Twitter Topic Detection Task

In this subsection, we use LVC technique (using $\epsilon = 0.3, t = 0.1$ and $\epsilon_{expand} = 0.7$) to detect topics from Twitter streams by reporting the centroids of the clusters detected by LVC. Twitter FA cup dataset [2] is used to evaluate this task which contains around 20,000 tweets about football cup event and the number of topics k to be detected is set to 2, 4, 6, 8 and 10. Moreover, the dataset was represented using TF-IDF scheme and cosine similarity was used. Four measures [5] are used to compare the techniques, which are:

- **Topic recall:** Percentage of topics successfully retrieved.
- **Term precision:** Number of correct keywords in the detected topics over the total number of keywords in these detected topics.

[5]The evaluation code was used from http://www.socialsensor.eu/results/datasets/72-twitter-tdt-dataset

- **Term recall:** Number of correct keywords over the total number of keywords in the ground truth topics.
- **Running time:** Time required to detect the topics.

In this task, LVC is compared to other clustering techniques that are used for topic detection and scale well with the Twitter data like K-means, DBSCAN (using $\epsilon = 0.5$ and $MinPts = 10$) and incremental batch K-means (IBK). Parameters were chosen empirically based on the ones that yield the best topic detection quality. Table 4.7, 4.8, 4.9 and 4.10 show the topic detection quality and the running time. As shown in the tables, LVC achieves the best topic recall in all the time slots except when the number of detected topics is 2 and achieves the best term recall and term precision except when the number of topics is 4. For the running time, the worst running time was for DBSCAN, then LVC, then K-means variations. LVC running time was more than K-means variations by only around 3 seconds, which is still a reasonable time for topic detection.

4.6 Summary

In this chapter, we have proposed the Local Variance-based Clustering (LVC) and we have evaluated LVC using different datasets from different domains. Additionally, we have evaluated LVC in the topic detection task. In the next chapter, we will conclude the thesis and discuss several future work extensions.

Table 4.1: Time and Memory Complexity of Different Clustering Techniques

Approach	**Time Complexity**	**Memory Complexity**
LVC	O($dnlogn$)	O($kn + dn$)
DBSCAN	O($dnlogn$)	O(dn)
K-means	O($ncdi$)	O(dn)
Affinity Propagation	O(in^2)	O(n^2)
Spectral Clustering	O(dn^2)	O(nt)

Table 4.2: Datasets used for Experiments

Approach	**Data Points Number**	**Dimensions Number**
SCC	600	60
Ecoli	336	7
COIL20	1440	1024
Pen Digit	10992	16

Table 4.3: Synthetic Control Charts (SCC) Dataset Results

Approach	**Fmeasure**	**Jaccard**	**Running Time (Seconds)**
LVC	**0.78**	**0.60**	0.3
DBSCAN	0.61	0.50	0.54
K-means	0.66	0.43	0.28
APM	0.43	0.26	2.73
APK	0.53	0.33	48.54
SC	0.59	0.34	**0.06**

Table 4.4: Ecoli Dataset Results

Approach	**Fmeasure**	**Jaccard**	**Running Time (Seconds)**
LVC	0.78	**0.64**	0.04
DBSCAN	**0.8**	0.50	0.08
K-means	0.63	0.35	**0.02**
APM	0.29	0.11	0.74
APK	0.67	0.41	9.47
SC	0.62	0.34	0.13

Table 4.5: COIL20 Dataset Results

Approach	**Fmeasure**	**Jaccard**	**Running Time (Seconds)**
LVC	**0.87**	0.64	**0.25**
DBSCAN	0.82	**0.68**	38.44
K-means	0.66	0.39	2.46
APM	0.26	0.14	12.17
APK	0.64	0.37	99.50
SC	0.60	0.34	0.27

