Document Clustering for IDEAL

Final Project Report

Date: 05/13/2015

CS5604 Information Storage and Retrieval

Virginia Polytechnic Institute and State University

Blacksburg, Virginia

Spring 2015

Submitted by

Names of Students	Email ID
Kalidas, Rubasri	krubasri@vt.edu
Thumma, Sujit Reddy	sujitt@vt.edu
Torkey, Hanaa	htorkey@vt.edu

Instructor

Prof. Edward A. Fox

Abstract

Document clustering is an unsupervised classification of text documents into groups (clusters). The documents with similar properties are grouped together into one cluster. Documents which have dissimilar patterns are grouped into different clusters. Clustering deals with finding a structure in a collection of unlabeled data. The main goal of this project is to enhance Solr search results with the help of offline data clustering. In our project, we propose to iterate and optimize clustering results using various clustering algorithms and techniques. Specifically, we evaluate the K-Means, Streaming K-Means, and Fuzzy K-Means algorithms available in the Apache Mahout software package. Our data consists of tweet archives and web page archives related to tweets. Document clustering involves data pre-processing, data clustering using clustering algorithms, and data post-processing. The final output which includes document ID, cluster ID, and cluster label, is stored in HBase for further indexing into the Solr search engine. Solr search recall is enhanced by boosting document relevance scores based on the clustered sets of documents. We propose three metrics to evaluate the cluster results: Silhoutte scores, confusion matrix with homogeneous labelled data, and human judgement. To optimize the clustering results we identify various tunable parameters that are input to the clustering algorithms and demonstrate the effectiveness of those tuning parameters. Finally, we have automated the entire clustering pipeline using several scripts and deployed them on a Hadoop cluster for large scale data clustering of tweet and webpage collections.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We deal with clustering in almost every aspect of daily life. Clustering is the subject of active research in several fields such as statistics, pattern recognition, and machine learning. In data mining, clustering deals with very large data sets with different attributes associated with the data. This imposes unique computational requirements on relevant clustering algorithms. A variety of algorithms have recently emerged that meet these requirements and were successfully applied to real life data mining problems [1]. Clustering methods are divided into two basic types: hierarchical and flat clustering. Within each of these types there exists a wealth of subtypes and different algorithms for finding the clusters. Flat clustering algorithm goal is to create clusters that are coherent internally, and clearly different from each other. The data within a cluster should be as similar as possible; data in one cluster should be as dissimilar as possible from documents in other clusters. Hierarchical clustering builds a cluster hierarchy that can be represented as a tree of clusters. Each cluster can be represented as child, a parent and a sibling to other clusters. Even though hierarchical clustering is superior to flat clustering in representing the clusters, it has a drawback of being computationally intensive in finding the relevant hierarchies [8].

The initial goal of the project is to use flat clustering methods to partition data into semantically related clusters. Further, based upon the clustering quality and understanding of the data we enhance cluster representation using hierarchical clustering. This may also result in hybrid clusters between flat and hierarchical arrangement. Clustering algorithms provided in the Apache Mahout library will be used in our work [2]. Mahout is a suite of generally designed machine learning libraries. It is associated with Apache Hadoop [4] for large scale machine learning in distributed environment. Currently Mahout supports mainly recommendation mining, clustering and classification algorithms. For our project we identified to evaluate a set of clustering algorithms - k-means, Canopy, fuzzy k-means, streaming k-means, and spectral k-means available in the Mahout library. We have used various collections: web pages and tweet as our data set to evaluate clustering.

Since clustering is an unsupervised classification finding the appropriate number of clusters apriori to categorize the data is a difficult problem to address. The most efficient way to learn about the number of clusters is to learn from the data itself. We address this challenge by estimating the number of clusters using methods like cross-validation and semi-supervised learning. Figure 1.1 shows an overview of the project.

In the project we evaluate a flat clustering algorithm on tweet and web page data sets using Mahout K-means clustering. The algorithm is initialized with random centroid points. We found, by emperical evaluation, that the best possible number of clusters for

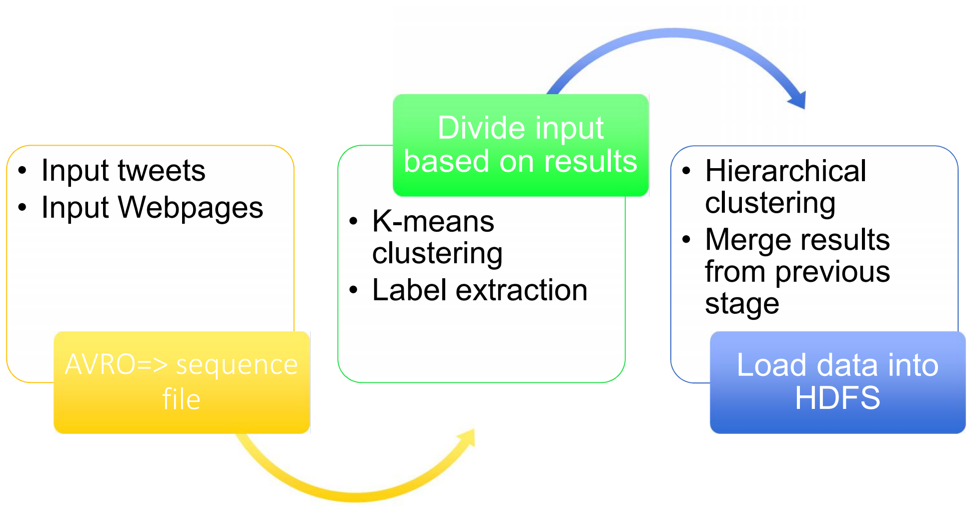


Figure 1.1: Project overview

small data set is 5 and large data set is 10. For the divisive hierarchical clustering, we have done two layer clustering. The first layer corresponds to clustered points output from a flat cluster like K-Means. The second layer corresponds to further flat clustering of clustered points from layer 1. For labeling, we chose the top terms (based on frequency of occurrence) in the clustered points closed to centroids. These top terms are identified from the K-means cluster dump results using Mahout tools. Future work include multiple layers of hierarchical clustering and advanced cluster labeling techniques.

The report is organized as follows: chapter 2 provides a brief literature review on existing clustering algorithms including flat clustering and hierarchical clustering, labeling procedures, and open-source tools such as Apach Solr and Apache Mahout. In chapter 3, we present our project requirements with pointers to relevant sections. In chapter 4, the design and implementation of the project is discussed along with the tools and dependencies. Project milestones and brief time line of weekly progress is presented in chapter 5. In chapter 6, we discuss the techniques used in clustering evaluation including Silhoutte scores, confusion matrix, and human judgement. Conclusion and future work is presented in chapter 7. Three appendix chapters are included: Appendix A provide detailed instructions to reproduce the clustering results we have obtained and a user guide to run various scripts we have developed over the course of the project. In appendix B we have detailed the implementation and evaluation of clustering algorithms to aid future developers to continue on this project. In appendix C we present a list of inventory files developed as part of this project and VTechWorks submissions.

Chapter 2

Literature Review

Clustering objects into groups is usually based on a similarity metric between objects, with the goal that objects within the same group are very similar, and objects between different groups are less similar. In this review we focus on document clustering for web pages and tweet data. The application of text clustering can be both online or offline. Online applications are considered to be more efficient compared to offline applications in terms of cluster quality, however, they suffer from latency issues.

Text clustering algorithms may be classified as flat clustering and hierarchical clustering. In the next two subsections we elaborate more details about these algorithms.

2.0.1 Flat clustering algorithms

Flat clustering explains how to create a flat set of clusters without any explicit structure that would relate clusters to each other [6]. Flat clustering methods are conceptually simple, but they have a number of drawbacks. Most of the flat clustering algorithms, like k-means, require a pre-specified number of clusters as input and are non-deterministic.

2.0.2 Hierarchical clustering algorithms

Hierarchical clustering builds a cluster hierarchy, or in other words, a tree of clusters. Figure 2.1 shows an example of hierarchical clustering for a set of points. Hierarchical clustering outputs is structured and more informative than flat clustering.

Hierarchical clustering algorithms are further subdivided into two types (1) agglomerative methods - a bottom-up cluster hierarchy generation by fusing objects into groups and groups into higher clusters. (2) divisive methods - a top-down cluster hierarchy generation by partitioning a single cluster encompassing all objects successively into finer clusters. Agglomerative techniques are more commonly used [10].

Hierarchical clustering does not require knowing the pre-specified number of clusters. However this advantage came with the cost of the algorithm complexity. Hierarchical clustering algorithms have a complexity that is at least quadratic in the number of documents compared to the linear complexity of flat algorithms like k-means or EM [10].

