Tribhuvan University
Institute of Science and Technology

Nepali Document Clustering using K-Means, Mini-Batch K-Means, and DBSCAN

Dissertation
Submitted to

Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements
for the Master's Degree in Computer Science and Information Technology

by
Aman Maharjan
Date (August, 2018)

Supervisor
Tej Bahadur Shahi

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than listed here have been used in this work.

_____

Aman Maharjan

Date: August, 2018

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my supervision by **Mr. Aman Maharjan** titled "**Nepali Document Clustering using K-Means, Mini Batch K-Means, and DBSCAN**" in partial fulfillment of the requirements for the degree of MSc in Computer Science and Information Technology be processed for the evaluation.

Asst. Prof. Tej Bahadur Shahi
Central Department of Computer Science and Information Technology,
Tribhuvan University,
Kathmandu, Nepal
(Supervisor)

Date: August, 2018

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that, we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Evaluation Committee

Asst. Prof. Nawaraj Poudel
Central Department of CSIT,
Tribhuvan University,
Kathmandu, Nepal
(Head of Department)

Asst. Prof. Tej Bahadur Shahi
Central Department of CSIT,
Tribhuvan University,
Kathmandu, Nepal
(Supervisor)

(External Examiner)

(Internal Examiner)

ACKNOWLEDGMENTS

I offer my profound gratitude to my supervisor Asst. Prof. Tej Bahadur Shahi (Tribhuvan University) for his generous advice, inspiring guidance and encouragement throughout my research for this dissertation. Without his kind and patient review of this work, it would have been impossible to complete this study.

I would like to extend my gratitude to Asst. Prof. Nawaraj Paudel (Head of Department, CDCSIT), Prof. Dr. Shashidhar Ram Joshi, Prof. Dr. Subarna Shakya, Prof. Dr. Arun Timilshina, Mr. Bikas Balami, Mr. Sarbin Sayami, Mr. Jagdish Bhatta, Mr. Arjun Saud, Mrs. Lalita Sthapit, and Mr. Dheeraj Kedar Pandey of CDCSIT, Tribhuvan University for their guidance and help throughout my Masters study.

I am indebted to my friends Mr. Sudon Prajapati and Mr. Subash Manadhar for inspiration during the research period. I am also thankful to all other people who have helped me directly or indirectly in the completion of this research.

ABSTRACT

Automated document clustering is the process of grouping documents into a small sets of meaningful and coherent collections. This research evaluates K-Means, Mini-Batch K-Means and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithms using four performance measures: Homogeneity, Completeness, V-Measure and Silhouette Coefficient in the context of Nepali documents. Features extraction is done using Term Frequency – Inverse Document Frequency (TFIDF) and TFIDF + Latent Semantic Indexing (LSI) combination. The empirical results shows that Mini-Batch K-Means performs better when using TFIDF only and K-Means performs better when using TFIDF + LSI. Similarly, in time constrained environments, the clustering time of Mini-Batch K-Means is better than other two algorithms.

Keywords: Clustering, Machine Learning, Nepali Document Clustering, K-Means, Mini-Batch K-Means, DBSCAN, TFIDF, LSI

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABBREVIATIONS

API	Application Programming Interface.
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies.
CSV	Comma-Separated Values.
DBSCAN	Density-Based Spatial Clustering of Applications with Noise.
EM	Expectation–Maximization.
GB	Giga-Byte.
HTML	Hyper Text Markup Language.
IDE	Integrated Development Environment.
KU	Kathmandu University.
LSI	Latent Semantic Indexing.
MPP	Madan Puraskar Pustakalaya.
NER	Named-Entity Recognition.
NLP	Natural Language Processing.
OPTICS	Ordering Points to Identify the Clustering Structure.
PoS	Part-of-Speech.
RAM	Random Access Memory.
SOM	Self Organizing Map.
SSD	Solid State Drive.
SVD	Singular Value Decomposition.

TFIDF Term Frequency – Inverse Document Frequency.

CHAPTER 1

BACKGROUND AND INTRODUCTION

1.1 Introduction

A wide range of research has already been done in the field of clustering. It is an active field of research due to its significance in areas like data mining, text mining, information retrieval, statistics, machine learning, biology, marketing and so on ([2]). The problem of clustering can be very useful in the text domain, where the objects to be clusters can be of different granularities such as documents, paragraphs, sentences or terms. Clustering is especially useful for organizing documents to improve retrieval and support browsing ([3]).

In the context of Nepal, more and more Nepali documents are created and stored in both online and offline forms each day. Manually clustering them into meaningful clusters is both tedious and error-prone. So, automatically clustering them using computers is highly desirable. As very little research has been done in this area in the past ([4–6]), this study intends to explore three well-known algorithms in Nepali document clustering.

As Nepali is a complex language, the clustering algorithms need to be aware of specific features related to the language beforehand. This work uses TFIDF and LSI for representing the features of the documents. It uses three algorithms K-Means, Mini-batch K-Means and DBSCAN to cluster the documents and then compares the accuracy and time taken to run the algorithms.

1.2 Problem Statement

Nepali is a morphologically rich and complex language. It is rich in vocabulary and has many inflectional and derivative words. Hence cluster analysis of documents in such a language is a challenging task. Compared to a language like English, not much research has been done in the field of Nepali document clustering. This includes research in preprocessing steps for Nepali documents as well [4–6]. So further research in this area is highly desirable.

Let D be the document set (corpus) and C be the predefined set of clusters, then, the problem of clustering is to find the mapping $f : D \rightarrow C$ such that intra-cluster distance is minimized and extra-cluster distance is maximized.

1.3 Objectives

The primary objective of this work is to study the effects and behavior of applying different clustering algorithms to Nepali documents. The specific objectives are:

1. To build feature representation scheme for Nepali text using TFIDF and LSI.
2. To cluster Nepali documents using K-Means, Mini-Batch K-Means and DBSCAN algorithms and analyze Homogeneity, Completeness, V-Measure and Silhouette Coefficient.

1.4 Scope and Limitations

This study focuses on studying the performance of K-Means, Mini-Batch K-Means and DBSCAN algorithms on a Nepali dataset. Analysis of cluster quality is based on four performance measures: Homogeneity, Completeness, V-Measure and Silhouette Coefficient. Performance analysis based on completion time has also been studied. In addition to these, the effects of feature extraction techniques TFIDF and LSI on the both analyses has been studied too.

Application Programming Interfaces (APIs) from Scikit-Learn library has been used for feature extraction as well as clustering. This choice means that DBSCAN computes the full $O(n^2)$ distance matrix and is costly, both memory-wise and runtime-wise, for large sample sizes. So this study limits the dataset size to 10,000 samples even though the other two algorithms can handle much larger sample sizes.

1.5 Background

1.5.1 Cluster Analysis

Cluster analysis is an exploratory data analysis tool. Given a sample of data vectors x defining the rows of a $(n \times K)$ data matrix $[X]$, the procedure will define groups and assign group memberships at varying levels of aggregation. Cluster analysis can bring out groupings in data that might otherwise be overlooked, possibly leading to an empirically useful classification for observed structure in the data [7]. Unlike classification, clustering and unsupervised learning do not rely on predefined classes and class-labeled training examples. For this reason, clustering is a form of learning by observation, rather than learning by examples [2].

1.5.2 Similarity and Distance Measures

To address the text data mining tasks of clustering, classification, and information retrieval, a notion of similarity or distance between the documents is necessary. Clusters should be comprised of points separated by small distances, relative to the distances between clusters. However, there are a wide variety of plausible definitions of distance in this context, and the results of a cluster analysis may depend quite strongly on the distance measure chosen [7, 8].

The most commonly used similarity measure in text data mining and information retrieval is the cosine of the angle between vectors representing the documents shown in eq. (1.1). Given two document vectors $\vec{a}$ and $\vec{b}$, the cosine of the angle between them, θ, is given by:

$$cos(\theta) = \frac{\vec{a}^T \vec{b}}{|\vec{a}|_2 |\vec{b}|_2} \tag{1.1}$$

where $|\vec{a}|_2$ is the usual L_2 norm of vector $\vec{a}$. Larger values of this measure indicate documents are close together, and smaller values indicate the documents are further apart [8, 9].

The most popular distance measure in cluster analysis is Euclidean distance which is defined as:

$$d_{i,j} = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2} \tag{1.2}$$

where $i = (x_{i,1}, x_{i,2}, ..., x_{i,n})$ and $j = (x_{j,1}, x_{j,2}, ..., x_{j,n})$ are two n dimensional objects [2].

In some cases the ordinary Euclidean distance may be a poor choice. For example, if the elements of the data vectors are unlike variables with inconsistent measurement units, the variable with the largest values will tend to dominate the Euclidean distance. In such cases it is better to use the weighted version of Euclidean distance between the vectors:

$$d_{i,j} = \sqrt{\sum_{k=1}^{n} w_k (x_{i,k} - x_{j,k})^2} \tag{1.3}$$

For $w_k = 1, \forall\ k = 1, 2, ..., n$, eq. (1.3) reduces to ordinary Euclidean distance [7].

Another well-known distance metric is Manhattan (or city block) distance, defined as:

$$d_{i,j} = \sum_{k=1}^{n} w_k |x_{i,k} - x_{j,k}| \tag{1.4}$$

The Manhattan distance metric is consistently more preferable than the Euclidean distance metric for high dimensional data mining applications [10].

Minkowski distance is a generalization of both Euclidean distance and Manhattan distance. It is defined as:

$$d_{i,j} = \sqrt[p]{\sum_{k=1}^{n} w_k |x_{i,k} - x_{j,k}|^p}, \; p \geqslant 1 \tag{1.5}$$

This distance is also called L_p norm. It represents the Manhattan distance when $p = 1$ (L_1 norm) and Euclidean distance when $p = 2$ (L_2 norm) [2, 7].

1.5.3 Distance Matrix

Having chosen a distance measure to quantify dissimilarity or similarity between pairs of vectors i and j, the next step in cluster analysis is to calculate the distances between all $n(n–1)/2$ possible pairs of the n observations. It can be convenient, either organizationally or conceptually, to arrange these into a $(n \times n)$ distance matrix of D:

$$\begin{bmatrix} 0 & & & & \\ d_{2,1} & 0 & & & \\ d_{3,1} & d_{3,2} & 0 & & \\ \vdots & \vdots & \vdots & & \\ d_{n,1} & d_{n,2} & \cdots & \cdots & 0 \end{bmatrix} \tag{1.6}$$

This symmetric matrix has zeros along the main diagonal, indicating zero distance between each x and itself [2, 7].

1.5.4 Clustering Methods

Many clustering algorithms can be found in existing literature. It is often difficult to categorize these algorithms in definite categories as the algorithms may have features from several categories. However, it is still useful to divide them into separate groups. The major clustering algorithms can be broadly organized into following categories [2]:

- Partitioning Methods
- Hierarchical Methods

- Density-Based Methods
- Grid-Based Methods
- Model-Based Methods

1.5.4.1 Partitioning Methods

Given a dataset of n objects, a partitioning method constructs k partitions of the data, where each partition represents a cluster and $0 < k \leqslant n$. The partitions must satisfy the following requirements:

1. each group must contain at least one object, and
2. each object must belong to exactly one group.

These algorithms minimize a given clustering criterion by iteratively relocating data points between clusters until a locally optimal partition is attained. The general criterion of a good partitioning is that objects in the same cluster are "close" or related to each other, whereas objects of different clusters are "far apart" or dissimilar.

To achieve global optimality in partitioning-based clustering would require the exhaustive enumeration of all of the possible partitions. However, finding the globally optimal partition is known to be NP-hard problem and exhaustive methods are not useful in practice. Instead, most applications adopt one of a few popular heuristic methods, such as the K-Means algorithm, where each cluster is represented by the mean value of the objects in the cluster, and the K-Medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. To find clusters with complex shapes and for clustering very large data sets, partitioning-based methods need to be extended [2, 11].

1.5.4.2 Hierarchical Methods

Hierarchical techniques produce a nested sequence of partitions, with a single, all-inclusive cluster at the top and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining two clusters from the next lower level (or splitting a cluster from the next higher level). The result of a hierarchical clustering algorithm can be graphically displayed as tree, called a dendogram [12].

A hierarchical method can be classified as being either agglomerative or divisive, based on how the hierarchical decomposition is formed. The agglomerative (bottom-up) approach starts with

each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one (the topmost level of the hierarchy), or until a termination condition is reached. The divisive (top-down) approach, starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until each object is in one cluster, or until a termination condition is reached.

In hierarchical methods, once a merge or split is done, it can never be undone. This rigid behavior is leads to smaller computation costs. However, this may pose problems when error corrections have to be done. Hierarchical clustering can be improved in two ways [2]:

1. performing careful analysis of object "linkages" at each hierarchical partitioning, such as in Chameleon
2. first using a hierarchical agglomerative algorithm to group objects into micro-clusters, and then performing macro-clustering on these using another method such as iterative relocation, as in Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH).