Table 4.6: Pen Digits Dataset Results

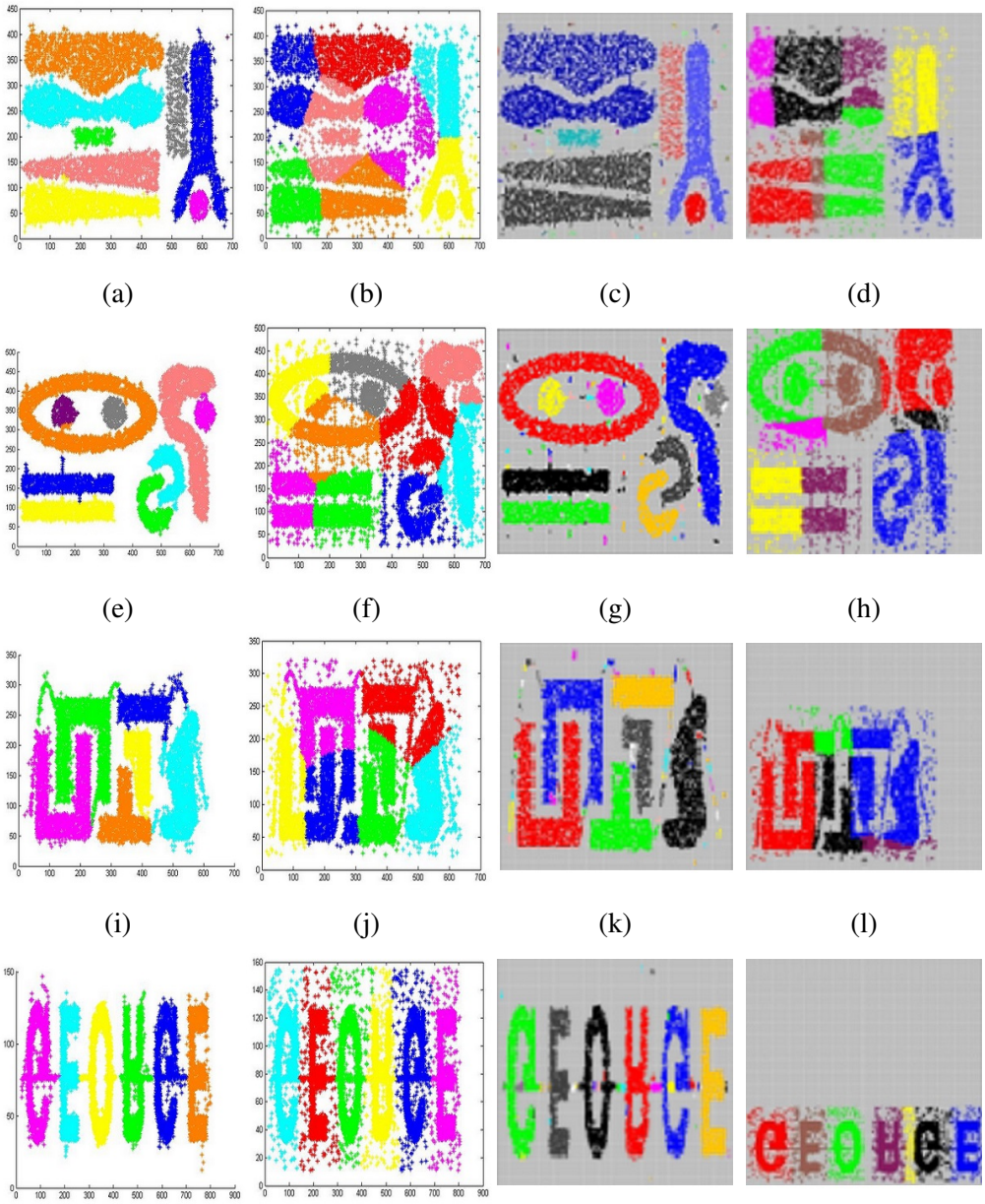
Approach	**Fmeasure**	**Jaccard**	**Running Time (Seconds)**
LVC	0.76	0.47	12.69
DBSCAN	0.69	0.41	27.03
K-means	0.64	0.33	8.76
SC	**0.78**	**0.48**	260.32

Table 4.7: FA cup Topic Recall

Approach	**k=2**	**k=4**	**k=6**	**k=8**	**k=10**
LVC	0.769	**0.923**	**0.923**	**0.923**	**0.923**
DBSCAN	0.769	0.769	0.769	0.769	0.769
K-means	0.769	0.846	0.769	**0.923**	**0.923**
IBK	**0.846**	0.769	0.846	0.846	0.692

Table 4.8: FA cup Term Precision

Approach	**k=2**	**k=4**	**k=6**	**k=8**	**k=10**
LVC	**0.187**	0.178	**0.178**	**0.178**	**0.178**
DBSCAN	0.167	0.167	0.167	0.167	0.167
K-means	0.167	**0.182**	0.167	0.161	0.156
IBK	0.176	0.173	0.152	0.139	0.156



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

Figure 4.6: Clustering results on Chameleon 2D datasets. Column one, two, three and four represent the results for LVC, spectral clustering, DBSCAN and mean shift respectively

Table 4.9: FA cup Term Recall

Approach	**k=2**	**k=4**	**k=6**	**k=8**	**k=10**
LVC	**0.622**	0.582	**0.582**	**0.582**	**0.582**
DBSCAN	0.556	0.556	0.556	0.556	0.556
K-means	0.556	**0.600**	0.556	0.537	0.509
IBK	0.580	0.578	0.500	0.460	0.512