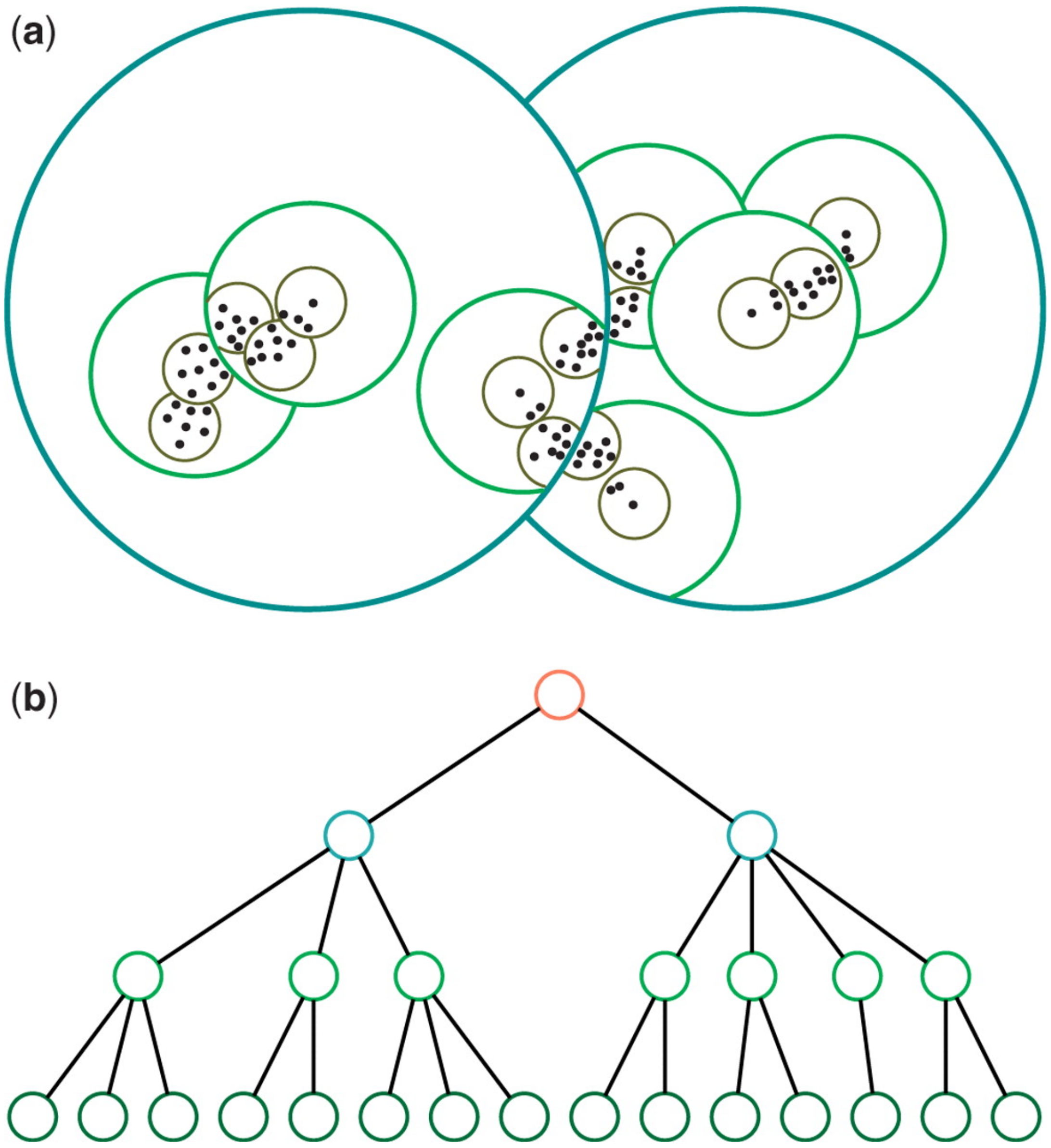


Figure 2.1: Hierarchical clustering

2.0.3 Clustering in Solr

Solr is an enterprise search engine that is optimized to search a given query in huge volumes of text centric data. The results obtained from the search are often sorted and ranked by relevance. Currently, Solr's built-in clustering component provides search result clustering. Solr's clustering component is responsible for taking in the request, identifying the clustering engine to be used and then delegating the work to that engine. Once the processing is done, the results are added to the search response. Its implementation is based on the Carrot2 framework [19]. Carrot2 has three algorithms: Lingo, STC, and k-means. Lingo was built to handle search results clustering. Due to complex operations to obtain semantically related documents in clusters, Lingo is effective only with for small amounts of data. STC (Suffix Tree Clustering) algorithm is a Generalized Suffix Tree (GST) built for all input documents. The algorithm traverses the GST to identify words and phrases that occurred more than once in the input documents. Each such word or phrase gives rise to one base cluster.The last stage of the clustering process is merging

base clusters to form the final clusters. K-Means is a generic clustering algorithm that can also be applied to clustering textual data. As opposed to Lingo and STC, bisecting k-means creates non-overlapping clusters.
Carrot2 is suited for clustering small to medium collections of documents. It may work for longer documents, but processing times will be too long for online search. The integration between Solr and Carrot2 is implemented as APIs [20]. Learning about Solr-Carrot2 integration will help us in integrating our clustering techniques with Solr.

2.0.4 Data Collection

We evaluate clustering techniques on various tweet and web page collections. The collections include small data sets ($< 500MB$) and big data sets ($> 1GB$) and are related to various events of historical importance such as Ebola outbreak, Charlie Hebdo shooting incident, various incidents that took place on January, 25, Plane crash incident, Winter storm, Suicide bomb attack, Elections, Diabetes, tweets related to Egypt, Malaysia Airlines, Shooting, Storm, and Tunisia. The web pages for corresponding events are crawled using web links in each of the tweet collection.

Web pages and tweets clustering

In clustering of web pages, clustering approaches could be classified in two broad categories: term-based clustering and link-based clustering. Term-based clustering is based on common terms shared among documents [13]. However, it does not adapt well to the web environment since it ignores the availability of hyperlinks between web pages. Link-based clustering could cluster web pages based on the information in the link. However, it suffers from the fact that pages without sufficient information in the links could not be clustered. It is natural to combine link and content information in the clustering algorithms to overcome the above problems [14]. For tweets, a standard document clustering algorithms can be used [12]. One interesting point in tweet clustering is the automatic detection of tweet topics, for example, the hash-tags that appear in tweets can be viewed as an approximate indicator of a tweet topic.

2.0.5 Mahout clustering

Mahout provides both in-memory and map-reduce versions of various clustering algorithms. These algorithms are K-Means, Canopy, Fuzzy K-Means, and streaming k-mean, and Spectral Clustering [9]. All these algorithms expect data in the form of vectors, so the first step is to convert the input data into this format, a process known as vectorization. Essentially, clustering is the process of finding nearby points in n-dimensional space, where each vector represents a point in this space, and each element of a vector represents a dimension in this space [21]. It is important to choose the right vector format for the clustering algorithm. For example, one should use the “Sequential Access Sparse Vector” for k-means. Other possibilities are the “Dense Vector” and the “Random Access Sparse Vector” formats. The input to a clustering algorithm is a sequence file containing key-value pairs of objects.

Document clustering using Mahout

For Mahout, we need to generate sequence files from cleaned data in HDFS and vectorize them in the format understandable by Mahout. Once the vectors are generated they will be input to common clustering algorithm like k-means. Due to the nature of text data with high dimensional features it is possible that dimensionality reduction techniques will be used to transform feature vectors for improving cluster quality.

For clustering text data, vector generation can be improved by removing noise and using a good weighting technique. Mahout allows specifying custom Lucene analyzers to its clustering sub-commands for this. Also, cluster quality depends on the measure used to calculate similarity between two feature vectors. Mahout supplies a large number of Distance Measure implementations (Manhattan, Squared Euclidean, Euclidean, Weighted Euclidean and Weighted Manhattan) and also allows the user to specify his/her own if the defaults don't suit the purpose. Within each dimension, points can be normalized to remove the effect of outliers - the normalization p-norm should match the p-norm used by the distance measure. Finally, if the dimensions are not comparable, then one should normalize across dimensions, a process known as weighting (this should be done during the vectorization process, which the user controls fully) [3].

Once the data is vectorized, the user invokes the appropriate clustering algorithm either by calling the appropriate Mahout sub-command from the command line, or through a program by calling the appropriate driver run method. All algorithms require the initial centroids to be provided, and the algorithm iteratively modifies the centroids until they converge. The user can either guess randomly or use the Canopy cluster to generate the initial centroids.

Finally, the output of the clustering algorithm (sequence files in binary format) can be read using the Mahout cluster dumper sub-command to get a human readable format.

K-Means Algorithm

The k-means clustering algorithm is known to be efficient in clustering large data sets. This algorithm is one of the simplest and the best known unsupervised learning algorithms. It solves the well-known clustering problem. The K-Means algorithm aims to partition a set of objects, based on their attributes/features, into k clusters, where k is a predefined constant. The algorithm defines k centroids, one for each cluster. The centroid of a cluster is formed in such a way that it is closely related, in terms of similarity (where similarity can be measured by using different methods such as Euclidean distance or Extended Jaccard) to all objects in that cluster [9]. Technically, what k-means is interested in, is the variance. It minimizes the overall variance, by assigning each object to the cluster such that the variance is minimized. Coincidentally, the sum of squared deviations, one objects contribution to the total variance, over all dimensions is exactly the definition of squared euclidean distance.

In Mahout implementation of k-mean, Each object will be represented as vector in space. Initially k points will be chosen by the algorithm randomly and treated as centers, every object closest to each center are clustered. There are several algorithms for the distance measure and the user should choose the required one.

Creating Vector Files:

- Unlike Canopy algorithm, the k-means algorithm requires vector files as input, therefore you have to create vector files.
- To generate vector files from sequence file format, Mahout provides the seq2parse utility.
- After creating vectors, proceed with k-means algorithm.

K-means clustering job requires input vector directory, output clusters directory, distance measure, maximum number of iterations to be carried out, and an integer value representing the number of clusters the input data is to be divided into. The next figure shows K-Means in action for Mahout implementation.

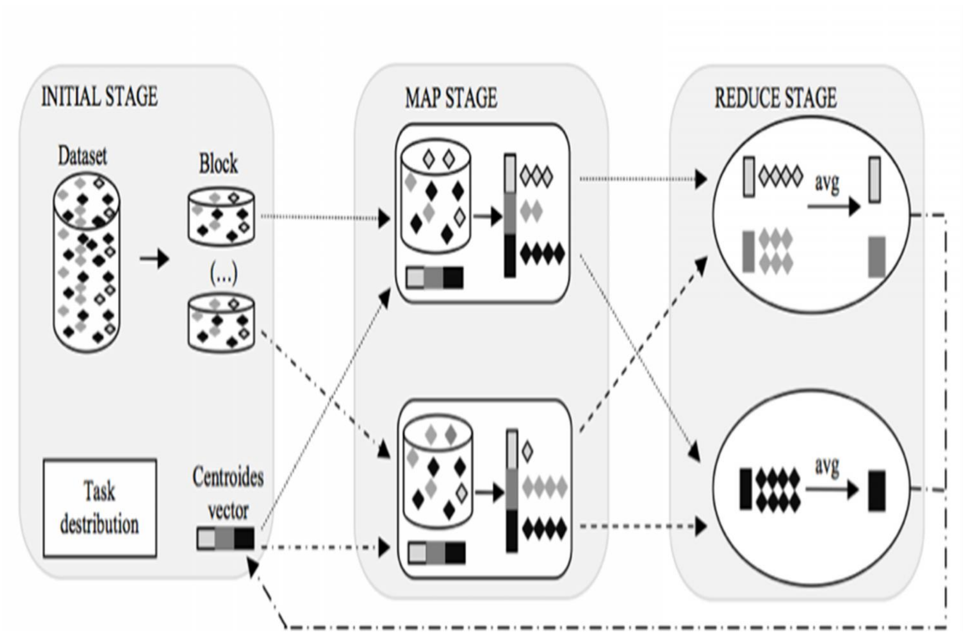


Figure 2.2: The three stage K-Means clustering in Mahout

Fuzzy K-Means

Fuzzy K-Means (also called Fuzzy C-Means) is an extension of K-Means, the popular simple clustering technique. While K-Means discovers hard clusters (a point belong to only one cluster), Fuzzy K-Means is a more statistically formalized method and discovers soft clusters where a particular point can belong to more than one cluster with certain probability.