1.5.4.3 Density-Based Methods

Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty at discovering clusters of arbitrary shapes. Other clustering methods have been developed based on the notion of density. They works by detecting areas where points are concentrated and where they are separated by areas that are empty or sparse. A given cluster can grow as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold. This means, for each data point within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of points. Such a method can be used to filter out noise or outliers and discover clusters of arbitrary shape. DBSCAN and Ordering Points to Identify the Clustering Structure (OPTICS) are typical examples of density-based partition methods [2].

1.5.4.4 Grid-Based Methods

Grid-based methods quantize the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on this grid. The main advantage of this approach is its fast processing time as it does not depend upon the number of objects in the dataset. Quality of cluster can be controlled by varying the number of cells in the quantized space.

STING is a typical example of a grid-based method. Another example is WaveCluster, which is a hybrid of both grid-based and density-based methods [2, 13, 14].

1.5.4.5 Model-Based Methods

Clustering algorithms can also be developed based on probability models, such as the finite mixture model for probability densities. The word model is usually used to represent the type of constraints and geometric properties of the covariance matrices. In the family of model-based clustering algorithms, one uses certain models for clusters and tries to optimize the fit between the data and the models. In the model-based clustering approach, the data are viewed as coming from a mixture of probability distributions, each of which represents a different cluster. In other words, in model-based clustering, it is assumed that the data are generated by a mixture of probability distributions in which each component represents a different cluster. Thus a particular clustering method can be expected to work well when the data conform to the model. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking "noise" or outliers into account and thus yielding robust clustering methods.

Expectation–Maximization (EM) is an algorithm that performs expectation-maximization analysis based on statistical modeling. COBWEB is a conceptual learning algorithm that performs probability analysis and takes concepts as a model for clusters. Self Organizing Map (SOM) is a neural network-based algorithm that clusters by mapping high-dimensional data into a 2-dimensional or 3-dimensional feature map [2, 15].

1.6 Report Organization

This dissertation consists of six chapters which will cover the study of applying three clustering algorithms on Nepali dataset. Here is an overview of the content in each chapter:

- **Chapter 1:** This chapter introduces the problem, gives some background information and lists out the objectives of the study. It also discusses the scope and limitations of the study.
- **Chapter 2:** This chapter discusses about previous related works on cluster analysis and Nepali document clustering done by other researchers.
- **Chapter 3:** This chapter covers the selected methodology that are used in this research. It explains about the dataset preparation method, system architecture, preprocessing, text representation, clustering algorithms used and performance evaluation parameters.
- **Chapter 4:** This chapter discusses the implementation aspects of the research. It covers the programming languages and frameworks, algorithms, pseudo-codes and parameter tuning methods used in this study.

- **Chapter 5:** This chapter discusses the evaluation process of the study. Evaluation is based on both cluster quality and completion time.
- **Chapter 6:** This chapter concludes the research and presents some recommendations and future works to improve this study.

CHAPTER 2

LITERATURE REVIEW

2.1 Cluster Analysis

Cluster analysis or simply clustering is the process of dividing a set of objects into a group of similar subsets (called a cluster) with respect to a given similarity measure. It was first used in anthropology by Driver and Kroeber in 1932. Later, it was popularized in the field of psychology by Zubin in 1938, R. Tryon in 1939 and Cattell in 1943. Clustering emerged as a major topic in the 1960's and 1970's when the monograph “Principles of Practice of Numerical Taxonomy” by R.R. Sokal, published in 1963, motivated world-wide research on clustering methods [16–18].

Initial researches on document clustering focused on improving performance measure and efficiency. C.J. Rijsbergen mentions methods of improving precision and recall in information retrieval systems in his book published in 1979 [19].

Research were also done to increase efficiency of information retrieval searches by finding the nearest neighbors of a document. In 1985, C. Buckley and A.F. Lewit, published a simple algorithm for increasing the efficiency of information retrieval searches which are implemented using inverted files. The algorithm employs knowledge about the methods used for weighting document and query terms in order to examine as few inverted lists as possible. An extension to the basic algorithm allows greatly increased performance optimization at a modest cost in retrieval effectiveness. They examined several different term weighting models showing the optimization possible with each one [20].

Recently clustering also been used in browsing documents. One such study was done in 1992 by D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey. They noted that document clustering has not been well received as an information retrieval tool. Objections to its use fall into two main categories: first, that clustering is too slow for large corpora (with running time often quadratic in the number of documents); and second, that clustering does not appreciably improve retrieval. They argued that these problems arise only when clustering is used in an attempt to improve conventional search techniques. However, looking at clustering as an information access tool in its own right obviates these objections, and provides a powerful new access paradigm. They presented a document browsing technique that employs document clustering as its primary operation in that paper. They also presented a fast (linear time) clustering algorithm which support this interactive browsing paradigm [21].

In 1997, O. Zamir described several novel clustering methods which intersect the documents in a cluster to determine the set of words or phrases shared by all the documents in a cluster. They evaluated these intersection-based clustering methods on collections of snippets returned from web search engines. They showed that word-intersection clustering produces superior clusters and does so faster than standard techniques. They also showed that their $O(n\ log\ n)$ time phrase-intersection clustering methods produces comparable clusters and does so more than two orders of magnitude faster than word-intersection [22].

Since then, clustering has been used in a large number of fields such as machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, computer graphics, archaeology, psychology and marketing [16–18].

2.2 Clustering Algorithms

2.2.1 K-Means

The term K-Means was first used by J. MacQueen in 1967, for his sequential, "single-pass" algorithm for (asymptotically) minimizing the continuous sum-of-squares criterion. In the paper he describes the algorithm as a process for partitioning an n-dimensional population into k sets on the basis of a sample. He notices that the process appears to give partitions which are reasonably eficient in the sense of within-class variance. He also mentions that the k-means procedure is easily programmed and is computationally economical, so that it is feasible to process very large samples on a digital computer. He also points out possible applications which include methods for similarity grouping, nonlinear prediction, approximating multivariate distributions, and nonparametric tests for independence among several variables [23, 24].

Though the term K-Means was first termed in 1967, the idea itself, i.e., dividing a body into meaningful groups of n clusters goes to H. Steinhaus [25], who published a mathematical treatment on the subject in 1957.

In 1965, E.W. Forgy published essentially the same method, which is why it is sometimes referred to as Lloyd-Forgy [26].

2.2.2 Mini-Batch K-Means

Mini-Batch K-Means has been proposed as an alternative to the K-Means algorithm for clustering massive datasets. The advantage of this algorithm is to reduce the computational cost by not using all the dataset each iteration but a subsample of a fixed size. This strategy reduces the number of distance computations per iteration at the cost of lower cluster quality.

Mini-Batch K-Means was first presented by D. Scully in 2010. It was one of the two modifications to the popular K-Means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications. The Mini-Batch method presented in the paper reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent [27].

2.2.3 DBSCAN

DBSCAN is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

DBSCAN was proposed by M. Ester, H.P. Kriegel, J. Sander and X. Xu in 1996. Their paper noted that prior clustering algorithms are attractive for the task of class identification in spatial databases, however, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to determine the input parameters, discovery of clusters with arbitrary shape and good efficiency on large databases. The well-known clustering algorithms offer no solution to the combination of these requirements. So they presented the new clustering algorithm DBSCAN relying on a density-based notion of clusters which is designed to discover clusters of arbitrary shape. DBSCAN requires only one input parameter and supports the user in determining an appropriate value for it. They performed an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and real data of the SEQUOIA 2000 bench-mark. The results of their experiments demonstrated that DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS [28], and that DBSCAN outperforms CLARANS by a factor of more than 100 in terms of efficiency [29].

2.3 Related Work in Nepali NLP

Work on Nepali Natural Language Processing (NLP) started from as far back as 2005, with the release of the first Nepali spell-checker. The same year, an English to Nepali machine translation system दोभासे (Dobhase) was completed jointly by Kathmandu University (KU) and Madan Puraskar Pustakalaya (MPP) as a Asia-Pacific Development Information Program [30, 31].

Since then, research has been done on various areas of Nepali NLP. Work on Part-of-Speech

(PoS) tagging was done by T.B. Shahi, T.N. Dhamala and B. Balami in 2013 [32] as well as A. Paul, B.S. Purkayastha and S. Sarkar in 2015 [33]. In 2004, B.K. Bal published his work on design and implementation issues as well as the linguistic aspects of the Morphological Analyzer and a Stemmer for Nepali [30]. Similarly S. Bam and T. Shahi worked on Named-Entity Recognition (NER) in 2014 [34]. The same year A. Dey, A. Paul and B.S. Purkayastha also published another work in the same field [35].

2.4 Related Work in Nepali Document Clustering

In 2014, S. Sarkar, A. Roy and B.S. Purkayastha presented a comparative analysis of three algorithms namely K-Means, Particle Swarm Optimization (PSO) and hybrid PSO+K-Means algorithm for clustering of Nepali text documents using WordNet. They represented text in terms of synsets corresponding to a word and performed experimental evaluation by using intra-cluster similarity and inter-cluster similarity [4].

The same year, A. Neupane published another paper that was used to create Nepali character dataset using semi-supervised clustering approach. Two algorithms EM and K-Means were used to create the database using extracted features from both handwritten and scanned Nepali text [5].

C. Sitaula also proposed an algorithm which combines the advantage of classical vector space model to cluster the semantic texts and ideas from fuzzy logic in 2014. The algorithm treated text having similar context words as semantic texts. It used the concept of advanced enhanced vector space model obtained by adding TFIDF with fuzzy membership value and perform the cosine operation in order to calculate the semantic distance between text [6].

Research in Nepali document clustering is sparse at best event to this day. Cluster analysis on large number of documents is still non-existent. So, this dissertation is an attempt to contribute some research in this area [4–6, 36].

CHAPTER 3

METHODOLOGY

3.1 Dataset Preparation

The official written script for Nepali is Devnagari (देवनागरी) which is an abugida (alphasyllabary) used commonly in Nepal, Bhutan and India. This script is also shared by other languages like Sanskrit, Hindi, Marathi and so on, due to which it contains unicode code points from U+0900 to U+097F [37] to encompass all their characters and symbols. Only a subset of these code points are used in current version of Nepali language [38, 39] and they can be further subdivided into 13 vowels (table 3.1a), 36 consonants (table 3.1b), 12 dependent vowel signs (table 3.1c), 10 numerals (table 3.1d) and various other signs (table 3.1e).

The vowels अं and अः and consonants क्ष, त्र and ज्ञ do not have separate code points in unicode [37] nor do they appear in the Nepali Brihat Shabdakosh (नेपाली बृहत् शब्दकोश) [39], the de facto standard dictionary for Nepali. But, they are still considered single entities in Nepali [38]. For the implementation part of this dissertation, they were considered as multiple characters or code points. Also, only a subset of table 3.1e was considered as the set of valid code points and all of the numerals in table 3.1d were considered invalid. The valid subset of code points can be found in appendix A.2.

अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ	अं	अः

(a) Vowels

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ
ट	ठ	ड	ढ	ण	त	थ	द	ध	न
प	फ	ब	भ	म	य	र	ल	व	
श	ष	स	ह	क्ष	त्र	ज्ञ			

(b) Consonants

ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ	ं	ः

(c) Dependent Vowel Signs

०	१	२	३	४	५	६	७	८	९

(d) Numerals

ँ	्	ऽ	ॐ	॰	।	॥	?	,	:	;

(e) Various Signs

Table 3.1: Nepali Character Set

For the purpose of this study, the dataset was collected from various online Nepali News portals like `http://www.ekantipur.com`, `http://www.onlinekhabar.com`, `http://www.setopati.com` etc., using a web crawler. The dataset was merged with some secondary data used in a recent Nepali News Classification study [36]. Altogether, a dataset of 10,000 sample was created.

3.2 System Architecture

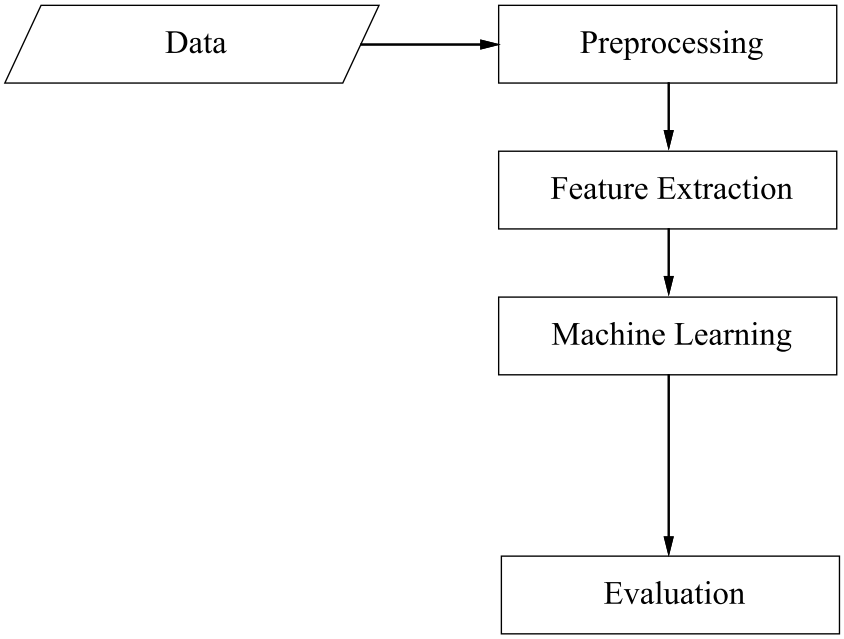


Figure 3.1: System Architecture

3.3 Preprocessing

The data in a raw corpus contains many unnecessary characters and words that do not contribute much to the clustering process. Filtering out these noisy data speeds up and simultaneously improves the result of cluster analysis [36, 40]. The following preprocessing steps were used on the corpus:

1. **Tokenization:** This step breaks each individual documents in the corpus, first into sentences and then into words. Since the corpus is in Nepali, related punctuation marks like devnagari danda (।), devnagari double danda (॥) and question mark can be used to break down sentences whereas space, comma, colon and semicolon can be used to break down words.