Table 4.10: FA cup Running Time

Approach	**k=2**	**k=4**	**k=6**	**k=8**	**k=10**
LVC	3.186	3.186	3.186	3.186	3.186
DBSCAN	5.150	5.150	5.150	5.150	5.150
K-means	**0.096**	**0.129**	**0.147**	0.161	**0.189**
IBK	**0.096**	**0.129**	0.164	**0.159**	0.206

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we have proposed two different scalable techniques for detecting topics from Twitter streams. The first technique is a scalable Exemplar-based topic detection approach, which is implemented using Apache Giraph. Results show that Giraph sliding windows implementation speeds up the native implementation up to a factor of 19X using $100,000$ tweets, while achieving topic quality similar to or higher than the native implementation topic quality. Giraph approach is compared to other distributed approaches like distributed K-means, LSA, LDA and NMF and is shown to achieve the highest topic quality with good running time. Additionally, Giraph approach is deployed to detect topics from real-time Twitter streams and its scalability is shown.

The second proposed technique for topic detection is a LVC, which uses a local variance measure to calculate the density of the points based on the variance of their similarity histogram. It was shown that using the local variance measure distinguishes well between core points, border points, connecting points and outliers. Experimental results show that LVC outperformed K-means, DBSCAN, spectral clustering and affinity propagation in clustering quality using control charts, ecoli and images datasets, while maintaining a good running time. Moreover, results show that LVC was able to detect topics from twitter with higher topic recall by around 15% and higher term precision by around 3% over DBSCAN.

5.2 Future Work

5.2.1 Extend Exemplar-based Topic Detection to "Think-like Graph" Model

A new "Think like a Graph" Model is proposed in [40]. The basic idea behind the new model is that the communication between the vertices within the same partition can be performed without exchanging messages, as all the information about these vertices are located on the same partition. The new model uses the information that each partition has, so that the information can flow freely inside one partition. In this paradigm, the computations are expressed in terms of what each partition has to do as opposed to what each vertex has to do in the vertex-centric paradigm. Employing this model will reduce the running time. However, to reduce the running time the system assumes that the vertices are partitioned such that there are no crossing edges between partitions, while this process is very costly. Thus, an efficient partitioning heuristic is needed to make use of the "Think like a graph model".

5.2.2 Exemplar-based Topic Detection Partitioning Strategies

Using a partitioning strategy that reduces the number of edges between different partitions will reduce the communication overhead. Although graph partition problems fall under the category of NP-hard problem, the following heuristic partitioning strategy can be used to place the tweet vertex and the term vertices that are connected to it on one partition. However, this may introduce imbalanced partitions. To alleviate the imbalanced partitions problem, one of the dynamic load balancing techniques as in [15] or graph Voronoi diagram based partitioner used by Blogel [43] can be used.

5.2.3 Exemplar-based Topic Detection Parameters Tuning

As the parameters k, ϵ and the window size are important parameters that have great effect on the running time and the detected topic quality of the Exemplar-based topic detection approach. It is essential to select these parameters carefully. Therefore, several approaches are needed to be developed to automatically tune the values of these three parameters and to adjust them dynamically in different time slots as the tweets keep coming.

5.2.4 Distributed Local Variance-based Clustering

As shown in this thesis, Local Variance-based Clustering is scalable as it runs in O($nlogn$) where n is the number of data points. However, it assumes that the data matrix fits in the memory of one machine, which is not a valid assumption in case of big data. Therefore, extending the computations of Local Variance-based Clustering to work on a distributed machines is an important task that will enhance the scalability of the approach.

5.2.5 Local Variance-based Clustering Parameters Tuning

Local Variance-based Clustering uses three parameters which are ϵ, t and ϵ_{expand}, which have a large effect on the quality of the detected clusters. Thus, developing an automatic parameter tuning techniques to adjust these parameters is essential.

References

[1] Manoj K Agarwal, Krithi Ramamritham, and Manish Bhide. Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments. *Proceedings of the VLDB Endowment*, 5(10):980–991, 2012.

[2] Luca Maria Aiello, Georgios Petkos, Carlos Martin, David Corney, Symeon Papadopoulos, Ryan Skraba, Ayse Goker, Ioannis Kompatsiaris, and Alejandro Jaimes. Sensing trending topics in twitter. *Multimedia, IEEE Transactions on*, 15(6):1268–1282, 2013.

[3] Michael W Berry, Murray Browne, Amy N Langville, V Paul Pauca, and Robert J Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[5] Ceren Budak, Theodore Georgiou, Divyakant Agrawal, and Amr El Abbadi. Geoscope: Online detection of geo-correlated information trends in social networks. *Proceedings of the VLDB Endowment*, 7(4), 2013.