Like K-Means, Fuzzy K-Means works on those objects which can be represented in n-dimensional vector space and a distance measure is defined. The algorithm is similar to k-means.

- Initialize k clusters
- Until converged
  1. Compute the probability of a point belong to a cluster for every pair.
  2. Recompute the cluster centers using above probability membership values of points to clusters.

Similar to K-Means, the program doesn't modify the input directories. And for every iteration, the cluster output is stored in a directory cluster-N. The code has set number of reduce tasks equal to number of map tasks. FuzzyKMeansDriver - This is similar to KMeansDriver. It iterates over input points and cluster points for specified number of iterations or until it is converged.During every iteration i, a new cluster-i directory is created which contains the modified cluster centers obtained during FuzzyKMeans iteration. This will be feeded as input clusters in the next iteration. FuzzyKMeansMapper - reads the input cluster during its configure() method, then computes cluster membership probability of a point to each cluster. FuzzyKMeansReducer - Multiple reducers receives certain keys and all values associated with those keys. The reducer sums the values to produce a new centroid for the cluster which is output.

2.0.6 Clustering Evaluation

Our evaluation approach is iterative and we aim to produce clustering results that can be improved over time by optimizing feature vectors and clustering algorithms. To ensure best results our approach is to: research $\rightarrow$ identify improvements $\rightarrow$ implement (or find an equivalent open source implementation) $\rightarrow$ integrate and evaluate. The end goal will be to document our experience with various clustering algorithms and techniques for identifying feature vectors using different set of data which ensures quality clustering with optimal performance.

2.0.7 Cluster Labeling

Clusters that are obtained as a result of the clustering process need to be labeled appropriately in order to understand the purpose of each cluster and to evaluate the effectiveness of clustering. Cluster labeling is based on selecting words from each cluster and then use them to label the clusters. There are three ways of selecting words for cluster labeling. The first method (discriminative labeling) assumes the existence of a document hierarchy, either manually constructed and/or populated, or a hierarchy resulting from application of a hierarchical clustering algorithm. Using chi square tests of independence at each node in the hierarchy starting from the root, we determine a set of words that are equally likely to occur in any of the children of a current node. Such words are general for all of the sub-trees of a current node, and are excluded form the nodes below. The second method (non-discriminative labeling) selects words which both occur frequently in a cluster or effectively discriminate the given cluster from the other clusters. While the third method, using titles for labeling clusters, is based on the titles of the documents within each cluster to find the most representative words for the cluster.

The work in [25] briefly describe a technique to label clusters based on how many times a feature is used in a cluster. By utilizing this information, and also drawing on knowledge of the code, short titles are manually selected for the obtained clusters. Although labeling is performed manually, they point out that the automatically developed feature summary of each cluster makes the labeling process much easier. In 2001 Tzerpos et al. [26] emphasizes that a clustering algorithm should have certain features to make its output easier to comprehend. These features include bounded cluster cardinality, which ensures that any single cluster does not contain a very large number of entities, and effective cluster naming. They use a pattern based approach to recognizing familiar subsystem structures within large systems. The identified patterns are expected to occur in large systems with around 100 source files. The same pattern-based approach is used for cluster labeling. In 2003 Tonella et al. [27] describe the use of keywords within web pages to cluster and label similar pages. Both single words and a contiguous sequence of two words i.e., bigrams are considered as representative keywords of a webpage. Clustering as well as cluster labeling are carried out on the basis of keywords within a webpage. Cluster labels are ranked according to inverse keyword frequency

Chapter 3

Requirements

The goal of our project is to improve the quality of the document searching by clustering the documents and using the results to influence the search results. We propose to do the clustering in an iterative manner such that a hierarchy is developed between the different iterations. This would further improve the search results quality since the hierarchical results could result in a scoring mechanism with different weights for different levels of hierarchy.

Below, we summarize our tasks for the project:

- Hands-on with various big data technologies like Hadoop, Map Reduce, HDFS, Solr. See subsection B.0.10.
- Gaining expertise in Apache Mahout clustering algorithms. See subsection B.0.11, subsection B.0.12.
- Flat clustering of tweets and webpages using K-means, Streaming K-means and/or Fuzzy K-means. See subsection B.0.13, subsection B.0.14.
- Evaluating and optimizing clustering results using three metrics which includes Silhoutte scores, confusion matrix with labelled data, and human judgement. See chapter 6.
- Cluster label extraction from the clustering results. See subsection B.0.18
- Hierarchical Clustering of tweets and webpages. See subsection B.0.15
- Merging the results of various levels of hierarchy to help the scoring mechanism in order to improve the search results quality. See subsection B.0.15

Chapter 4

Design

Offline document clustering is shown in Figure 4.1. For a major part of the project we emphasize on offline document clustering and optimize the scores based on the clustering results. The input to document clustering are tweets and web pages in Avro format in HDFS. The input data is cleaned and pre-processed using various techniques discussed in reducing noise team's report. The yellow boxes in Figure 4.1 is developed as part of this project. The blue boxes represent our leverage of Mahout clustering algorithms and tools. For divisive hierarchical clustering we leveraged K-Means clustering of Mahout and iteratively calling those algorithms at various hierarchical levels. Finally, the output produced is in Avro format with schema presented in the Hadoop team's report. The clustering output is further ingested into HBase and pipeline is completed by indexing the clustering information into Solr fields.

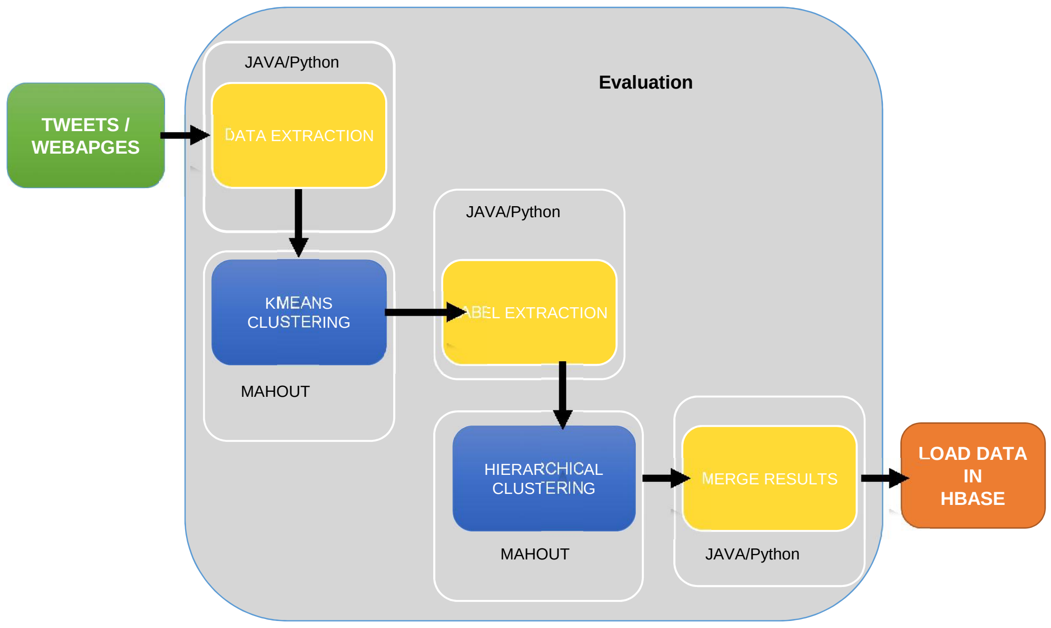


Figure 4.1: Offline Document Clustering in Hadoop Environment

4.0.8 Workflow

We adapt flat clustering methods for offline document clustering initially and then move on to hierarchical clustering methods. The Apache Mahout project provides access to scalable machine learning libraries which includes several clustering algorithms. Thus, in our project we leverage various clustering algorithms in Apache Mahout library. The input format to clustering algorithms is an *Avro file*. The output format of clustering is also an *Avro file*. Since we don't need any meta-data for clustering the actual input data format (Avro) will be converted to sequence files while providing input to Mahout clustering algorithms. For generating human readable representation of *sequence files* we use the clusterdumper tool in Mahout. The information from the output of the clusterdumper tool would be used to add two more fields to the Solr schema. For flat clustering, the final output from our team is the addition of two fields "Cluster ID" and "Cluster label". The overall workflow is presented in Figure 4.2

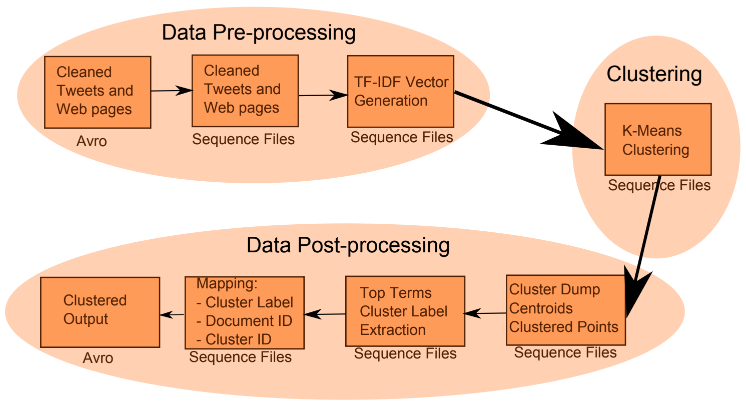


Figure 4.2: System architecture

4.0.9 Tools

For document clustering we use Apache Mahout clustering algorithms and relevant tools provided as part of the Mahout library. The document collections are saved in HDFS and the output of clustering is saved in HBase. The format of input and output files will be *sequence files*. KEA is used for labeling the clustering result. KEA is implemented in Java and is platform independent and an open-source software.