2. **Special Symbol and HTML Tag Removal:** Punctuation marks and HTML tags, which do not have any significance to the corpus, are removed.

3. **Stop Word Removal:** Stop words are the words which have very high frequency in the corpus. They either do not contribute anything or their contribution is negligible in differentiating documents. So common Nepali stop words like म, हामी, आफू, छ, हो, हुन्छ, भयो etc. are removed.

4. **Stemming:** Stemming is the process of removing affixes from words. Affixes may be either inflectional or derivational [36]. In Nepali, the meaning of compound words created using derivational affixes are often very different from the root or stem words [41]. So this work will focus only on stemming inflectional affixes, which reduces the lexicons to root form without changing their overall meaning.

3.4 Text Representation

3.4.1 TFIDF

The classic formula for TFIDF is:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{3.1}$$

where, $w_{i,j}$ is the weight for term i in document j, N is the number of documents in the corpus, $tf_{i,j}$ is the term frequency of term i in document j and df_i is the document frequency of term i in the corpus. The main idea behind TFIDF is that the terms in any document can be divided into words with eliteness and words without eliteness [42], which basically means whether a term is relevant or not with the topic of the document.

3.4.2 LSI

LSI [43] is a popular linear algebraic indexing method to produce low dimensional representation. The idea behind LSI is to take advantage of implicit higher order structure in the association of terms with documents (“semantic structure”) in order to improve the detection of relevant documents, on the basis of terms found in queries [42]. It is based on Singular Value Decomposition (SVD) and projects the document vectors into an approximated subspace, so that cosine similarity can accurately represent semantic similarity.

Given a term-document matrix $X = [x_1, x_2, ..., x_n] \in \mathbb{R}^m$ with rank r, LSI decomposes X

using SVD as follows:

$$X = U\Sigma V^T \tag{3.2}$$

where $\Sigma = diag(\sigma_1, ..., \sigma_r)$ and $\sigma_1 \geqslant \sigma_2 \geqslant ...\sigma_r$ are the singular values of X. $U = [u_1, ..., u_r]$ and u_i is the left singular vector. $V = [v_1, ..., v_r]$ and v_i is the right singular vector. LSI uses the first k vectors in U as the transformation matrix to embed the original documents into a k-dimensional space.

3.5 Clustering Algorithms

3.5.1 K-Means

The K-Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

The K-Means algorithm divides a set of N samples X into k disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids"; however, they are not, in general, points from X, although they live in the same space. The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_j - \mu_i||^2) \tag{3.3}$$

The psuedo-code for K-Means is shown in algorithm 1 [44].

Algorithm 1: K-Means

```
arbitrarily choose an initial k centers C = μ_1, μ_2, ..., μ_k
repeat
    for i ∈ 1, ..., k do
        set the cluster C_i to be the set of points in X that are closer to c_i than they are
          to c_j ∀ j ≠ i
    end
    for i ∈ 1, ..., k do
        set c_i to be the center of mass of all points in C_i : c_i = (1/|C_i|) Σ_{x∈C_i} x
    end
until C no longer changes;
```

3.5.2 Mini-Batch K-Means

The Mini-Batch K-Means is a variant of the K-Means algorithm which uses mini-batches to reduce the computation time, while still attempting to optimize the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of K-Means, Mini-Batch K-Means produces results that are generally only slightly worse than the standard algorithm.

The psuedo-code for Mini-Batch K-Means is listed in algorithm 2 [27].

Algorithm 2: Mini-Batch K-Means

Input: k, mini-batch size b, iterations t, data set X

Initialize each $c \in C$ with an x picked randomly from X
$v \leftarrow 0$
for $i = 1$ **to** t **do**
    $M \leftarrow b$ examples picked randomly from X
    for $x \in M$ **do**
        $d[x] \leftarrow f(C, x)$ `// Cache the center nearest to` x
    end
    for $x \in M$ **do**
        $c \leftarrow d[x]$ `// Get cached center for this` x
        $v[c] \leftarrow v[c] + 1$ `// Update per-center counts`
        $\eta \leftarrow \frac{1}{v[c]}$ `// Get per-center learning rate`
        $c \leftarrow (1-\eta)c + \eta x$ `// Take gradient step`
    end
end

3.5.3 DBSCAN

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to K-Means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, $MinPts$ and ε, which formally define what is meant by dense. Higher $MinPts$ or lower ε indicate higher density necessary to form a cluster.

The pseudo-code for DBSCAN is listed in algorithm 3. DBSCAN delegates some of its logic to another algorithm "Expand Cluster". Its psuedo-code is listed in algorithm 4 [29].

Algorithm 3: DBSCAN

```
1  def DBSCAN(setOfPoints, pps, minPts)
       // setOfPoints is UNCLASSIFIED
2      clusterId ← nextId(NOISE)
3      for i = 1 to setOfPoints.size do
4          point ← setOfPoints.get(i)
5          if point.ClId = UNCLASSIFIED then
6              if expandCluster(setOfPoints, clusterId, eps, minPts) then
7                  clusterId ← nextId(clusterId)
8              end
9          end
10     end
11 end
```

Algorithm 4: Expand Cluster

```
1  def expandCluster(setOfPoints, point, clId, eps, minPts) : Boolean
2      seeds ← setOfPoints.regionQuery(point, eps)
3      if seeds.size < minPts then   // no core point
4          setOfPoint.changeClId (point, NOISE)
5          return False;
6      else  // all points in seeds are density-reachable from point
7          setOfpoints.changeClIds(seeds, clId)
8          seeds.delete(point)
9          while seeds ≠ Empty do
10             currentP ← seeds.first()
11             result ← setofPoints.regionQuery(currentP, eps)
12             if result.size ⩾ minPts then
13                 for i = 1 to result.size do
14                     resultP ← result.get(i)
15                     if resultP.clId ∈ (UNCLASSIFIED, NOISE) then
16                         if resultP.clId = UNCLASSIFIED then
17                             seeds.append(resultP)
18                         end
19                         setOfPoints.changeClId(resultP, clId)
20                     end
21                 end
22             end
23             seeds.delete(currentP)
24         end
25         return True
26     end
27 end
```

3.6 Performance Evaluation Parameters

For the purposes of the following discussion, except for silhouette coefficient, a data set comprising N data points, and two partitions of these, a set of classes, $C = \{c_i | i = 1, ..., n\}$ and a set of clusters, $K = \{k_i | i = 1, ..., m\}$ has been assumed. Let A be the contingency table produced by the clustering algorithm representing the clustering solution, such that $A = \{a_{ij}\}$ where a_{ij} is the number of data points that are members of class c_i and elements of cluster k_j.

3.6.1 Homogeneity

The result of a clustering operation satisfies homogeneity if each of the clusters contain data points from a single class only. The determination of how close a given clustering is to this ideal is done by examining the conditional entropy of the class distribution given the proposed clustering. In a perfectly homogeneous case, $H(C|K) = 0$. However this in not the case in almost all situations. Usually, the size of this value, in bits, is dependent on the size of the dataset and the distribution of class sizes. Hence, instead of taking the raw conditional entropy, this value is normalized by the maximum reduction in entropy the clustering information could provide, specifically, $H(C)$.

$H(C|K) = H(C)$ and is maximal when the clustering provides no new information. $H(C|K) = 0$ when each cluster contains only members of a single class and the clustering is perfectly homogeneous. In this degenerate case ($H(C|K) = 0$), when there is only a single class, homogeneity is defined as 1. So, adhering to the convention of 1 being desirable and 0 undesirable, homogeneity is defined as [45]:

$$h = \begin{cases} 1 & \text{if } H(C, K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases} \tag{3.4}$$

3.6.2 Completeness

Completeness is a metric symmetrical to homogeneity. The result of a clustering operation satisfies completeness if all the data points that are members of a given class are elements of the same cluster. In a perfectly complete clustering solution, distributions of cluster assignments within each class will be completely skewed to a single cluster. This degree of skew can be evaluated by calculating the conditional entropy of the proposed cluster distribution given the class of component data points, $H(K|C)$. In the perfectly complete case, $H(K|C) = 0$ and in the worst case scenario, each class is represented by every cluster with a distribution equal to

the distribution of cluster sizes, i.e., $H(K|C) = H(K)$ and is maximal. In the degenerate case where $H(K) = 0$, when there is a single cluster, completeness is defined as 1. So, similar to homogeneity, the full definition of completeness is [45]:

$$c = \begin{cases} 1 & \text{if } H(K|C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases} \tag{3.5}$$

where,

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{a_{ck}}{N}\, log\left(\frac{a_{ck}}{\sum_{k=1}^{|K|} a_{ck}}\right)$$
$$H(K) = -\sum_{k=1}^{|K|} \frac{\sum_{c=1}^{|C|} a_{ck}}{n}\, log\left(\frac{\sum_{c=1}^{|C|} a_{ck}}{n}\right)$$

3.6.3 V-Measure

V-measure is the weighted harmonic mean of homogeneity and completeness,

$$V_\beta = \frac{(1+\beta)hc}{(\beta h) + c} \tag{3.6}$$

If $\beta > 1$ completeness is weighted more strongly in the calculation. Conversely, if $\beta < 1$, homogeneity is weighted more strongly. There is no reason to believe that the data used in this dissertation is skewed towards homogeneity or completeness, so β has been set to 1. Therefore, the eq. (3.6) simplifies to:

$$V = \frac{2hc}{h + c} \tag{3.7}$$

The computations of homogeneity, completeness and V-measure are completely independent of the number of classes, the number of clusters, the size of the data set and the clustering algorithm used. Thus these measures can be applied to any clustering analysis irrespective of number of data points (n-invariance), number of classes or number of clusters [45].

3.6.4 Silhouette Coefficient

Silhouette coefficient provides a graphical display for partitioning techniques. Each cluster is represented by a so-called silhouette, which is based on the comparison of its tightness and separation. This silhouette shows which objects lie well within their cluster, and which ones

are merely somewhere in between clusters. The entire clustering is displayed by combining the silhouettes into a single plot, allowing an appreciation of the relative quality of the clusters and an overview of the data configuration. The average silhouette width provides an evaluation of clustering validity, and might be used to select an "appropriate" number of clusters.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to $+1$, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

Given a data point i and clusters k let $a(i)$ be the average distance between i and all other data within the same cluster. $a(i)$ can then be interpreted as a measure of how well i is assigned to its cluster (smaller values are better). The average dissimilarity of point i a cluster c can then be defined as the average of the distance from i to all points in c.

Let $b(i)$ be the lowest average distance of i to all points in any other cluster, of which i is not a member. The cluster with this lowest average dissimilarity is defined as the "neighbouring cluster" of i as it is the next best fit cluster for point i. Silhouette coefficient of point i can now be defined as [46]:

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, & \text{if } a(i) > b(i) \end{cases} \tag{3.8}$$

which can be rewritten concisely as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.9}$$

It is also possible to consider the overall average silhouette width for the entire plot, which is simply the average of the $s(i)$ for all objects i in the whole data set. In general, each value of k will yield a different overall average silhouette width $s(k)$. One way to choose k "appropriately" is to select that value of k for which $s(k)$ is a large as possible [46, 47].

Theoretically, no cluster validity index has a clear advantage over others in every case. However, the silhouette coefficient has performed well over other indices in many comparative experiments [47–50].

CHAPTER 4

IMPLEMENTATION

4.1 Programming Language and Frameworks

All the algorithms have been implemented using Python (version 3.6.3) language, PyCharm Community Edition Integrated Development Environment (IDE) and Jupyter plugin for Python. For efficiency and ease of implementation, APIs from Scikit-Learn [51] machine learning library and Pandas library were extensively used in this research. The study is carried out in a MacBook Pro (Retina, Mid 2012) with 2.3 GHz Intel Core i7 processor and 8 GB (1600 MHz DDR3) RAM.