[6] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] Ahmed Elbagoury, Rania Ibrahim, Ahmed Farahat, Mohamed Kamel, and Fakhri Karray. Exemplar-based topic detection in twitter streams. In *Ninth International AAAI Conference on Weblogs and Social Media*, 2015.

[9] Ahmed Elgohary, Ahmed K Farahat, Mohamed S Kamel, and Fakhri Karray. Embed and conquer: scalable embeddings for kernel k-means on mapreduce. *CoRR abs/1311.2334*, 2013.

[10] Tamer Elsayed, Jimmy Lin, and Douglas W Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 265–268. Association for Computational Linguistics, 2008.

[11] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *SDM*, pages 47–58. SIAM, 2003.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[13] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[14] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[15] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[16] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[17] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514. International World Wide Web Conferences Steering Committee, 2013.

[18] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[19] Christian Hennig. Cluster-wise assessment of cluster stability. *Computational Statistics & Data Analysis*, 52(1):258–271, 2007.

[20] Andreas Hotho, Steffen Staab, and Gerd Stumme. Ontologies improve text document clustering. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 541–544. IEEE, 2003.

[21] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 332–339. ACM, 1994.

[22] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[23] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[24] Pei Lee, Laks VS Lakshmanan, and Evangelos E Milios. Incremental cluster evolution tracking from highly dynamic network data. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 3–14. IEEE, 2014.

[25] Rui Li, Kin Hou Lei, Ravi Khadiwala, and KC-C Chang. Tedas: A twitter-based event detection and analysis system. In *Data engineering (icde), 2012 ieee 28th international conference on*, pages 1273–1276. IEEE, 2012.

[26] Rui Li, Shengjie Wang, and Kevin Chen-Chuan Chang. Towards social data platform: Automatic topic-focused monitor for twitter stream. *Proceedings of the VLDB Endowment*, 6(14):1966–1977, 2013.

[27] Tao Li and Sarabjot Singh Anand. Diva: a variance-based clustering approach for multi-type relational data. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 147–156. ACM, 2007.

[28] Young Won Lim and Sang Uk Lee. On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques. *Pattern recognition*, 23(9):935–952, 1990.

[29] Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*, 2014.

[30] Andrew J McMinn, Yashar Moshfeghi, and Joemon M Jose. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 409–418. ACM, 2013.

[31] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. Improving lda topic models for microblogs via tweet pooling and automatic labeling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 889–892. ACM, 2013.

[32] Gilad Mishne, Jeff Dalton, Zhenghua Li, Aneesh Sharma, and Jimmy Lin. Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1147–1158. ACM, 2013.

[33] David Newman, Edwin V Bonilla, and Wray Buntine. Improving topic coherence with regularized topic models. In *Advances in neural information processing systems*, pages 496–504, 2011.

[34] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[35] Raymond T Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 14(5):1003–1016, 2002.

[36] Tim Oates, Matthew D Schmill, and Paul R Cohen. A method for clustering the experiences of a mobile robot that accords with human judgments. In *AAAI/IAAI*, pages 846–851, 2000.

[37] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 938–947. Association for Computational Linguistics, 2009.

[38] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[39] Ron Shamir and Roded Sharan. Algorithmic approaches to clustering gene expression data. In *Current Topics in Computational Biology*. Citeseer, 2001.

[40] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. From" think like a vertex" to" think like a graph. *Proceedings of the VLDB Endowment*, 7(3):193–204, 2013.

[41] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[42] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. Topicsketch: Real-time bursty topic detection from twitter. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 837–846. IEEE, 2013.

[43] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *Proceedings of the VLDB Endowment*, 7(14), 2014.

[44] Xiaohui Yan, Jiafeng Guo, Shenghua Liu, Xue-qi Cheng, and Yanfeng Wang. Clustering short text using ncut-weighted non-negative matrix factorization. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2259–2262. ACM, 2012.

[45] Noha A Yousri, Mohamed S Kamel, and Mohamed A Ismail. A distance-relatedness dynamic model for clustering high dimensional data of arbitrary shapes and densities. *Pattern Recognition*, 42(7):1193–1209, 2009.

[46] Mindi Yuan, Kun-Lung Wu, Gabriela Jacques-Silva, and Yi Lu. Efficient processing of streaming graphs for evolution-aware clustering. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 319–328. ACM, 2013.

[47] Zhenjie Zhang, Hu Shu, Zhihong Chong, Hua Lu, and Yin Yang. C-cube: Elastic continuous clustering in the cloud. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 577–588. IEEE, 2013.