4.0.10 Programming Languages

Apache Mahout and HBase are written in Java. Thus, any driver programs we use to generate sequence files to interact with Mahout/HBase will be in Java. Further, the Python and bash scripting languages are used to perform post-processing operations and clustering evaluation.

4.0.11 Dependencies

The dependencies we had during our work progress and performance evaluation are listed below.

- Feature extraction team: This is a mutual dependency. We are dependent on the feature extraction team to provide optimized feature vectors that are necessary for improving clustering. Currently, the feature vectors extracted from sequence files are directly fed to Mahout algorithms. However, cluster results can be improved if important features are extracted and fed to Mahout algorithms. We collaborate/leverage the results from the feature extraction team to optimize the document clustering. We have used very basic feature extraction methods like pruning high frequency words, pruning very low frequency words, stop word removal and stemming to reducing the dictionary size.

- LDA team: As part of the output we provide appropriate cluster labels and evaluate those labels with the LDA team to ensure that relevant topics are clustered as expected. As an evaluation measure we have collaborated with LDA team and provided a cosine similarity matrix for documents within each cluster and for two collections ebola_S and Malaysia_Airlines_B tweet collections. This mutual evaluation helps to compare statistical document clustering with probabilistic document clustering based on topic modeling from LDA team. More details on evaluation can be found in LDA team report for project IDEAL. To aid the clustering evaluation with respect to statistical measures we have provided three evaluation metrics discussed in detail in chapter 6

- Reducing Noise team: We have obtained cleaned versions of the data from reducing noise team that helped us to improve the document quality and quality of the cluster results.

- Solr team: Helps us to provide feedback on scoring the Solr search results based on offline clustering.

- Hadoop team: While we work with large volumes of data, we seek help to configure the Hadoop environment and resolve issues related to developing map-reduce programs used in clustering.

Chapter 5

Implementation

5.0.12 Milestones and Deliverables

- Hadoop-ready driver program to run Mahout clustering algorithms on hadoop cluster with large collections.
- Flat clustering of tweets and webpages collections
- Hierarchical clustering of tweets and webpages collections
- Merging of the results of various levels of clustering
- Collecting Statistics on the runtimes
- Clustering quality evaluation using Silhoutte scores, confusion matrix and human judgement
- Integration of the cluster output into Solr fields to improve recall of relevant documents

5.0.13 Timeline

The implementation time line of the and member contributions are listed in the Table 5.1 in chronological order.

Weekly Reports	Task	Done by
1	Installation of SOLR on laptops.	Sujit, Rubasri, Hanna
1	Clustering literature review	Sujit, Rubasri, Hanna
2	Understanding of workflow	Sujit, Rubasri, Hanna
2	Mahout setup and integration with Lucene vector packages	Sujit
3	Carrot clustering in SOLR	Sujit
3	SOLR and Mahout integration	Sujit
3	Reorganization of previous week's report according to Report15 requirements	Sujit, Rubasri, Hanna
4	Downloading of webpages mentioned in tweets by using the script provided by the TA	Rubasri
4	Indexing of Webpages and tweets in SOLR	Sujit and Hanna
4	Exploration of clustering options in Mahout (Kmeans, Streaming Kmeans, Fuzzy Kmeans)	Sujit
5	Extraction of all the tweets from the CSV file using the python script to sequence file since Mahout requires sequence file as input.	Rubasri
5	Clustering of sample Tweets using Mahout	Rubasri
5	Clustering of sample Webpages using Mahout	Rubasri
6	Conversion of AVRO format to sequence file and extraction of the cluster results from Mahout.	Sujit
6	Identification of 100 relevant tweets for training data set and 100 other random tweets for the test data set to be provided to the Classification team.	Sujit
6	Crawling of webpages using Nutch in our local machine and cluster for small collections	Rubasri

8	Implemention of K-means clustering in Mahout and association of the results of clustering with the input data collection	Sujit, Rubasri
8	Extraction of the cluster labels (used the top most term) from the cluster results and association of the labels with each tweet	Rubasri
8	Modification of the AVRO schema to include the cluster id and cluster label for each of the tweets. We were able to get the output files in AVRO format which has the clustering results.	Sujit
9	Analysis of Mahout clustering algorithms (Streaming K means, fuzzy k means, K means)	Sujit
9	Crawling of webpages using Nutch in our local machine and cluster for big collections	Sujit
9	Automation of clustering using bash script	Sujit
9	Implementation of Hierarchical clustering	Rubasri
10	Merging of the results from different levels of hierarchical clustering	Rubasri
10	Implementation of KIA labelling	Hanna
10	Statistics for evaluation	Sujit
10	Clustering of cleaned webpages	Sujit
10	Automation of the hierarchical clustering process using bash script	Rubasri
Final	Clustering evaluation using Silhoutte scores, confusion matrix, and human judgement	Sujit
Final	Final project presentation and report	Sujit, Rubasri, Hanna

Table 5.1: Weekly status report

Chapter 6

Evaluation

We have chosen three metrics for evaluation which includes Silhoutte scores, confusion matrix, and human judgement. In the following sections we provide our results that includes evaluation for tweet and web page collections.

6.1 Silhoutte Scores

The goal of clustering is to ensure that the documents that are similar are clustered into their own cluster and documents that are dissimilar are clustered into separate clusters. Silhoutte scoring is a well known technique to evaluate cluster results when labelled data is not available [28]. Since clustering is an unsupervised learning we have chosen to evaluate our clustering results using Silhoutte scores which does not require labelled data.

After obtaining clustering results from the work flow described in previous chapter, we categorize the the documents into their own clusters using the document ID and cluster ID mapping. Further, we compute the dissimilarity ($a(i)$) of each data point i with respect to all other data points within the same cluster k. In addition, we compute the lowest average dissimilarity ($b(i)$) from data point i to any other cluster. Silhoutte coefficient for the data point i is computed as follows:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{6.1}$$

Silhoutte score for the document collection is the mean average of all coefficients computed for each of the data points. To efficiently compute the Silhoutte score we use scikit learn python package. A Silhoutte score of $+1$ represents that the documents are clustered with high quality, a score of -1 represents that the documents are clustered with poor quality. Normally, the Silhoutte score for text documents will be close to zero due to the sparsity of the documents (99%). For our evaluation we assume that the Silhoutte score of anything greater than zero is considered to be decent clustering result.

Table 6.1, 6.2, and 6.3 provide Silhoutte scores for small tweet collection, big tweet collection and web page collection respectively.

From the scores we interpret that for most of the collections the clustering results are good enough as the Silhoutte scores are greater than zero. As a future work we propose that the scores can be improved by performing more advanced feature selection, dimensionality reduction, and various other clustering procedures.

Data Set	***Silhoutte Score***
charlie_hebdo_S	0.0106168
ebola_S	0.00891114
Jan.25_S	0.112778
plane_crash_S	0.0219601
winter_storm_S	0.00836856
suicide_bomb_attack_S	0.0492852
election_S	0.00522204

Table 6.1: Silhoutte Scores for Small Tweet Collection

Data Set	***Silhoutte Score***
bomb_B	0.0114236
diabetes_B	0.014169
egypt_B	0.0778305
Malaysia_Airlines_B	0.0993336
shooting_B	0.00939293
storm_B	0.011786
tunisia_B	0.0310645

Table 6.2: Silhoutte Scores for Big Tweet Collection

6.2 Confusion Matrix

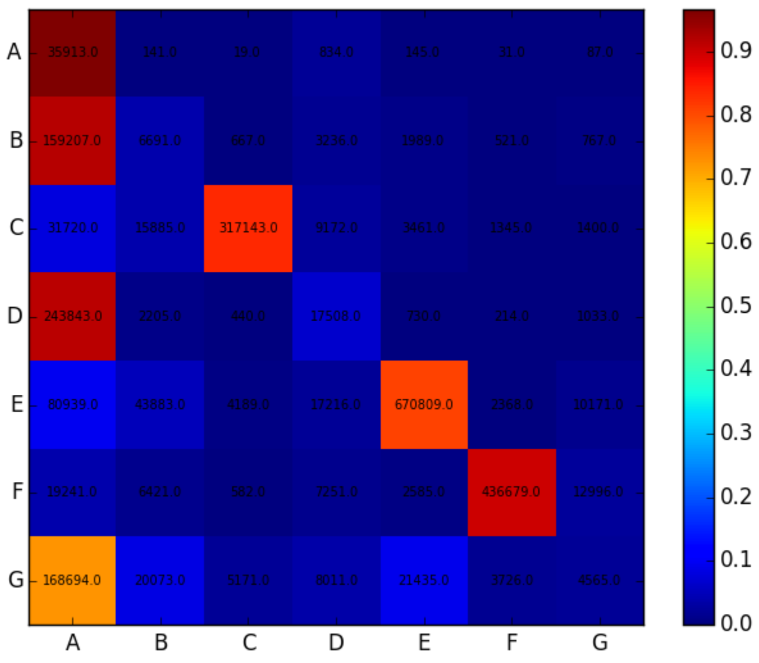
In our corpora we observed that document collections are already categorized into various event types. For example, ebola_S tweet collection mostly comprises tweets related to Ebola disease. Thus, we have concatenated data sets of small collections and big collections into single data set and performed clustering to evaluate the effectiveness of unsupervised clustering algorithm we have used.

We have concatenated data sets from small tweet collection which includes tweet collections from seven different event types: charlie_hebdo_S, ebola_S, Jan.25_S, plane_crash_S, winter_storm_S, suicide_bomb_attack_S, election_S and performed K-Means clustering on the concatenated data set. Since we now know the labels of each of the cluster we demonstrate the effectiveness of clustering procedure by building a confusion matrix as presented in Figure 6.1. The figure represent a heat map of number of documents in the cluster normalized by the total number of documents in the partitioned collection. The higher the intensity of heat map (blue to red) the larger the number of documents concentrated in that cluster.