4.2 Preprocessing

4.2.1 Parser

The **Parser** implemented in this section is intended to remove invalid characters like punctuation marks, Hyper Text Markup Language (HTML) characters and any devanagari characters or symbols that do not add any meaningful contributions to the clustering process. It replaces any characters not specified in a collection of whitelist characters and symbols with spaces and makes the dataset ready for the **Tokenizer** algorithm (section 4.2.2). The pseudo-code for **Parser** is listed in algorithm 5.

4.2.2 Tokenizer

The **Tokenizer** splits the output given by **Parser** algorithm using spaces as delimiters. Its output is a list of tokens that serves as input for both stop-words removal procedure (appendix B.5) as well as Stemmer algorithm (section 4.2.4). The pseudo-code for **Tokenizer** is listed in algorithm 6.

Algorithm 5: Parse Valid Characters

```
1  def parseValidCharacters(value)
2      value.remove(ZWNJ) // zero width non joiner
3      value.remove(ZWJ) // zero width joiner
4      whitelist ← ''
5      whitelist.append('अआइईउऊऋएऐओऔ')
6      whitelist.append('कखगघङचछजझञटठडढणतथदधनपफबभमयरलवशषसह')
7      whitelist.append('ा ि ी ु ू ृ े ै ो ौ ं ः ँ ्')
8      for character ∈ value do
9          if character ∉ whitelist then
10             replace character with SPACE in value
11         end
12     end
13     return value
14 end
```

Algorithm 6: Tokenize

```
1  def tokenize(value)
2      value ← parseValidCharacters(value)
3      tokens ← value.split(SPACE)
4      return tokens
5  end
```

4.2.3 Stop-Word Remover

The **Stop-Word Remover** removes stop-words from the output of **Tokenizer** algorithm. The list of stop-words are listed in appendix A.3. The pseudo-code for **Stop-Word Remover** is listed in algorithm 7.

Algorithm 7: Remove Stop-Words

```
1  def removeStopWords(tokens)
2      stopWords ← [ ]
3      stopWords ← readFromFile(STOP_WORDS_FILE)
4      for token ∈ tokens do
5          if token.length ⩽ 2 or token ∈ stopWords then
6              tokens.remove(token)
7          end
8      end
9      return tokens
10 end
```

4.2.4 Stemmer

The following algorithm represents a basic rule-based **Stemmer** that removes inflectional suffixes. The rules are listed in appendix A.4.

For each token in the dataset, the **Stemmer** first reads the rules from its data file and orders them by descending order of length. Since a token may contain multiple suffixes, and some suffixes may be proper subset of a longer suffix, applying corresponding rules randomly or rules sorted by ascending order of length may result in incorrect removal of suffixes in many cases. Sorting rules by descending order of length takes care of this problem. The pseudo-code for **Stemmer** is listed in algorithm 8.

Algorithm 8: Find Root Word

```
1 def findRootWord(word)
2     rules ← [ ]
3     rules ← readFromFile(RULE_FILE)
4     rules.descendingSort(rule → rule.length)  // sort by rule length
5     for suffix ∈ rules do // rule = suffix
6         word.remove(suffix)
7     end
8     return word
9 end
```

4.2.5 Combined Preprocessor

The **Combined Preprocessor** applies all the algorithms mentioned above (sections 4.2.1 to 4.2.4) to the dataset and saves the result to an output Comma-Separated Values (CSV) file. The resulting file can be used directly by clustering algorithms used in this research. The pseudo-code for **Combined Preprocessor** is listed in algorithm 9.

Algorithm 9: Perform Preprocessing

```
1 def performPreprocessing()
2     data ← [ ]
3     data ← readFromFile(INPUT_FILE)
4     tokens ← tokenize(data)
5     tokens ← removeStopWords(tokens)
6     tokens ← findRootWord(tokens)
7     saveToFile(OUTPUT_FILE, tokens)
8 end
```

4.3 Parameter Tuning

Before using the clustering algorithms, it is necessary to find and use optimal values for the parameters expected by them.

For DBSCAN there are two parameters to be optimized, ε and $MinPts$. Using the code in appendices B.6 and B.7, the optimal value for the parameters was determined to be 0.55 and 10 respectively. The pseudo-code for tuning ε is listed in algorithm 10 and for $Minpts$ is listed in algorithm 11.

For both K-Means and Mini-Batch K-Means, number of clusters, k, was directly determined from sample data using the Pandas library for Python (appendices B.9 and B.10). Separate parameter tuning code was unnecessary for the purpose of this study, due to way k was determined.

Algorithm 10: Epsilon Tuner

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 minPts ← 10
 3 for size ∈ dataSizes do
 4     data ← readFromFile(DATA_FILE, size)
 5     tokens ← data.tokens
 6     X ← tfidf(tokens)
 7     X ← lsi(X)
 8     labels ← data.target
 9     for ε ∈ [0.1 : 1.0 : 0.01] do
10         model ← dbscanModel(ε, minPts)
11         model.fit(X)
12         homogeneity ← homogeneityScore(labels, model.labels)
13         completeness ← completenessScore(labels, model.labels)
14         vMeasure ← vMeasureScore(labels, model.labels)
15         print(size, ε, homogeneity, completeness, vMeasure)
16     end
17 end
```

4.4 Cluster Analysis

Algorithms 12 to 15 are used to study the performance of K-Means, Mini-Batch K-Means and DBSCAN. Each of these algorithms reads tokens from the data file created by **Combined Preprocessor** (section 4.2.5), applies TFIDF or TFIDF + LSI combination on the tokens and prints out respective analysis measures (i.e., time or performance measures) for different data sizes.

Algorithm 11: MinPts Tuner

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 ε ← 0.55
 3 for size ∈ dataSizes do
 4     data ← readFromFile(DATA_FILE, size)
 5     tokens ← data.tokens
 6     X ← tfidf(tokens)
 7     X ← lsi(X)
 8     labels ← data.target
 9     for minPts ∈ [5 : 100 : 1] do
10         model = dbscanModel(ε, minPts)
11         model.fit(X)
12         homogeneity ← homogeneityScore(labels, model.labels)
13         completeness ← completenessScore(labels, model.labels)
14         vMeasure ← vMeasureScore(labels, model.labels)
15         print(size, minPts, homogeneity, completeness, vMeasure)
16     end
17 end
```

Algorithm 12: Performance Analysis with TFIDF

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 ε ← 0.55
 3 minPts ← 10
 4 for size ∈ dataSizes do
 5     data ← readFromFile(DATA_FILE, size)
 6     tokens ← data.tokens
 7     k ← data.category.unique().length
 8     X ← tfidf(tokens)
 9     models ← [kmeansModel(k, X), miniBatchKMeansModel(k, X), dbscanModel(ε,
         minPts]
10     labels ← data.target
11     for model ∈ models do
12         model.fit(X)
13         homogeneity ← homogeneityScore(labels, model.labels)
14         completeness ← completenessScore(labels, model.labels)
15         vMeasure ← vMeasureScore(labels, model.labels)
16         silhouette ← silhouetteScore(labels, model.labels)
17         print(size, homogeneity, completeness, vMeasure, silhouette)
18     end
19 end
```

Algorithm 13: Performance Analysis with TFIDF and LSI

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 ε ← 0.55
 3 minPts ← 10
 4 for size ∈ dataSizes do
 5     data ← readFromFile(DATA_FILE, size)
 6     tokens ← data.tokens
 7     k ← data.category.unique().length
 8     X ← tfidf(tokens)
 9     X ← lsi(X)
10     models ← [kmeansModel(k, X), miniBatchKMeansModel(k, X), dbscanModel(ε,
         minPts]
11     labels ← data.target
12     for model ∈ models do
13         model.fit(X)
14         homogeneity ← homogeneityScore(labels, model.labels)
15         completeness ← completenessScore(labels, model.labels)
16         vMeasure ← vMeasureScore(labels, model.labels)
17         silhouette ← silhouetteScore(labels, model.labels)
18         print(size, homogeneity, completeness, vMeasure, silhouette)
19     end
20 end
```

Algorithm 14: Time Analysis with TFIDF

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 ε ← 0.55
 3 minPts ← 10
 4 for size ∈ dataSizes do
 5     data ← readFromFile(DATA_FILE, size)
 6     tokens ← data.tokens
 7     k ← data.category.unique().length
 8     X ← tfidf(tokens)
 9     models ← [kmeansModel(k, X), miniBatchKmeansModel(k, X), dbscanModel(ε,
         minPts]
10     labels ← data.target
11     for model ∈ models do
12         startTime ← time()
13         model.fit(X)
14         duration ← time() - startTime
15         print(size, duration)
16     end
17 end
```

Algorithm 15: Time Analysis with TFIDF and LSI