Similarly, we have concatenated big data sets which includes tweet collections from seven different event types: bomb_B, diabetes_B, egypt_B, Malaysia_Airlines_B,

Data Set	***Silhoutte Score***
classification_small_00000_v2 (plane_crash_S)	0.0239099
classification_small_00001_v2 (plane_crash_S)	0.296624
clustering_large_00000_v1 (diabetes_B)	0.124263
clustering_large_00001_v1 (diabetes_B)	0.0284772
clustering_small_00000_v2 (ebola_S)	0.0407911
clustering_small_00001_v2 (ebola_S)	0.0163434
hadoop_small_00000 (egypt_B)	0.206282
hadoop_small_00001 (egypt_B)	0.264068
ner_small_00000_v2 (storm_B)	0.0237915
ner_small_00000_v2 (storm_B)	0.219972
noise_large_00000_v1 (shooting_B)	0.027601
noise_large_00000_v1 (shooting_B)	0.0505734
noise_large_00001_v1 (shooting_B)	0.0329083
noise_small_00000_v2 (charlie_hebdo_S)	0.0156003
social_00000_v2 (police)	0.0139787
solr_large_00000_v1 (tunisia_B)	0.467372
solr_large_00001_v1 (tunisia_B)	0.0242648
solr_small_00000_v2 (election_S)	0.0165125
solr_small_00001_v2 (election_S)	0.0639537

Table 6.3: Silhoutte Scores for Web page Collections



	A	B	C	D	E	F	G
A	35913.0	141.0	19.0	834.0	145.0	31.0	87.0
B	159207.0	6691.0	667.0	3236.0	1989.0	521.0	767.0
C	31720.0	15885.0	317143.0	9172.0	3461.0	1345.0	1400.0
D	243843.0	2205.0	440.0	17508.0	730.0	214.0	1033.0
E	80939.0	43883.0	4189.0	17216.0	670809.0	2368.0	10171.0
F	19241.0	6421.0	582.0	7251.0	2585.0	436679.0	12996.0
G	168694.0	20073.0	5171.0	8011.0	21435.0	3726.0	4565.0

Figure 6.1: Confusion Matrix for Concatenated Small Tweet Collection

shooting_B, storm_B, tunisia_B. Confusion matrix for concatenated big data set is shown in Figure 6.2

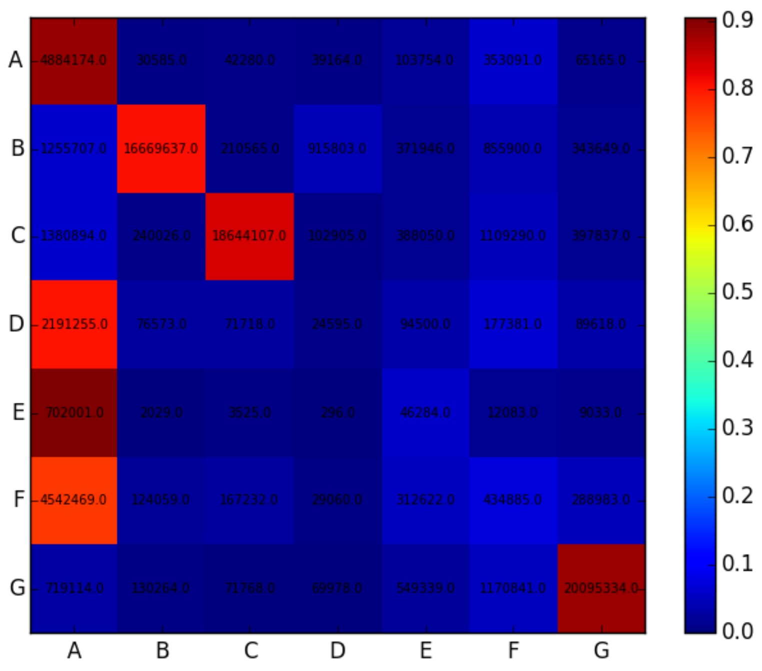


Figure 6.2: Confusion Matrix for Concatenated Big Tweet Collection

Unlike confusion matrix for classification, in the context of clustering we are interested in just homogeneous labeling than accurate labeling, i.e., we are interested only in the groupings of documents rather than the exact class which they belong to. Thus, instead of providing exact labels to the clusters obtained, we label the clusters ranging from A to G in both figures.

An interesting insight into the confusion matrix lead us that in both the data sets we observe that approximately 4 collections out of 7 collections are placed in the same cluster. After manually analyzing the data sets we concluded that most of the data sets that belong to single cluster have the event type bomb, shooting, tunisia, etc. Thus, we concluded that the clustering output is reasonable.

In addition to the confusion matrix, we have calcualated the Silhoutte scores for each of the concatenated data sets. Table 6.4 shows the scores obtained for the collections. Even if there is mis-classification as judged by confusion matrix we still see Silhoutte scores to be greater than zero, which further confirms our expectation that the tweets in those documents might be similar.

Data Set	***Silhoutte Score***
Small Tweet Collection	0.0190188
Big Tweet Collection	0.0166863

Table 6.4: Silhoutte Scores for Concatenated Tweet Collection

6.3 Human Judgement

A third metric we have chosen to evaluate clustering results is to compare the results of clustering and labeling method with that of human judgement. Due to time constraints we haven't evaluated final results of all of the document collections. In this section, we provide an evaluation of Ebola data set.

6.3.1 Clustering Result for Ebola Data Set

Since ebola_S is a small data set in our experiments we have found that an optimal number of clusters such data sets is 5. After this count even if we increase the number of clusters we observe that only a small fraction of documents (possibly outliers) are present in the clusters and thus are sparsely clustered. Table 6.5 shows the labeling information of the clustering result along with number of documents in each of the cluster.

Cluster ID	**Cluster Label**	**Number of Tweets**
1	Death	28361
2	Doctor	32527
3	Obama	25108
4	Ebola	274040
5	Drug	20123

Table 6.5: Clustering result for ebola_S data set

By comparing the cluster results and manually analyzing random samples of the documents in each of the cluster we present a summary table 6.6

6.4 Clustering Statistics

Table 6.7 provides details on the runtime of clustering procedures. For small tweet collections clustering takes approximately 5.5 minutes while big tweet collections take around 16 minutes. Web page collections are small and take around 5 minutes to converge. The sparsity index column represents the sparsity ($\frac{\#\_of\_zeros}{size\_of\_matrix}$) of the TF-IDF matrix after performing feature selection using high frequency word pruning, stemming, lemmatization. The sparsity can be further reduced by performing feature transformation techniques like Latent Semantic Analysis (LSA) or Singular Value Decomposition (SVD) on the data matrix. However, such feature transformation techniques are not scalable to large data. Typically, in our evaluation Apache Mahout SVD took 10 hours to reduce the dimensions of concatenated small tweet collection previously mentioned.

Human Label	***Sample Tweets***
Death	Ebola kills fourth victim in Nigeria The death tollfrom the Ebola outbreak in Nigeria has risen to four whi, RT Dont be victim 827 Ebola death toll rises to 826, RT Ebola outbreak now believed to have infected 2127 people killed 1145 health officials say, RT Two people in have died after drinking salt water which was rumoured to be protective against
Doctors	US doctor stricken with the deadly Ebola virus while in Liberia and brought to the US for treatment in a speci, Moscow doctors suspect that a Nigerian man might have
Politics	RT For real Obama orders Ebola screening of Mahamaother African Leaders meeting him at USAfrica Summit Patrick Sawyer was sent by powerful people to spread Ebola to Nigeria Fani Kayode has reacted, RT The Economist explains why Ebola wont become a pandemic View video via, Obama Calls Ellen Commits to fight amp WAfrica
Symptoms	How is this Ebola virus transmitted, RT Ebola symptoms can take 2 21 days to show It usually start in the form of malaria or cold followed by Fever Diarrhoea Ebola virus forces Sierra Leone and Liberia to withdraw from Nanjing Youth Olympics
Drugs	Ebola FG Okays Experimental Drug Developed By Nigerian To Treat, RT Drugs manufactured in tobacco plants being tested against Ebola other diseases, Tobacco plants prove useful in Ebola drug production EBOLA Western drugs firms have not tried to findvaccine because virus only affects Africans

Table 6.6: Human judgement matches that of clustering results

Data Set	***Dictionary Size***	***Sparsity Index***	***Time (Minutes)***
charlie_hebdo_S	13452	99.85911799	5.39
ebola_S	25648	99.89573479	6.37
Jan.25_S	17639	99.91855227	5.45
plane_crash_S	16725	99.86282832	5.82
winter_storm_S	23717	99.86906727	6.28
suicide_bomb_attack_S	3748	99.72621082	5.45
election_S	59643	99.92313503	6.93
Concat_S	100548	99.92509434	NA
bomb_B	508347	99.85911799	14.68
diabetes_B	224233	99.89573479	15.32
egypt_B	159347	99.91855227	8.87
Malaysia_Airlines_B	18498	99.86282832	15.6
shooting_B	600305	99.86906727	16.6
storm_B	516101	99.72621082	16.22
tunisia_B	175966	99.92313503	7.34
Concat_B	1559466	99.9460551	NA
classification_small_00000_v2	3839	97.39412057	5.1
classification_small_00001_v2	536	97.62272621	5.1
clustering_large_00000_v1	11374	99.18097229	5.3
clustering_large_00001_v1	15593	99.23332298	5.2
clustering_small_00000_v2	385	97.96474954	4.9
clustering_small_00001_v2	384	96.78017164	5.1
hadoop_small_00000	4387	99.17565805	5.4
hadoop_small_00001	4387	99.17561654	5.1

Table 6.7: Clustering statistics for various document collections

Chapter 7

Conclusion and Future work

7.0.1 Conclusion

In our project, we performed flat clustering on the various input data sets using Mahout K-means clustering. In the Mahout K-means algorithm, the initial centroids of the cluster are chosen as random and it is required to specify the number of clusters. Using empirical analysis we found that the best number of clusters for the small data set is 5 and large data set is 10. In order to further improve the search quality results, we performed hierarchical clustering on the input data sets by further clustering the results obtained in flat clustering. We chose the top terms present in the cluster as the cluster label. These top terms are identified from the K-means cluster dump results using Mahout tools. Although top terms are not the best way to label the clusters they work well for tweet collection (short text).