```
   // uses tfidf, lsi, dbscan, homogeneity, completeness and
   // v-measure from scikit-learn
 1 dataSizes ← [2 000 : 10 000 : 1 000]  // 2 000, 3 000, ..., 10 000
 2 ε ← 0.55
 3 minPts ← 10
 4 for size ∈ dataSizes do
 5     data ← readFromFile(DATA_FILE, size)
 6     tokens ← data.tokens
 7     k ← data.category.unique().length
 8     X ← tfidf(tokens)
 9     X ← lsi(X)
10     models ← [kmeansModel(k, X), miniBatchKmeansModel(k, X), dbscanModel(ε,
         minPts]
11     labels ← data.target
12     for model ∈ models do
13         startTime ← time()
14         model.fit(X)
15         duration ← time() - startTime
16         print(size, duration)
17     end
18 end
```

CHAPTER 5

RESULT ANALYSIS AND DISCUSSIONS

The dataset mentioned in section 3.1 was clustered using three algorithms DBSCAN, K-Means and Mini-Batch K-Means with various sample data sizes and their performance was studied using four measures: Homogeneity, Completeness, V-Measure and Silhouette Coefficient. Differences in cluster quality when using TFIDF and combination of TFIDF + LSI for text representation was also studied. Similarly, the time taken by the algorithms was also studied using both text representation schemes.

The sample data size or number of documents used in the experiments are denoted by "**size**" and the run-time in seconds are denoted by "**time**" in the tables and figures in the following sections.

5.1 Performance Analysis with TFIDF

Table 5.1 lists the result of clustering the Nepali dataset using different data sizes. Text representation was done using TFIDF only.

Figure 5.1 shows the plots of the clustering algorithms v/s performance measures for table 5.1. Mini-Batch K-Means has the best performance and DBSCAN has the worst. The performance of K-Means is nearly identical to Mini-Batch version for higher sample sizes.

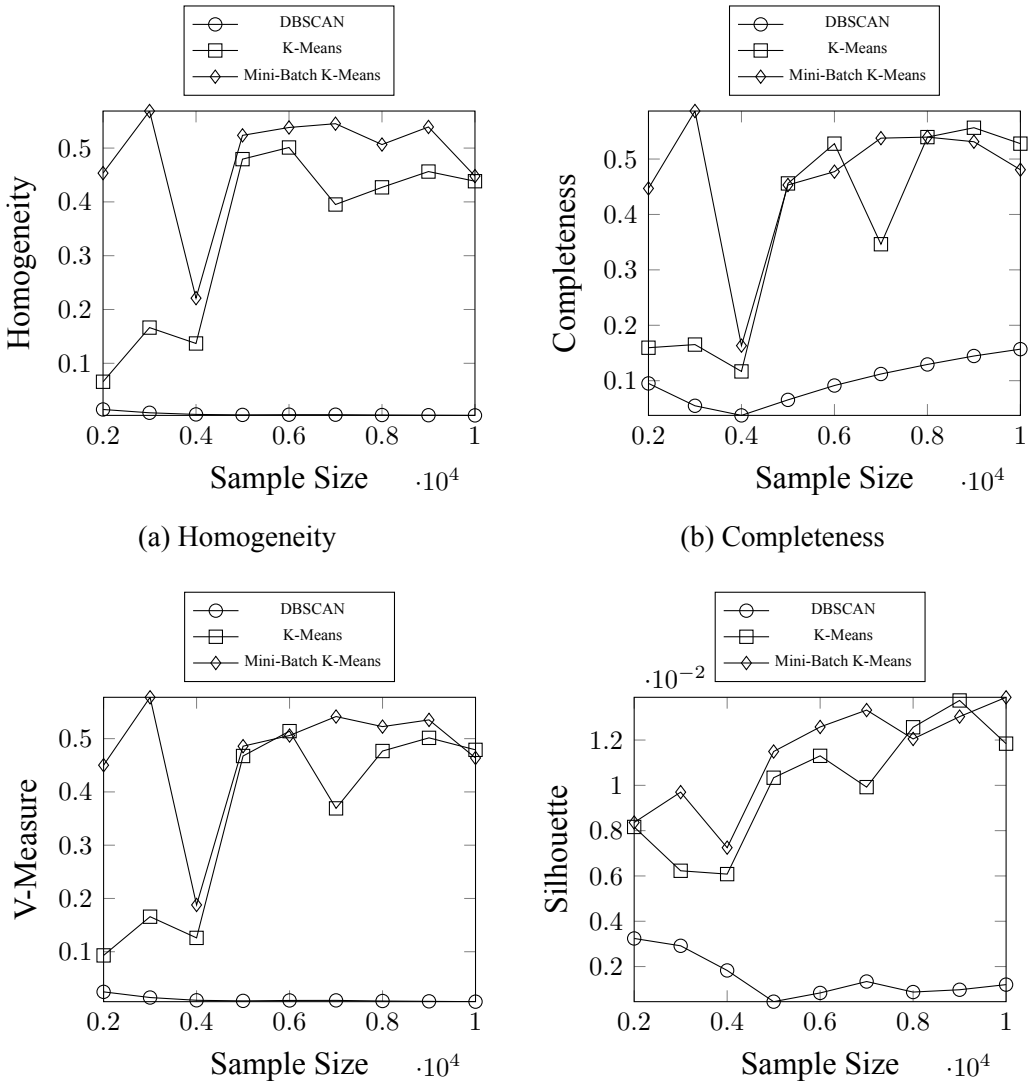
5.2 Performance Analysis with TFIDF and LSI

Table 5.2 shows the results of clustering the dataset using different data sizes similar to Table 5.1, with one difference – text representation used for this was TFIDF + LSI. The major advantage of using LSI is its ability to transform the text representation into a lower dimensional form. This improves the result of clustering algorithms.

Figure 5.2 shows the plots of the clustering algorithms v/s performance measures for table 5.2. DBSCAN performs much better with the application of LSI. However, it is still behind the other two in Completeness, Homogeneity and V-Measure. The performance of K-Means and Mini-Batch also improves a little, except in Silhouette score where the improvement ratio is comparatively better. K-Means performs best from the perspective of Completeness, V-Measure and Silhouette Coefficient whereas Mini-Batch K-Means scores better in Homogeneity.

Algorithm	Size	Homogeneity	Completeness	V-Measure	Silhouette
	2,000	0.014162557	0.095062318	0.024652360	0.003240279
	3,000	0.008039397	0.054708011	0.014018728	0.002916194
	4,000	0.004998860	0.037372858	0.008818225	0.001828447
	5,000	0.004111333	0.065374395	0.007736147	0.000450257
DBSCAN	6,000	0.004478619	0.091272191	0.008538274	0.000835649
	7,000	0.004490954	0.112061915	0.008635821	0.001339375
	8,000	0.003865849	0.129309259	0.007507260	0.000874452
	9,000	0.003525922	0.144541874	0.006883919	0.000976385
	10,000	**0.003190042**	**0.156913349**	**0.006252962**	**0.001202579**
	2,000	0.065725926	0.159497783	0.093090905	0.008163300
	3,000	0.166349805	0.165164577	0.165755072	0.006228649
	4,000	0.136843898	0.116724418	0.125985964	0.006081243
	5,000	0.479450862	0.455990192	0.467426332	0.010336318
K-Means	6,000	0.501097067	0.527766597	0.514086177	0.011302530
	7,000	0.395121842	0.346275902	0.369089799	0.009921182
	8,000	0.427014022	0.539584375	0.476744209	0.012563696
	9,000	0.456360194	0.556505987	0.501482200	0.013747476
	10,000	**0.438415457**	**0.527807275**	**0.478976244**	**0.011840634**
	2,000	0.453432557	0.446961315	0.450173681	0.008359614
	3,000	0.568853662	0.586650900	0.577615223	0.009700666
	4,000	0.221093048	0.163240458	0.187812563	0.007248560
	5,000	0.523651710	0.453139567	0.485850590	0.011493876
Mini-Batch K-Means	6,000	0.538269372	0.477199390	0.505898016	0.012577101
	7,000	0.545322140	0.537703477	0.541486011	0.013322621
	8,000	0.506561918	0.539595857	0.522557340	0.012047895
	9,000	0.539257283	0.531231006	0.535214055	0.013033515
	10,000	**0.447876592**	**0.480701863**	**0.463709041**	**0.013886573**

Table 5.1: Performance Analysis with TFIDF



(a) Homogeneity

(b) Completeness

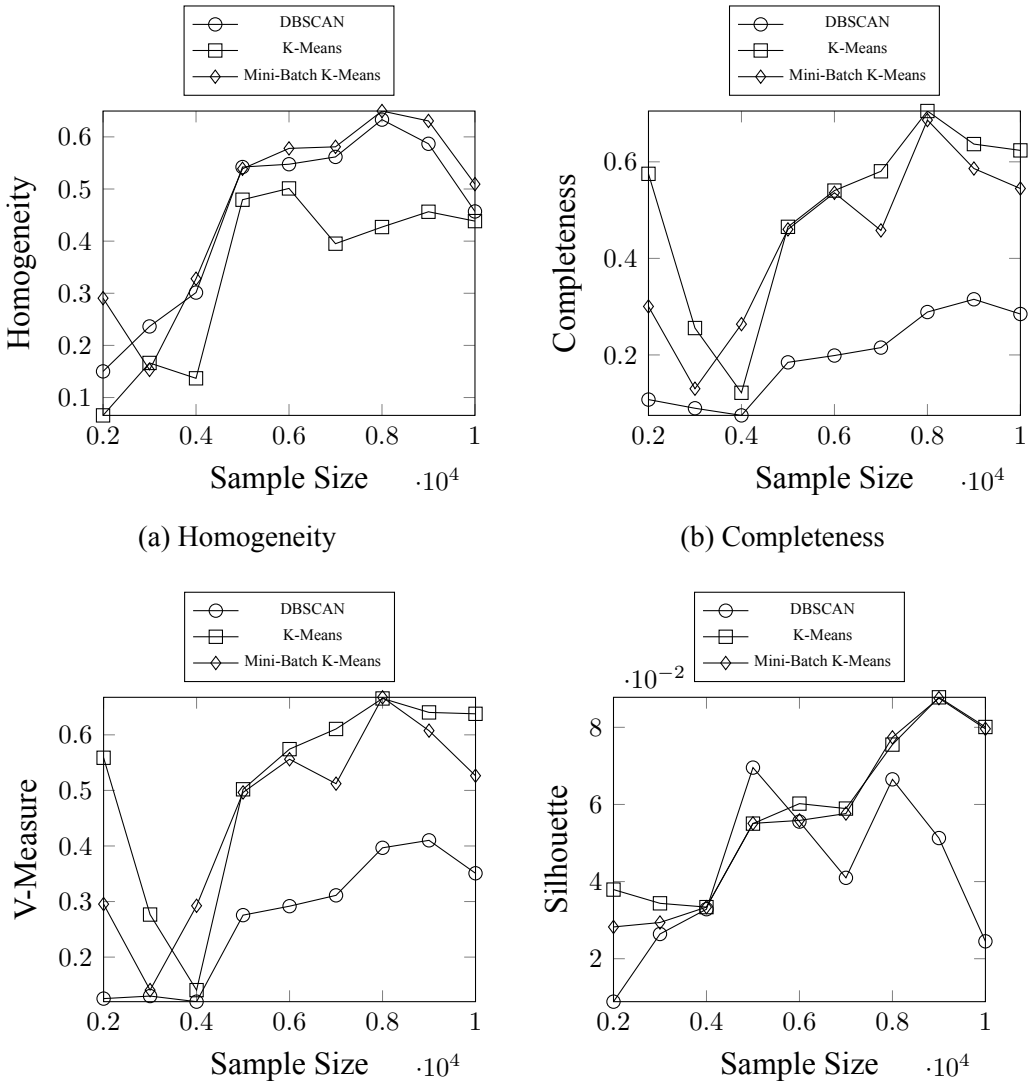
(c) V-Measure

(d) Silhouette

Figure 5.1: Performance Analysis with TFIDF

Algorithm	Size	Homogeneity	Completeness	V-Measure	Silhouette
	2,000	0.150149442	0.107527804	0.125313663	0.008862449
	3,000	0.236370347	0.089563840	0.129904974	0.026407684
	4,000	0.301301370	0.074772534	0.119811913	0.032838504
	5,000	0.542138617	0.184681096	0.275509187	0.069554143
DBSCAN	6,000	0.547528443	0.198763208	0.291651419	0.055555315
	7,000	0.561548739	0.215189415	0.311145639	0.040973935
	8,000	0.633093584	0.288844825	0.396698529	0.066511200
	9,000	0.586737996	0.315266314	0.410150425	0.051288355
	10,000	**0.456941973**	**0.284778871**	**0.350879769**	**0.024507214**
	2,000	0.543283235	0.575136726	0.558756374	0.037925249
	3,000	0.301166180	0.255707946	0.276581661	0.034362360
	4,000	0.166439740	0.121641776	0.140557616	0.033377461
	5,000	0.544907224	0.465347943	0.501994870	0.055030756
K-Means	6,000	0.612628729	0.540454683	0.574282939	0.060214767
	7,000	0.644088334	0.580377200	0.610575265	0.058932564
	8,000	0.629940794	0.705226792	0.665461220	0.075520788
	9,000	0.644012688	0.636742599	0.640357009	0.087783905
	10,000	**0.652657490**	**0.623700829**	**0.637850691**	**0.080085281**
	2,000	0.290555770	0.300421680	0.295406373	0.028236413
	3,000	0.153040141	0.130022070	0.140595213	0.029385830
	4,000	0.327868864	0.263777399	0.292351703	0.033375921
	5,000	0.538831489	0.460097958	0.496361917	0.055093065
Mini-Batch K-Means	6,000	0.577887273	0.535832003	0.556065611	0.055856954
	7,000	0.580925741	0.457917074	0.512138721	0.057641862
	8,000	0.649567713	0.686554082	0.667548971	0.077399998
	9,000	0.630569416	0.586108722	0.607526712	0.087561338
	10,000	**0.509335828**	**0.544688453**	**0.526419266**	**0.079607913**

Table 5.2: Performance Analysis with TFIDF and LSI



(a) Homogeneity

(b) Completeness

(c) V-Measure

(d) Silhouette

Figure 5.2: Performance Analysis with TFIDF and LSI

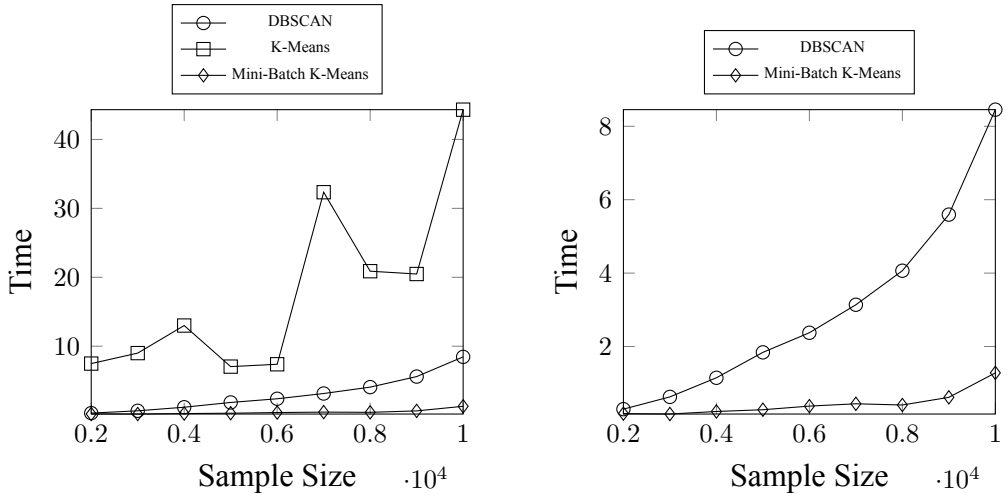
All of the algorithms used in this research use Euclidean distance as distance metric. For high dimensional data, such as that used in this research, this metric tend to become inflated, which is an instance of the so called curse of dimensionality. The experiments for this section use LSI, prior to applying the clustering algorithms to alleviate this problem. This has improved the performance measures as seen from comparing table 5.1 with table 5.2 and fig. 5.1 with fig. 5.2.

5.3 Time Analysis with TFIDF

Table 5.3 lists the time taken by the algorithms using TFIDF. Figure 5.3 shows the corresponding plots. The figure shows that K-Means is the slowest algorithm and Mini-Batch K-Means the fastest. The performance of DBSCAN is similar to Mini-Batch K-Means from fig. 5.3a. However, their difference is completely overshadowed by K-Means, so they are separately plotted (without K-Means) in fig. 5.3b to highlight the differences.

Algorithm	**Size**	**Time**
DBSCAN	2,000	0.294145107
	3,000	0.628072023
	4,000	1.148311853
	5,000	1.841221094
	6,000	2.374995232
	7,000	3.137669325
	8,000	4.065223932
	9,000	5.593692303
	10,000	**8.453108072**
K-Means	2,000	7.479647875
	3,000	8.999480963
	4,000	12.999316931
	5,000	7.039831161
	6,000	7.380022049
	7,000	32.342036009
	8,000	20.883139133
	9,000	20.475090027
	10,000	**44.317399025**
Mini-Batch K-Means	2,000	0.171869040
	3,000	0.160945892
	4,000	0.230267048
	5,000	0.278146982
	6,000	0.376816988
	7,000	0.439945221
	8,000	0.406978130
	9,000	0.615887880
	10,000	**1.282078981**

Table 5.3: Time Analysis with TFIDF



(a) DBSCAN, K-Means, Mini-Batch K-Means

(b) DBSCAN, Mini-Batch K-Means

Figure 5.3: Time Analysis with TFIDF

5.4 Time Analysis with TFIDF and LSI

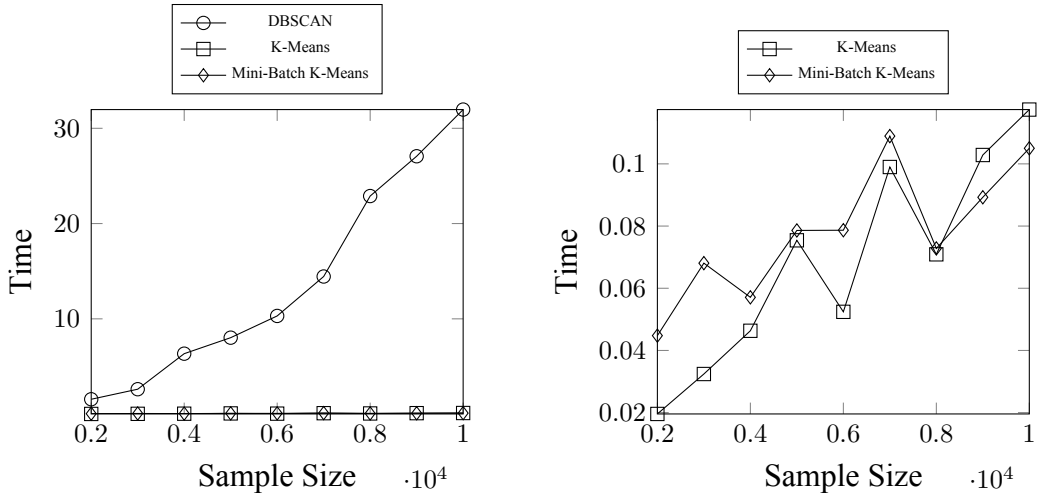
Table 5.4 lists the time taken by the algorithms using TFIDF + LSI. Figure 5.4 shows the corresponding plots. With the application of LSI, the performance of K-Means improves significantly. Mini-Batch K-Means improves only slightly. However, strangely, the time performance for DBSCAN degrades drastically with this.

Time plot for DBSCAN overshadows the other two algorithms in (fig. 5.4a). So, they have been replotted without DBSCAN for better comparison (fig. 5.4b). The performances of K-Means and Mini-Batch K-Means are very similar to each other. K-Means barely outperforms the Mini-Batch version for low sample sizes whereas the Mini-Batch version slightly outperforms the former in case of high sample sizes.

Similar to the improvement in performance measures, the use of LSI for dimensionality reduction, prior to using clustering algorithms, has improved the run-times of all all the algorithms, except DBSCAN. The anomalous behavior shown by DBSCAN is due to the large memory requirement for running the algorithm. Since the computer used for running the experiment only contained 8GB Random Access Memory (RAM), the python runtime used secondary storage (in this case, a 256 Giga-Byte (GB) Solid State Drive (SSD)) as virtual memory, which led to the degradation in run-time behavior for this algorithm.

Algorithm	Size	Time
DBSCAN	2,000	1.561387062
	3,000	2.615876198
	4,000	6.345207691
	5,000	8.024958372
	6,000	10.305550814
	7,000	14.444278002
	8,000	22.888608932
	9,000	27.065243006
	10,000	**31.963739157**
K-Means	2,000	0.019575119
	3,000	0.032418013
	4,000	0.046323776
	5,000	0.075416088
	6,000	0.052417755
	7,000	0.098990917
	8,000	0.070907116
	9,000	0.102853775
	10,000	**0.117464066**
Mini-Batch K-Means	2,000	0.044759035
	3,000	0.068073034
	4,000	0.057080030
	5,000	0.078593016
	6,000	0.078658104
	7,000	0.108935833
	8,000	0.072808266
	9,000	0.089255810
	10,000	**0.105010271**

Table 5.4: Time Analysis with TFIDF and LSI



(a) DBSCAN, K-Means, Mini-Batch K-Means

(b) K-Means, Mini-Batch K-Means

Figure 5.4: Time Analysis with TFIDF and LSI

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Separating large number of documents into similar and meaningful clusters using computers has a wide range of applications. Extensive study has been done in this field for English language but for Nepali language, study in clustering documents, including some fields in preprocessing, is still lacking.

The algorithms chosen for this study are generic and are applicable to any language, but there is no guarantee that the performance measures and run-times will be the same for each and every language. This is due to the nature and complexities of individual languages, which directly affects not only the preprocessing steps but also the dimensionality of the documents to be clustered. This study is an attempt to contribute some research in this area.

The summary of cluster quality analysis after applying K-Means, Mini-Batch K-Means and DBSCAN is listed in table 6.1. Mini-Batch K-Means performs better than remaining two algorithms when using TFIDF only in text representation whereas K-Means performs better when using LSI. DBSCAN performs worst in almost all cases.

	TFIDF	**TFIDF + LSI**
Homogeneity	Mini-Batch K-Means	Mini-Batch K-Means
Completeness	Mini-Batch K-Means	K-Means
V-Measure	Mini-Batch K-Means	K-Means
Silhouette	Mini-Batch K-Means	K-Means

(a) Best

	TFIDF	**TFIDF + LSI**
Homogeneity	DBSCAN	K-Means
Completeness	DBSCAN	DBSCAN
V-Measure	DBSCAN	DBSCAN
Silhouette	DBSCAN	DBSCAN

(b) Worst

Table 6.1: Performance Measures Summary

Similarly, the summary of completion times for the algorithms is listed in table 6.2. Mini-Batch K-Means is the best algorithm in all cases whereas K-Means is the worst when using TFIDF only and DBSCAN is the worst when using TFIDF + LSI combination.

All of the algorithms used in this research use Euclidean distance as distance metric. For high dimensional data, such as that used in this research, this metric tend to become inflated, which is an instance of the so called curse of dimensionality. When using TFIDF with LSI, prior to applying the clustering algorithms, the performance measures and run-times of all the algorithms are improved, except the run-time of DBSCAN. The anomalous behavior shown by DBSCAN is due to the large memory requirement for running the algorithm and use of virtual memory when main memory is insufficient.

	TFIDF	**TFIDF + LSI**
Best	Mini-Batch K-Means	Mini-Batch K-Means
Worst	K-Means	DBSCAN

Table 6.2: Time Summary

6.2 Future Work