In order to verify the effectiveness of the clustering, we have evaluated the clustering results using Silhouette scores, confusion matrix and human judgement. Silhouette scores measure how similar the documents are within each cluster and how dissimilar the documents are in different clusters. We obtained positive Silhouette scores for all of the data sets which shows that the quality of the clustering is commendatory. In addition to Silhouette scores, confusion matrix was also used to evaluate the quality of the clustering where in we used K-means algorithm with various tunables. Due to high sparsity in the data set the Silhoutte scores are low (close to zero). However, feature transformation methods like Latent Semantic Analysis (LSA) can be applied to transform the data set into lower dimensional space decreasing the sparsity and increasing the Silhoutte scores. We also found that the Silhouette scores of web pages are higher than the tweets mainly because of the length of the web pages and sparsity index compared to that of tweets.

7.0.2 Future Work

This work can be extended to perform incremental way of clustering as new data is streamed in. Also, identifying appropriate number of clusters for a particular data set through some kind of evaluation instead of empirical analysis could also be a possible extension. In addition to labeling the clusters using the top terms, methods like Wikipedia cluster labeling which identifies top terms from the cluster results and search in Wikipedia for best possible titles that match top terms can be an enhancement to this work.

Appendices

Appendix A

User manual

A.0.3 Pre-requisites

Users are recommended to have following environment setup prior to performing data clustering as described in the later subsections.

- Access to Hadoop cluster (for example, hadoop.dlib.vt.edu)
- Access to data loaded in HDFS (cleaned version of data is optional but highly recommended)
- Data loaded in HDFS is in Avro format as specified in schema reported Hadoop team
- Installed Mahout and Hadoop utilities and configurations are setup properly

A.0.4 Data preparation

Input data in Avro format does not work with Mahout clustering algorithms. Hence, as a first step the input must be converted to sequence files with “key” as document ID and “value” as cleaned text. Following commands convert Avro files to sequence files.

```
$ hadoop fs -copyToLocal /user/cs5604s15_noise/TWEETS_CLEAN/
    ebola_S .

$ java -jar AvroToSequenceFilesCleanedTweetDocId.jar ./ebola_S/
    part-m-00000.avro ./ebola_S/part-m-00000.seq

$ hadoop fs -mkdir cleaned_tweets_docid

$ hadoop fs -copyFromLocal ./ebola_S  cleaned_tweets_docid/
```

A.0.5 Data Clustering

Once input format is converted from Avro to sequence files, clustering procedure involves executing a bash script as shown below.

```
$ ./clustering_cs5604s15.sh <input> <output> 2>&1 | tee
    logfile_name.log

# Note that <input> and <output> arguments above will be HDFS
    paths
# follow the instructions if you want to change the clustering
    procedure (kmeans, streaming kmeans, fuzzy kmeans)
```

A.0.6 Cluster Labeling

Once clustering is done, we need to copy the cluster output to local file system before executing cluster labeling

```
# ensure clustering output is present
$ hadoop fs -ls <output>/output-kmeans

# copy cluster output to local fs
$ hadoop fs -copyToLocal <output>/output-kmeans .

# execute cluster labeling
$ java -jar labelWithIDAvroOut.jar <output folder name> <output
    file name>

# final output will be located in:
$ ls -al <output folder name>/output-kmeans/<output file name>.
    avro
```

A.0.7 Cluster output

As a final step, copy the cluster output in local filesystem to HDFS. This output will be of the Avro schema presented in Hadoop team's report. Once the output is uploaded to HDFS, execute Hbase loading scripts as mentioned in Hadoop team's report.

A.0.8 Hierarchical Clustering

In order to do hierarchical Clustering on the flat clustering results, the following steps have to be followed

```
# ensure input sequence file is present in local fs
$ ls <input>

# Split the dataset according to the flat clustering results
$ java -jar HClustering.jar <input sequence file folder> <output
     of cluster results folder> <dataset>

# Copy the folder to HDFS
```

```
$ hadoop fs −put <output of cluster results folder> <level 2
    input folder>


# Execute K−means clustering on each of the sequence file
    generated
$ ./clustering_cs5604s15.sh <level 2 input folder>s/<dataset> <
    level 2 output folder>/<dataset> 2>&1 | tee logfile_name.log

# ensure clustering output is present
$ hadoop fs −ls <level 2 output folder>/<dataset>/output−kmeans

# copy cluster output to local fs
$ hadoop fs −copyToLocal <level 2 output folder>s/<dataset>/
    output−kmeans .

# execute cluster labeling on each of the results generated
$ java −jar labelWithIDAvroOut.jar <level 2 output folder>/<
    dataset> <dataset>



# merge the results of both the levels of clustering
$java −jar merge.jar <level 1 output folder> <level 2 output
    folder>
```

A.0.9 Working with many collections

To avoid repetitive tasks for many collections, users can look into scripts that automate most of the clustering process described in above sections.

```
# perform clustering
$ ./tweet_clustering.sh < clusterinput_big.txt

# perform cluster labeling and generating output in Avro format
$ ./putkmeans.sh < clusterinput_big.txt

# Hierarchical clustering and merging (datasets have to be
    edited in the script)
$ ./hierarchical_clustering.sh
```

Appendix B

Developers manual

This section details implementation of our project and provides further information for a developer who is interested in extending this work.

In this manual it is assumed that the development environment is configured as following:

Operating System: Ubuntu 14.04 Linux
CPU Architecture: x86_64
CPU cores: 4
Memory: 4GB
Java version: 1.7.0_75
JVM: 64-bit Server
IDE: Eclipse (Luna with Maven repositories indexed)

B.0.10 Solr Installation

Solr is an open source enterprise search platform. Its major features include full-text search, hit highlighting, faceted search, dynamic clustering, database integration, and rich document (e.g., Word, PDF) handling.

Installation of Solr can be done either on a local machine or server environment. Developers are recommended to go through the quick start tutorial [22] for installing Solr and indexing data into Solr. To index customized data ensure that the schema.xml [23] is updated with necessary fields and values.

After successful installation http://localhost:8983/solr/ shows as in Figure B.1.

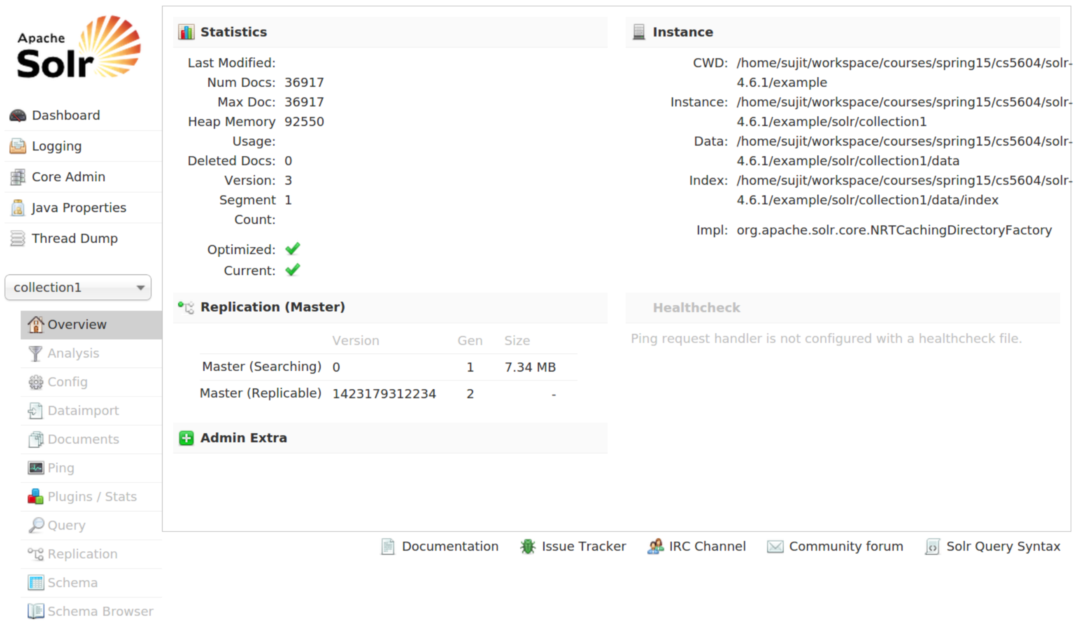
B.0.11 Mahout Installation

– Download latest source code from Apache Mahout git repository:

```
$ git clone git://git.apache.org/mahout.git
```

– Ensure Maven is installed:

```
$ sudo apt-get install maven
```



Statistics

```
     Last Modified:
         Num Docs:  36917
          Max Doc:  36917
      Heap Memory   92550
           Usage:
     Deleted Docs:  0
          Version:  3
          Segment   1
           Count:
        Optimized:  ✓
          Current:  ✓
```

Instance

```
       CWD:  /home/sujit/workspace/courses/spring15/cs5604/solr-
             4.6.1/example
  Instance:  /home/sujit/workspace/courses/spring15/cs5604/solr-
             4.6.1/example/solr/collection1
      Data:  /home/sujit/workspace/courses/spring15/cs5604/solr-
             4.6.1/example/solr/collection1/data
     Index:  /home/sujit/workspace/courses/spring15/cs5604/solr-
             4.6.1/example/solr/collection1/data/index
      Impl:  org.apache.solr.core.NRTCachingDirectoryFactory
```

Replication (Master)

	Version	Gen	Size
Master (Searching)	0	1	7.34 MB
Master (Replicable)	1423179312234	2	-

Admin Extra

Healthcheck

Ping request handler is not configured with a healthcheck file.

Figure B.1: Snapshot of Solr user interface

– Compile and install Mahout:

```
$ cd mahout
$ mvn -DskipTests=true clean install
```

Note: Several problems might arise during installation.
1) Java JDK is not installed or *JAVA_HOME* environment variable is not set properly
2) Some compilation issues like scala installation failures
To avoid such issues ensure Java JDK is installed properly. In addition, Maven version in ubuntu repositories may not be compatible with the Mahout source. Use Maven v3.2.5 from
official Maven download page [24] and installation will be successful.

B.0.12 Solr and Mahout Integration

After successfully installing Solr and Mahout, document index output from Solr can be used to generate feature vectors in a format understandable by Mahout.

Solr index data is available in:
<solr dir>/solr/collection1/data/index

Use Mahout lucene.vector package to get term vectors:

```
$ bin/mahout lucene.vector \
--dir /solr/collection1/data/index \
--field content \
--dictOut dict.txt \
--output dict.out
```

In order to ensure that the term vectors can be extracted using the lucene.vector package the Solr schema.xml should be modified as following:

```
<field name=‘‘content” type=‘‘text general” indexed=‘‘true”
    stored=‘‘true” termVectors=‘‘true”/>
```

B.0.13 Clustering Webpages

As a first step, webpages are extracted from the URLs in the tweets using the script provided by the RA (tweet_URL_archivingFile.py). The input file is a small tweet collection focusing on the Ebola disease collection (z224t.csv) provided in scholar under Resources/Data. The threshold in the script was set to 1 to extract the contents from all the URLs that appeared in the tweets at least once. Upon running the script, 302 URLs were extracted as text files. The steps involved in clustering the webpages are described below.

A new directory is created in HDFS for storing the webpages collection

```
 $ hadoop fs -mkdir webpages
```

The webpages collection is put in the HDFS using the following command.

```
$ hadoop fs -put Webpages webpage/
```

Since Mahout accepts only *Sequence file* as input format, the extracted webpages which are in the form of text files are converted to a sequence file using the following command.

```
$ mahout seqdirectory -i webpages/Webpages/ \
    -o webpages/WebpagesSeq -xm sequential
```

“-xm sequential” is given to specify that the conversion has to be done sequentially. If it is omitted, the conversion would be done in the form of mapreduce jobs.

The TFIDF vectors from the sequence file are generated using the following command.

```
$ mahout seq2parse -i webpages/WebpagesSeq \
  -o webpages/WebpagesVectors
```

Canopy clustering is done before K-means clustering to guess the best value of K. Output of this stage becomes the input of K-means clustering.

```
$ mahout canopy \
  -i webpages/WebpagesVectors/tfidf-vectors/ \
  -o webpages/WebpagesCentroids -t1 500 -t2 250
```

Each canopy cluster is represented by two circles. The radius of the outer circle is T1 and the radius of the inner circle is T2. The options "-t1" and "-t2" specifies the T1 and T2 radius thresholds respectively. The option "-dm" specifies which distance measure to use (default is SquaredEuclidean).

The results of canopy clustering can be dumped to a text file using the following command. This step is optional.

```
$ mahout clusterdump -dt sequencefile -d \
  webpages/WebpagesVectors/dictionary.file-* \
  -i webpages/WebpagesCentroids/clusters-0-final \
  -o webpages_report.txt
```

K-means clustering is done using the following command.

```
$ mahout kmeans \
  -i webpages/WebpagesVectors/tfidf-vectors \
  -c webpages/WebpagesCentroids/clusters-0-final \
  -o webpages/WebpagesClusters \
  -cd 1.0 -x 20 -cl
```

The option "-cd" specifies the convergence delta. The default is 0.5. "-x" specifies the maximum number of iterations and "-cl" specifies that K-means clustering has to be done after the canopy clustering.

Finally, the results of K-means clustering is dumped using the following command.

```
$ mahout clusterdump -dt sequencefile -d \
  webpages/WebpagesVectors/dictionary.file-* \
  -i webpages/WebpagesClusters/clusters-2-final -o \
  webpages_report_kmeans.txt
```

B.0.14 Clustering tweets

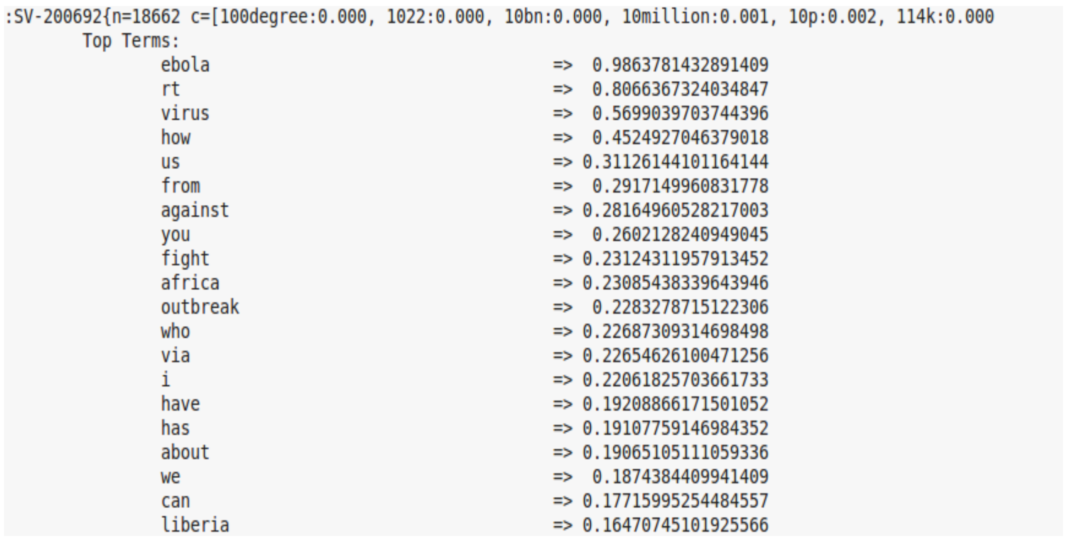
With Avro ebola_S_AVRO/part-m-00000.avro file we were able to extract the sequence files that can be used to provide input to the Mahout clustering algorithms. The code for converting Avro data file to sequence file is provided at the end of this section. The current implementation is only with sequential reading and writing. We plan to implement Map-Reduce conversion with further help from the Hadoop team. The sequence file is generated using two fields in the Avro schema - "tweet_id" and "text_clean". For clustering we need not use all the meta-data provided in the Avro file. In addition, most of the fields in the Avro schema are unused.

We have used cleaned tweets (ebola_S) data set to perform clustering. We have performed three types of clustering (1) streaming k-means (2) fuzzy k-means and (3) K-means. To avoid manual command line input each time to execute clustering algorithms we have developed a bash script that performs most of the clustering steps. The script is still evolving in terms of feature additions. A working script is appended at the end of the next section.

Fuzzy K-means:

Figure B.2 shows an example output of fuzzy k-means cluster dump. As Fuzzy K-Means is a soft clustering algorithm, each document in the cluster can be in multiple clusters.

From the observed output we have identified that alphabet "i" is appearing as a top term in the cluster output. This is expected since we are using uncleaned data. This is reported to the reducing noise team.



```
:SV-200692{n=18662 c=[100degree:0.000, 1022:0.000, 10bn:0.000, 10million:0.001, 10p:0.002, 114k:0.000
        Top Terms:
                ebola                                   =>  0.9863781432891409
                rt                                      =>  0.8066367324034847
                virus                                   =>  0.5699039703744396
                how                                     =>  0.4524927046379018
                us                                      => 0.31126144101164144
                from                                    =>  0.2917149960831778
                against                                 => 0.28164960528217003
                you                                     =>  0.2602128240949045
                fight                                   => 0.23124311957913452
                africa                                  => 0.23085438339643946
                outbreak                                =>  0.2283278715122306
                who                                     => 0.22687309314698498
                via                                     => 0.22654626100471256
                i                                       => 0.22061825703661733
                have                                    => 0.19208866171501052
                has                                     => 0.19107759146984352
                about                                   => 0.19065105111059336
                we                                      =>  0.1874384409941409
                can                                     => 0.17715995254484557
                liberia                                 => 0.16470745101925566
```

Figure B.2: Cluster dump output for fuzzy k-means

Streaming K-Means:

Choosing approximate cluster centroids using canopy clustering and further using k-means to provide cluster output is inefficient and the hadoop cluster might take long time to process the jobs. This is especially due to the squared euclidean distance measurement between each data point in the collection. When the collection is too large clustering take long time to converge. Streaming k-means overcomes this disadvantage. It has two steps - (1) streaming step (2) ball k-means step. In streaming step, the algorithm passes through all the data points once and computes approximate number of centroids. In the ball k-means step, an optimized algorithm is used to compute clusters efficiently and accurately compared to conventional k-means algorithm.

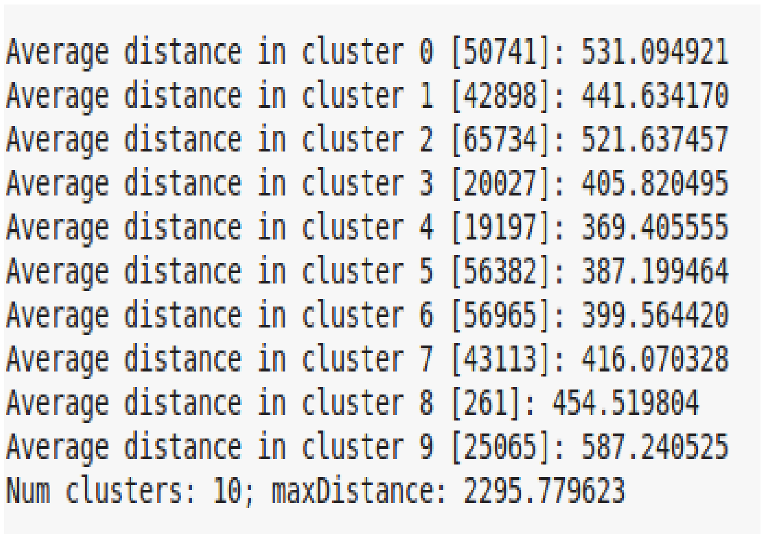
The command used in streaming k-means is as listed below:

```

$ mahout streamingkmeans --numClusters 10 \
  -i /user/cs5604s15_cluster/tweets_S_mar28/ebola_vectors/tfidf-vectors \
  -o /user/cs5604s15_cluster/tweets_S_mar28/ebola_vectors/streamingkmeans \
  -km 2000
```

In our further evaluation, we have generated a tiny collection with just 1000 tweets to ensure that our clustering flow is correct. We have extracted 1000 tweets from the Avro file and generated a sequence file in the format understandable by Mahout. Further, we have used canopy and k-means algorithms (similar to webpage clustering) to perform clustering on this tiny data set.

In the Figure B.3, we present brief statistics while performing Streaming K-Means on cleaned ebola_S tweet data set. The statistics are collected using the "qualcluster" tool in the Mahout library. The statistics indicate that the choice of tunable parameters are reasonably performing well with streaming k-means. The average distance in all of the clusters is around 400 while maximum distance from centroid to farthest point is 2295. This difference is expected as the k-means algorithm is susceptible to outliers. In our future evaluation, we attempt to identify such outliers and filter them out during pre-processing stages.