This dissertation limits its study to a maximum of 10,000 data samples. This is mainly due to the fact that DBSCAN does not behave nicely when sample sizes are very large (see table 5.2). It consumes too much memory in such cases, due to $O(n^2)$ space complexity, and much of processing time is spent using virtual memory instead of doing useful calculations. Future studies can focus on how to remove this bottleneck. Similarly, it is also possible to compare the performance of other algorithms with large data samples.

REFERENCES

[1] A. Huang, "Similarity measures for text document clustering," *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand*, pp. 49–56, 2008.

[2] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, ISBN: 978-1-55860-901-3, 1-55860-901-6.

[3] C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," pp. 77–128, Aug. 2012.

[4] S. Sarkar, A. Roy, and B S. Purkayastha, "A comparative analysis of particle swarm optimization and k-means algorithm for text clustering using nepali wordnet," *International Journal on Natural Language Computing*, vol. 3, pp. 83–92, Jun. 2014.

[5] A. Neupane, "Development of nepali character database for character recognition based on clustering," *International Journal of Computer Applications*, vol. 107, no. 11, pp. 42–46, Dec. 2014.

[6] C. Sitaula, "Semantic text clustering using enhanced vector space model using nepali language," *International Journal on Natural Language Computing (IJNLC)*, vol. 3, no. 3, pp. 83–92, Jun. 2014.

[7] D. Wilks, *Statistical Methods in the Atmospheric Sciences*, 3rd, ser. International geophysics series. Academic Press, 2011, ISBN: 9780127519661. DOI: 10.1016/B978-0-12-385022-5.00015-4.

[8] J. L. Solka, "Text data mining: Theory and methods," *Statistics Surveys*, vol. 2, pp. 94–112, 2008. DOI: 10.1214/07-SS016.

[9] M. Berry and M. Browne, *Understanding Search Engines*, 2nd. Philadelphia: Society for Industrial and Applied Mathematics, 2005. DOI: 10.1137/1.9780898718164.

[10] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional spaces," in *Proceedings of the 8th International Conference on Database Theory*, ser. ICDT '01, Berlin, Heidelberg: Springer-Verlag, 2001, pp. 420–434, ISBN: 3-540-41456-8. [Online]. Available: http://dl.acm.org/citation.cfm?id=645504.656414.

[11] S. Äyrämö and T. Kärkkäinen, "Introduction to partitioning-based clustering methods with a robust example," in *Reports of the Department of Mathematical Information Technology Series C. Software and Computational Engineering*, University of Jyväskylä, 2006.

[12] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining*, Jun. 2000.

[13] W. Wang, J. Yang, and R. R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 186–195, ISBN: 1-55860-470-7.

[14] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: A wavelet-based clustering approach for spatial data in very large databases," *The VLDB Journal*, vol. 8, no. 3-4, pp. 289–304, Feb. 2000, ISSN: 1066-8888. DOI: 10.1007/s007780050009.

[15] G. Gan, C. Ma, and J. Wu, "Model-based clustering algorithms," in *Data Clustering: Theory, Algorithms, and Applications*, ser. ASA-SIAM Series on Statistics and Applied Mathematics. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics, 2007, pp. 227–242. DOI: 10.1137/1.9780898718348.ch14.

[16] K. D. Bailey, *Typologies and Taxonomies: An Introduction to Classification Techniques*, M. S. Lewis-Beck, Ed. Thousand Oaks, CA: Sage Publications, 1994. DOI: http://dx.doi.org/10.4135/9781412986397.

[17] R. and R. Sokal, "Numerical taxonomy. the principles and practice of numerical classification," vol. 12, no. 5, pp. 190–199, Jun. 1963.

[18] H. P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 1, 1:1–1:58, Mar. 2009, ISSN: 1556-4681. DOI: 10.1145/1497577.1497578.

[19] C. J. V. Rijsbergen, *Information Retrieval*, 2nd. Newton, MA, USA: Butterworth-Heinemann, 1979, ISBN: 0408709294.

[20] C. Buckley and A. F. Lewit, "Optimization of inverted vector searches," in *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '85, New York, NY, USA: ACM, 1985, pp. 97–110, ISBN: 0-89791-159-8. DOI: 10.1145/253495.253515.

[21] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections," in *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '92, Copenhagen, Denmark: ACM, 1992, pp. 318–329, ISBN: 0-89791-523-2. DOI: 10.1145/133160.133214.

[22] O. Zamir, O. Etzioni, O. Madani, and R. M. Karp, "Fast and intuitive clustering of web documents," in *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, 1997, pp. 287–290.

[23] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[24] H. H. Bock, "Origins and extensions of the k-means algorithm in cluster analysis," *Electronic Journ@l for History of Probability and Statistics*, vol. 4, no. 2, Dec. 2008.

[25] H. Steinhaus, "Sur la division des corps matériels en parties.," French, *Bull. Acad. Pol. Sci., Cl. III*, vol. 4, pp. 801–804, 1957, ISSN: 0001-4095.

[26] E. Forgy, "Cluster analysis of multivariate data: Efficiency versus interpretability of classification," *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[27] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10, New York, NY, USA: ACM, 2010, pp. 1177–1178, ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772862.

[28] R. T. Ng and J. Han, "Clarans: A method for clustering objects for spatial data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1003–1016, Sep. 2002, ISSN: 1041-4347. DOI: 10.1109/TKDE.2002.1033770.

[29] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[30] B. K. Bal and P. Shrestha, "A morphological analyzer and a stemmer for Nepali," *PAN Localization, Working Papers*, vol. 2007, pp. 324–31, 2004.

[31] *Language technology kendra | projects*. [Online]. Available: http://www.ltk.org.np/projects.php (visited on 04/09/2018).

[32] T. B. Shahi, T. N. Dhamala, and B. Balami, "Support vector machines based part of speech tagging for Nepali text," *International Journal of Computer Applications*, vol. 70, no. 24, pp. 38–42, May 2013.

[33] A. Paul, B. Syam Purkayastha, and S. Sarkar, "Hidden markov model based part of speech tagging for nepali language," pp. 149–156, Sep. 2015.

[34] S. Bam and T. Shahi, "Named entity recognition for nepali text using support vector machines," *Intelligent Information Management*, vol. 6, pp. 21–29, 2014.

[35] A. Dey, A. Paul, and B. S. Purkayastha, "Named entity recognition for nepali language: A semi hybrid approach," *International Journal of Engineering and Innovative Technology (IJEIT)*, vol. 3, no. 8, pp. 21–25, Feb. 2014.

[36] T. B. Shahi and A. K. Pant, "Nepali news classification using naïve bayes, support vector machines and neural networks," in *2018 International Conference on Communication information and Computing Technology (ICCICT)*, Feb. 2018, pp. 1–5. DOI: 10.1109/ICCICT.2018.8325883.

[37] (2018). The unicode standard 10.0, [Online]. Available: https://www.unicode.org/charts/PDF/U0900.pdf (visited on 04/10/2018).

[38] N. F. S. Committee. (2018). Nepali font standards (white paper v2), [Online]. Available: https://www.unicode.org/L2/L1999/99235.pdf (visited on 04/10/2018).

[39] B. K. Pokharel, B. Tripathi, K. P. Parajuli, G. Sharma, and H. Bhattarai, Eds., *Nepali Brihat Shabdakosh*, 7th. Kamaladi, Kathmandu, Nepal: Nepal Pragya Pratishthan, 2011.

[40] N. Haghtalab, "Clustering in the presence of noise," Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2013. [Online]. Available: https://uwspace.uwaterloo.ca/bitstream/handle/10012/7742/Haghtalab_Nika.pdf.

[41] I. Shrestha, S. S. Dhakal, and M. Kadariya, "A comparative study of stemming algorithms for nepali language," *National Student's Conference on Information Technology (NaSCoIT)*, 2016.

[42] W. Zhang, T. Yoshida, and X. Tang, "A comparative study of tf*idf, lsi and multi-words for text classification," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758–2765, Mar. 2011, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2010.08.066.

[43] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[44] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035, ISBN: 978-0-898716-24-5.

[45] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Association for Computational Linguistics, Jan. 2007, pp. 410–420.

[46] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987, ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7.

[47] R. C. de Amorim and C. Hennig, "Recovering the number of clusters in data sets with noise features using feature rescaling factors," *Information Sciences*, vol. 324, pp. 126–145, 2015, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2015.06.039.

[48] J. C. Bezdek and N. R. Pal, "Some new indexes of cluster validity," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 3, pp. 301–315, Jun. 1998, ISSN: 1083-4419. DOI: 10.1109/3477.678624.

[49] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243–256, 2013, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2012.07.021.

[50] K. S.P.M. J. van der Laan, "A method to identify significant clusters in gene expression data," in *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA: Inpress, 2002, pp. 318–325.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

APPENDIX A

DATASET DESCRIPTION

A.1 Number of Clusters

Sample Size	Clusters
2,000	2
3,000	2
4,000	2
5,000	3
6,000	3
7,000	4
8,000	4
9,000	5
10,000	5

A.2 Valid Subset of Nepali Character Set (White-List)

अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ

(a) Vowels

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ
ट	ठ	ड	ढ	ण	त	थ	द	ध	न
प	फ	ब	भ	म	य	र	ल	व	
श	ष	स	ह						

(b) Consonants

ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ	ं	ः

(c) Dependent Vowel Signs

ँ	्

(d) Various Signs

A.3 Stop-Words

अक्सर अगाडि अगाडी अघि अझै अठार अथवा अनि अनुसार अन्तर्गत अन्य अन्यत्र अन्यथा अब अरु अरुलाई अरू अर्को अर्थात अर्थात् अलग अलि अवस्था अहिले आए आएका आएको आज आजको आठ आत्म आदि

आदिलाई आफनो आफू आफूलाई आफै आफैँ आफ्नै आफ्नो आयो उ उक्त उदाहरण उनको उनलाई उनले उनि उनी उनीहरुको उन्नाइस उप उसको उसलाई उसले उहालाई ऊ एउटा एउटै एक एकदम एघार ओठ औ औं कता कति कतै कम कमसेकम कसरि कसरी कसै कसैको कसैलाई कसैले कसैसँग कस्तो कहाँबाट कहिलेकाहीं का काम कारण कि किन किनभने कुन कुनै कुन्नी कुरा कृपया के केहि केही को कोहि कोहिपनि कोही कोहीपनि क्रमशः गए गएको गएर गयौ गरि गरी गरे गरेका गरेको गरेर गरौं गर्छ गर्छन् गर्छु गर्दा गर्दै गर्न गर्नु गर्नुपर्छ गर्ने गैर घर चार चाले चाहनुहुन्छ चाहन्छु चाहिं चाहिए चाहिंले चाहीं चाहेको चाहेर चोटी चौथो चौध छ छन छन् छु छू छैन छैनन् छौ छौं जता जतातै जना जनाको जनालाई जनाले जब जबकि जबकी जसको जसबाट जसमा जसरी जसलाई जसले जस्ता जस्तै जस्तो जस्तोसुकै जहाँ जान जाने जाहिर जुन जुनै जे जो जोपनि जोपनी झैं ठाउँमा ठीक ठूलो त तता तत्काल तथा तथापि तथापी तदनुसार तपाइ तपाई तपाईको तब तर तर्फ तल तसरी तापनि तापनी तिन तिनि तिनिहरुलाई तिनी तिनीहरु तिनीहरुको तिनीहरू तिनीहरूको तिनै तिमी तिर तिरको ती तीन तुरन्त तुरुन्त तुरुन्तै तेश्रो तेस्कारण तेस्रो तेह्र तैपनि तैपनी त्यत्तिकै त्यत्तिकैमा त्यस त्यसकारण त्यसको त्यसले त्यसैले त्यसो त्यस्तै त्यस्तो त्यहाँ त्यहिँ त्यही त्यहीँ त्यहीं त्यो त्सपछि त्सैले थप थरि थरी थाहा थिए थिएँ थिएन थियो दर्ता दश दिए दिएको दिन दिनुभएको दिनुहुन्छ दुइ दुइवटा दुई देखि देखिन्छ देखियो देखे देखेको देखेर दोश्री दोश्रो दोस्रो द्वारा धन्न धेरै धौ न नगर्नु नगर्नू नजिकै नत्र नत्रभने नभई नभएको नभनेर नयाँ नि निकै निम्ति निम्न निम्नानुसार निर्दिष्ट नै नौ पक्का पक्कै पछाडि पछाडी पछि पछिल्लो पछी पटक पनि पन्ध्र पर्छ पर्थ्यो पर्दैन पर्ने पर्नेमा पर्याप्त पहिले पहिलो पहिल्यै पाँच पांच पाचौँ पाँचौं पिच्छे पूर्व पो प्रति प्रतेक प्रत्यक प्राय प्लस फरक फेरि फेरी बढी बताए बने बरु बाट बारे बाहिर बाहेक बाह्र बिच बिचमा बिरुद्ध बिशेष बिस बीच बीचमा बीस भए भएँ भएका भएकालाई भएको भएन भएर भन भने भनेको भनेर भन् भन्छन् भन्छु भन्दा भन्दै भन्नुभयो भन्ने भन्या भयेन भयो भर भरि भरी भा भित्र भित्री भीत्र म मध्य मध्ये मलाई मा मात्र मात्रै माथि माथी मुख्य मुनि मुन्तिर मेरो मैले यति यथोचित यदि यद्ध्यपि यद्यपि यस यसका यसको यसपछि यसबाहेक यसमा यसरी यसले यसो यस्तै यस्तो यहाँ यहाँसम्म यही या यी यो र रही रहेका रहेको रहेछ राखे राख्छ राम्रो रुपमा रूप रे लगभग लगायत लाई लाख लागि लागेको ले वटा वरीपरी वा वाट वापत वास्तवमा शायद सक्छ सक्रे सँग संग सँगको सँगसँगै सँगै संगै सङ्ग सङ्गको सट्टा सत्र सधै सबै सबैको सबैलाई समय समेत सम्भव सम्म सय सरह सहित सहितै सही साँच्चै सात साथ साथै सायद सारा सुनेको सुनेर सुरु सुरुको सुरुमै सो सोचेको सोचेर सोही सोह्र स्थित स्पष्ट हजार हरे हरेक हामी हामीले हाम्रा हाम्रो हुँदैन हुन हुनत हुनु हुने हुनेछ हुन् हुन्छ हुन्थ्यो हैन हो होइन होकि होला

A.4 Stemmer Rules

काल
सामान्य भूत
ैँ िस् यो ी ौँ ौं ौ े िन् # सामान्य वर्तमान
छु छस् छ छे छौँ छौं छौ छन् छिन् # सामान्य भविष्यत्
नेछु नेछस् नेछ नेछिन् नेछौँ नेछौं नेछौ नेछन् नेछिन्
अपूर्ण भूत/वर्तमान/भविष्यत्
दै ँदै िरहेको िरहेका िरहेकी
पूर्ण भूत/वर्तमान/भविष्यत्
ेको ेका ेकी

अज्ञात भूत
ेछु ेछौँ ेछौं िछस् ेछौ ेछ ेछन् िछ
अभ्यस्त भूत
थैँ थें थ्यौँ थ्यौं थिस् थ्यौ थ्यो थे थी थिन् नुहुन्थ्यो

वचन
हरू

विभक्ति
ले बाट द्वारा लाई देखि को का कि रो रा री नो ना नी मा

A.5 Plot Data for Performance Measures with TFIDF

A.5.1 tfidf-homogeneity.txt

```
Size DBSCAN KM MBKM
2000 0.014162557 0.065725926 0.453432557
3000 0.008039397 0.166349805 0.568853662
4000 0.004998860 0.136843898 0.221093048
5000 0.004111333 0.479450862 0.523651710
6000 0.004478619 0.501097067 0.538269372
7000 0.004490954 0.395121842 0.545322140
8000 0.003865849 0.427014022 0.506561918
9000 0.003525922 0.456360194 0.539257283
10000 0.003190042 0.438415457 0.447876592
```

A.5.2 tfidf-completeness.txt

```
Size DBSCAN KM MBKM
2000 0.095062318 0.159497783 0.446961315
3000 0.054708011 0.165164577 0.586650900
4000 0.037372858 0.116724418 0.163240458
5000 0.065374395 0.455990192 0.453139567
6000 0.091272191 0.527766597 0.477199390
7000 0.112061915 0.346275902 0.537703477
8000 0.129309259 0.539584375 0.539595857
9000 0.144541874 0.556505987 0.531231006
```

```
10000 0.156913349 0.527807275 0.480701863
```

A.5.3 tfidf-v-measure.txt

```
Size DBSCAN KM MBKM
2000 0.024652360 0.093090905 0.450173681
3000 0.014018728 0.165755072 0.577615223
4000 0.008818225 0.125985964 0.187812563
5000 0.007736147 0.467426332 0.485850590
6000 0.008538274 0.514086177 0.505898016
7000 0.008635821 0.369089799 0.541486011
8000 0.007507260 0.476744209 0.522557340
9000 0.006883919 0.501482200 0.535214055
10000 0.006252962 0.478976244 0.463709041
```

A.5.4 tfidf-silhouette.txt

```
Size DBSCAN KM MBKM
2000 0.003240279 0.008163300 0.008359614
3000 0.002916194 0.006228649 0.009700666
4000 0.001828447 0.006081243 0.007248560
5000 0.000450257 0.010336318 0.011493876
6000 0.000835649 0.011302530 0.012577101
7000 0.001339375 0.009921182 0.013322621
8000 0.000874452 0.012563696 0.012047895
9000 0.000976385 0.013747476 0.013033515
10000 0.001202579 0.011840634 0.013886573
```

A.6 Plot Data for Performance Measures with TFIDF and LSI

A.6.1 tfidf-lsi-homogeneity.txt

```
Size DBSCAN KM MBKM
2000 0.150149442 0.543283235 0.290555770
3000 0.236370347 0.301166180 0.153040141
4000 0.301301370 0.166439740 0.327868864
5000 0.542138617 0.544907224 0.538831489
```

```
6000 0.547528443 0.612628729 0.577887273
7000 0.561548739 0.644088334 0.580925741
8000 0.633093584 0.629940794 0.649567713
9000 0.586737996 0.644012688 0.630569416
10000 0.456941973 0.652657490 0.509335828
```

A.6.2 tfidf-lsi-completeness.txt

```
Size DBSCAN KM MBKM
2000 0.107527804 0.575136726 0.300421680
3000 0.089563840 0.255707946 0.130022070
4000 0.074772534 0.121641776 0.263777399
5000 0.184681096 0.465347943 0.460097958
6000 0.198763208 0.540454683 0.535832003
7000 0.215189415 0.580377200 0.457917074
8000 0.288844825 0.705226792 0.686554082
9000 0.315266314 0.636742599 0.586108722
10000 0.284778871 0.623700829 0.544688453
```

A.6.3 tfidf-lsi-v-measure.txt

```
Size DBSCAN KM MBKM
2000 0.125313663 0.558756374 0.295406373
3000 0.129904974 0.276581661 0.140595213
4000 0.119811913 0.140557616 0.292351703
5000 0.275509187 0.501994870 0.496361917
6000 0.291651419 0.574282939 0.556065611
7000 0.311145639 0.610575265 0.512138721
8000 0.396698529 0.665461220 0.667548971
9000 0.410150425 0.640357009 0.607526712
10000 0.350879769 0.637850691 0.526419266
```

A.6.4 tfidf-lsi-silhouette.txt

```
Size DBSCAN KM MBKM
2000 0.008862449 0.037925249 0.028236413
3000 0.026407684 0.034362360 0.029385830
```

```
4000 0.032838504 0.033377461 0.033375921
5000 0.069554143 0.055030756 0.055093065
6000 0.055555315 0.060214767 0.055856954
7000 0.040973935 0.058932564 0.057641862
8000 0.066511200 0.075520788 0.077399998
9000 0.051288355 0.087783905 0.087561338
10000 0.024507214 0.080085281 0.079607913
```

A.7 Plot Data for Time with TFIDF and TFIDF + LSI Combination

A.7.1 time-tfidf.txt

```
Size DBSCAN KM MBKM
2000 0.294145107 7.479647875 0.17186904
3000 0.628072023 8.999480963 0.160945892
4000 1.148311853 12.99931693 0.230267048
5000 1.841221094 7.039831161 0.278146982
6000 2.374995232 7.380022049 0.376816988
7000 3.137669325 32.34203601 0.439945221
8000 4.065223932 20.88313913 0.40697813
9000 5.593692303 20.47509003 0.61588788
10000 8.453108072 44.31739902 1.282078981
```

A.7.2 time-tfidf-lsi.txt

```
Size DBSCAN KM MBKM
2000 1.561387062 0.019575119 0.044759035
3000 2.615876198 0.032418013 0.068073034
4000 6.345207691 0.046323776 0.057080030
5000 8.024958372 0.075416088 0.078593016
6000 10.305550814 0.052417755 0.078658104
7000 14.444278002 0.098990917 0.108935833
8000 22.888608932 0.070907116 0.072808266
9000 27.065243006 0.102853775 0.089255810
10000 31.963739157 0.117464066 0.105010271
```

APPENDIX B

SOURCE CODE

B.1 Constants.py

```
EMPTY_STRING = ''
SPACE = ' '
```

B.2 Parser.py

```
from Constants import *


class Parser:
    def __init__(self):

        VOWELS = 'अआइईउऊऋएऐओऔ'
        CONSONANTS = 'कखगघङचछजझञटठडढणतथदधनपफबभमयरलवशषसह'


        # spaces added for clarity
        # (needed to properly see the dependent vowels in some editors)

        DEPENDENT_VOWELS = 'ा ि ी ु ू ृ े ै ो ौ ं ः ँ ्'

        whitelist = VOWELS
        whitelist += CONSONANTS
        whitelist += DEPENDENT_VOWELS
        self.whitelist = whitelist


    @staticmethod
    def remove_blacklist(value, black_list):
        value = str(value);
        for item in black_list:
            value = value.replace(item, EMPTY_STRING)
        return value
```

```
        @staticmethod
        def remove_zwnj(value):
            HTML_ZWNJ_TEXT = '&zwnj;'
            HTML_ZWNJ_NUMERIC = '&#8204;'
            ZWNJ = '\u200C'  # zero width non-joiner
            return Parser.remove_blacklist(value,
                [HTML_ZWNJ_TEXT, HTML_ZWNJ_NUMERIC, ZWNJ])

        @staticmethod
        def remove_zwj(value):
            HTML_ZWJ_TEXT = '&zwj;'
            HTML_ZWJ_NUMERIC = '&#8205;'
            ZWJ = '\u200D'  # zero width joiner
            return Parser.remove_blacklist(value,
                [HTML_ZWJ_TEXT, HTML_ZWJ_NUMERIC, ZWJ])

        # replaces chars not contained in whitelist with spaces
        # exceptions: replaces zero width non-joiners and
        # zero width joiners (both HTML and unicode)
        # with empty strings
        def parse_valid_chars(self, value):
            value = Parser.remove_zwnj(value)
            value = Parser.remove_zwj(value)
            return EMPTY_STRING.join(c if c in
                self.whitelist else SPACE for c in value)
```

B.3 Tokenizer.py

```
from Parser import Parser
from Constants import *


class Tokenizer:
    def __init__(self):
        self.parser = Parser()

    def tokenize(self, value):
```

```
        value = self.parser.parse_valid_chars(value)
        tokens = value.split(SPACE)
        # remove empty elements from tokens
        return list(filter(None, tokens))
```

B.4 Stemmer.py

```
class Stemmer:
    def __init__(self):
        rule_file_name = 'Data/StemmerRules.txt'
        rule_file = open(rule_file_name, 'r', encoding='utf-8')
        self.rules = rule_file.read().split('\n')
        # remove blank lines and comments
        self.rules = [x for x in [y.lstrip() for y in self.rules]
                 if len(x) > 0 and not x.startswith('#')]
        # split lines to individual rules
        self.rules = [x for row in self.rules for x in row.split(' ')]
        self.rules.sort(key=len, reverse=True)

    def remove_suffix(self, word, suffix):
        if not word.endswith(suffix):
            return word
        return word[:len(word) - len(suffix)]

    def find_root(self, word):
        for suffix in self.rules:
            word = self.remove_suffix(word, suffix)
        return word
```

B.5 Preprocessor.py

```
from Parser import Parser
from Tokenizer import Tokenizer
from Stemmer import Stemmer
import pandas as pd

from ast import literal_eval
```

```
from Constants import *

from sklearn import preprocessing


class Preprocessor:

    @staticmethod
    def tokenize(df):
        df['data'] = df.title.map(lambda x: x + ' ') + df.story
        df.drop(columns=['title', 'story'], inplace=True)

        p = Parser()
        t = Tokenizer()

        print('starting tokenization')

        df.data = df.data.map(t.tokenize)
        print('data converted')

        df.category = df.category.map(p.parse_valid_chars)
        print('category converted')

        print('tokenization complete')
        print()

        return df

    @staticmethod
    def remove_stopwords(df):
        stop_words_txt = 'Data/StopWords.txt'
        stop_words_file = open(stop_words_txt, 'r', encoding='utf-8')
        stop_words = stop_words_file.read().splitlines()

        print('removing stop words')
        # note: single characters are also considered stop words
        df.data = df.data.map(lambda tokens:
            [t for t in tokens if len(t) > 2 and t not in stop_words])
        print('stop words removed from data')
```

```
        print()

        return df

    @staticmethod
    def perform_stemming(df):
        print('Stemming')

        s = Stemmer()
        df.data = df.data.map(lambda tokens:
            [s.find_root(t) for t in tokens])

        print('Stemming Complete')
        print()

        return df

    @staticmethod
    def encode_labels(df):
        print('Encoding labels')

        le = preprocessing.LabelEncoder()
        le.fit(df.category)
        df['target'] = le.transform(df.category)

        print('Encoding complete')
        print()
        return df

    @staticmethod
    def start():
        input_csv = 'Data/News.csv'
        output_csv = 'Data/NewsFinal.csv'

        df = pd.read_csv(input_csv,
                                 encoding="utf-8",
                                 delimiter=';',
                                 quotechar='"',
                                 usecols=['title', 'story', 'category'])
```

```
        df = Preprocessor.tokenize(df)

        df = Preprocessor.remove_stopwords(df)

        df = Preprocessor.perform_stemming(df)

        # change df.data back to space separated text to use in tf-idf
        df.data = df.data.map(lambda x: SPACE.join(x))

        df = Preprocessor.encode_labels(df)

        df.to_csv(output_csv,
                    encoding="utf-8",
                    sep=';',
                    quotechar='"',
                    index=False)  # suppress index number
```

B.6 EpsilonTuner.py

```
import numpy as np
from sklearn import metrics
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
    df = Analysis.read_data(data_size)

    X = Analysis.tfidf(df)
    X = Analysis.lsi(X)

    labels = df.target

    min_samples = 10

    for epsilon in np.arange(0.1, 1.0, 0.01):
        model = Analysis.dbscan_model(eps=epsilon, min_samples=min_samples)
```

```
        model.fit(X)

        homogeneity = metrics.homogeneity_score(labels, model.labels_)
        completeness = metrics.completeness_score(labels, model.labels_)
        v_measure = metrics.v_measure_score(labels, model.labels_)

        print(data_size, epsilon, homogeneity, completeness, v_measure)
```

B.7 MinPtsTuner.py

```
import numpy as np
from sklearn import metrics
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
    df = Analysis.read_data(data_size)

    X = Analysis.tfidf(df)
    X = Analysis.lsi(X)

    labels = df.target

    epsilon = 0.55

    for min_samples in np.arange(5, 101, 1):
        model = Analysis.dbscan_model(eps=epsilon, min_samples=min_samples)
        model.fit(X)

        homogeneity = metrics.homogeneity_score(labels, model.labels_)
        completeness = metrics.completeness_score(labels, model.labels_)
        v_measure = metrics.v_measure_score(labels, model.labels_)

        print(data_size, min_samples, homogeneity, completeness, v_measure)
```

B.8 Analysis.py

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer

from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import DBSCAN


class Analysis:

    @staticmethod
    def read_data(nrows):
        data_csv = 'Data/NewsFinal.csv';
        df = pd.read_csv(data_csv,
                         encoding="utf-8",
                         delimiter=';',
                         quotechar='"', nrows=nrows)
        return df

    @staticmethod
    def tfidf(df):
        vectorizer = TfidfVectorizer(tokenizer=lambda x: x.split(' '))
        return  vectorizer.fit_transform(df.data)

    @staticmethod
    def lsi(tf_idf):
        svd = TruncatedSVD(n_components=100)
        normalizer = Normalizer(copy=False)
        lsa = make_pipeline(svd, normalizer)
        return lsa.fit_transform(tf_idf)

    @staticmethod
```

```
    def kmeans_model(true_k, X):
        return KMeans(n_clusters=true_k, init='k-means++',
            max_iter=100, n_init=1, verbose=0)

    @staticmethod
    def mini_batch_kmeans_model(true_k, X):
        return MiniBatchKMeans(n_clusters=true_k, init='k-means++',
            batch_size=1000, verbose=0)

    @staticmethod
    def dbscan_model(eps, min_samples):
        return DBSCAN(eps=eps, min_samples=min_samples)
```

B.9 TFIDFAnalysis.py

```
from sklearn import metrics
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
    df = Analysis.read_data(data_size)

    true_k = len(df.category.unique())

    X = Analysis.tfidf(df)

    models = [Analysis.kmeans_model(true_k, X),
              Analysis.mini_batch_kmeans_model(true_k, X),
              Analysis.dbscan_model(eps=0.55, min_samples=10)]

    labels = df.target

    for model in models:
        model.fit(X)

        homogeneity = metrics.homogeneity_score(labels, model.labels_)
        completeness = metrics.completeness_score(labels, model.labels_)
```

```
        v_measure = metrics.v_measure_score(labels, model.labels_)
        silhouette = metrics.silhouette_score(X, model.labels_)

        print(data_size, homogeneity, completeness, v_measure, silhouette)
```

B.10 TFIDF_LSI_Analysis.py

```
from sklearn import metrics
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
    df = Analysis.read_data(data_size)

    true_k = len(df.category.unique())

    X = Analysis.tfidf(df)
    X = Analysis.lsi(X)

    models = [Analysis.kmeans_model(true_k, X),
              Analysis.mini_batch_kmeans_model(true_k, X),
              Analysis.dbscan_model(eps=0.55, min_samples=10)]

    labels = df.target

    for model in models:
        model.fit(X)

        homogeneity = metrics.homogeneity_score(labels, model.labels_)
        completeness = metrics.completeness_score(labels, model.labels_)
        v_measure = metrics.v_measure_score(labels, model.labels_)
        silhouette = metrics.silhouette_score(X, model.labels_)

        print(data_size, homogeneity, completeness, v_measure, silhouette)
```

B.11 TimeWith_TFIDF.py

```
from time import time
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
df = Analysis.read_data(data_size)

true_k = len(df.category.unique())

X = Analysis.tfidf(df)

models = [Analysis.kmeans_model(true_k, X),
            Analysis.mini_batch_kmeans_model(true_k, X),
            Analysis.dbscan_model(eps=0.55, min_samples=10)]

labels = df.target

for model in models:
    t0 = time()
    model.fit(X)
    duration = time() - t0

    print(data_size, duration)
```

B.12 TimeWith_TFIDF_LSI.py

```
from time import time
from Analysis import *

data_sizes = list(range(2000, 10_001, 1000))

for data_size in data_sizes:
    df = Analysis.read_data(data_size)

    true_k = len(df.category.unique())
```

```
X = Analysis.tfidf(df)
X = Analysis.lsi(X)

models = [Analysis.kmeans_model(true_k, X),
            Analysis.mini_batch_kmeans_model(true_k, X),
            Analysis.dbscan_model(eps=0.55, min_samples=10)]

labels = df.target

for model in models:
    t0 = time()
    model.fit(X)
    duration = time() - t0

    print(data_size, duration)
```