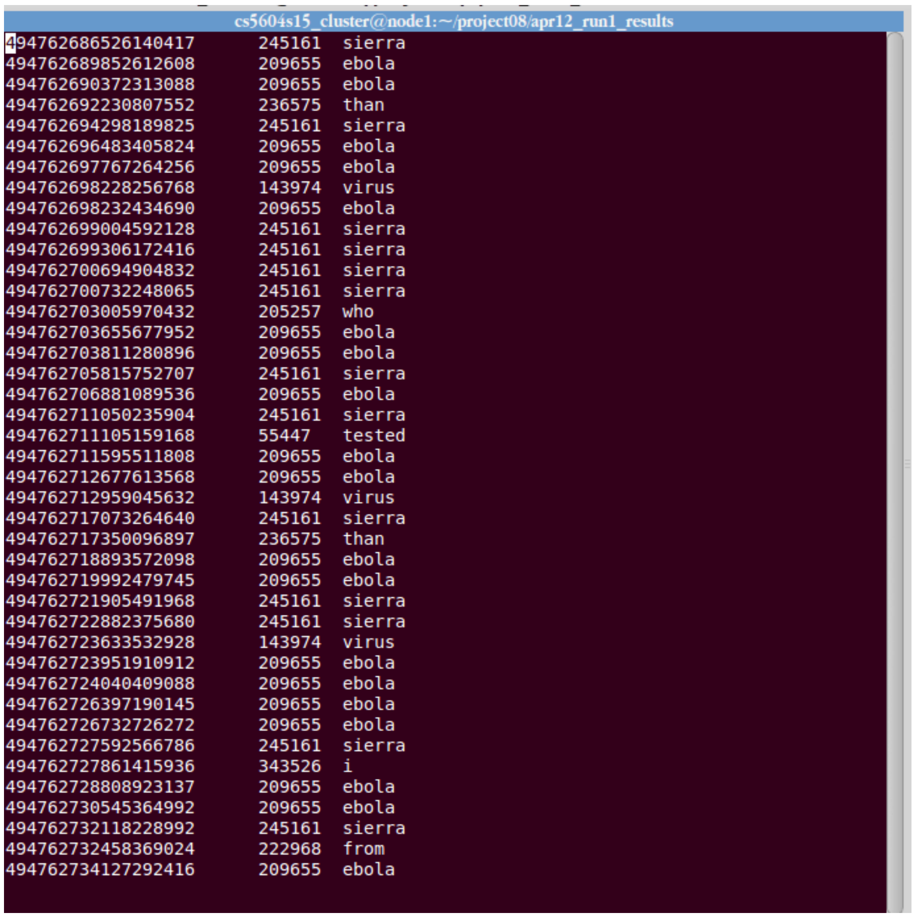
```
Average distance in cluster 0 [50741]: 531.094921
Average distance in cluster 1 [42898]: 441.634170
Average distance in cluster 2 [65734]: 521.637457
Average distance in cluster 3 [20027]: 405.820495
Average distance in cluster 4 [19197]: 369.405555
Average distance in cluster 5 [56382]: 387.199464
Average distance in cluster 6 [56965]: 399.564420
Average distance in cluster 7 [43113]: 416.070328
Average distance in cluster 8 [261]: 454.519804
Average distance in cluster 9 [25065]: 587.240525
Num clusters: 10; maxDistance: 2295.779623
```

Figure B.3: Cluster dump output for streaming k-means

The results are interesting. Out of 5 clusters (k in k-means is set to 5) we have accumulated about 310 tweets in cluster-1, about 684 tweets in cluster-2 and the remaining tweets in small clusters. These big clusters has top terms: cluster-1 - "sierra", "leone", "emergency", "declares", "wabah" which roughly represents the affected places of ebola disease. cluster-2: "ebola", "http", "t.co" which represents the disease itself. cluster-3: "world", "alert", "fears" which represents general terms in the tweets. Due the noise involved we are observing some unrelated top terms such as "http" and non-english characters. We have reported this to the reducing noise team and has corrected the data set with cleaned version.

K-means:

We performed K-means clustering on the whole of our small collection, Ebola data set. K-means provided by the Mahout package is a hard clustering algorithm. Although it is slower than streaming K-means, it produces clustered points also as an output. This output is essential in order to obtain the cluster IDs to associate with each tweet. We tried to modify streaming K-means and Fuzzy K-means to output the clustered points as well. But Mahout is not very flexible and it required us to modify the entire algorithm. We felt the effort required is not worth the speed up we got. Considering the fact that we are doing offline document clustering, the inefficiency of K-means clustering can be tolerated and hence we resorted to K-means clustering at this point. We have reported this to Apache Mahout forums for further evaluation - https://issues.apache.org/jira/browse/MAHOUT-1698.



cs5604s15_cluster@node1:~/project08/apr12_run1_results

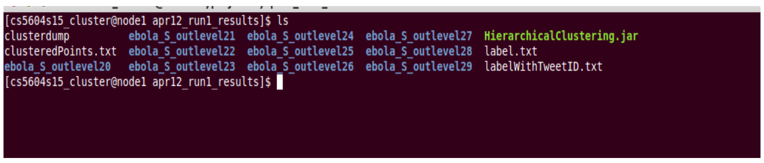
```
494762686526140417       245161  sierra
494762689852612608       209655  ebola
494762690372313088       209655  ebola
494762692230807552       236575  than
494762694298189825       245161  sierra
494762696483405824       209655  ebola
494762697767264256       209655  ebola
494762698228256768       143974  virus
494762698232434690       209655  ebola
494762699004592128       245161  sierra
494762699306172416       245161  sierra
494762700694904832       245161  sierra
494762700732248065       245161  sierra
494762703005970432       205257  who
494762703655677952       209655  ebola
494762703811280896       209655  ebola
494762705815752707       245161  sierra
494762706881089536       209655  ebola
494762711050235904       245161  sierra
494762711105159168       55447   tested
494762711595511808       209655  ebola
494762712677613568       209655  ebola
494762712959045632       143974  virus
494762717073264640       245161  sierra
494762717350096897       236575  than
494762718893572098       209655  ebola
494762719992479745       209655  ebola
494762721905491968       245161  sierra
494762722882375680       245161  sierra
494762723633532928       143974  virus
494762723951910912       209655  ebola
494762724040409088       209655  ebola
494762726397190145       209655  ebola
494762726732726272       209655  ebola
494762727592566786       245161  sierra
494762727861415936       343526  i
494762728808923137       209655  ebola
494762730545364992       209655  ebola
494762732118228992       245161  sierra
494762732458369024       222968  from
494762734127292416       209655  ebola
```

Figure B.4: Labeled output for K-Means clustering

Figure B.4 shows the clustering results. The first column is the tweet ID, second column is the cluster ID and the third column is the cluster label. We used Python to extract the cluster ID and the cluster label associated with each of the clusters from the cluster dump output. Another Python program would extract the tweet IDs and the cluster IDs associated with each of the tweets and use the label output obtained from the previous step to associate each of the tweets with the cluster it belongs too. We then modified the AVRO schema with the results obtained to produce an Avro output file with cluster results. The Avro output for ebola data set can be found in the cluster at *user/cs5604s15_cluster/clustered_tweets/ebola_S*

B.0.15 Hierarchical Clustering

Mahout does not have any inbuilt packages for hierarchical clustering. We used the cluster output we got from applying Mahout clustering algorithms on the cleaned dataset to separate the input sequence file into several sequence files. Thus each new sequence file generated represents a cluster. The figure below shows the sequence file directories generated.



```
[cs5604s15_cluster@node1 apr12_run1_results]$ ls
clusterdump          ebola_S_outlevel21  ebola_S_outlevel24  ebola_S_outlevel27  HierarchicalClustering.jar
clusteredPoints.txt  ebola_S_outlevel22  ebola_S_outlevel25  ebola_S_outlevel28  label.txt
ebola_S_outlevel20   ebola_S_outlevel23  ebola_S_outlevel26  ebola_S_outlevel29  labelWithTweetID.txt
[cs5604s15_cluster@node1 apr12_run1_results]$ 
```

We iteratively applied the Mahout K-means clustering algorithm on each of these clusters(sequence files) to obtain another level of hierarchy in clustering of documents. The results obtained are merged with the initial clustering output. These are then converted to Avro format. We have used Python to parse the clustering results obtained from Mahout to create a text file with tweet IDs and cluster labels. The final merged output is also a text file. We then converted it to Avro format.

B.0.16 Clustering small collection of tweets

We have concatenated all of the small collections (tweets) and attempted to cluster all of them at once using the K-Means algorithm. The results we expected are segregating documents into each of their clusters. We have used charlie_hebdo_S, ebola_S, election_S, plane_crash_S, suicide_bomb_attack_S, winter_storm_S small tweet collection to perform clustering and labeling. Since we are using terms with one word we obtained the following labels:

Cluster ID	Cluster Label
1822952	Plane
1969538	Winter
1899881	I
2048395	Election
1018103	Reelection
2079107	Storm
134706	did
1265069	My
1612349	We
1609831	General

As we can observe from the cluster labels the noise is still present and words like My, We, I appear frequently which contributes heavily to top terms. We have reported this observation to noise reducing team to further optimize the noise reduction. In addition, we have used feature selection to reduce such occurrences of the stop words.

B.0.17 Code Listings

Following is the Java code to extract sequence files from Avro input. The generated sequence file has "tweet id" as Key and "tweet text" as Value.

```java

package edu.vt.cs5604s15.clustering;

import java.io.File;
import java.net.URI;

import org.apache.avro.file.DataFileReader;
import org.apache.avro.io.DatumReader;
import org.apache.avro.specific.SpecificDatumReader;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class AvroToSequenceFiles extends Configured implements
    Tool {

  public int run(String[] args) throws Exception {
    File file = new File("/home/sujit/workspace/cs5604/
    ebola_S_AVRO/part-m-00000.avro");
    DatumReader<sqoop_import_z_224> datumReader = new
    SpecificDatumReader<sqoop_import_z_224>(sqoop_import_z_224.
    class);
    DataFileReader<sqoop_import_z_224> dataFileReader = new
    DataFileReader<sqoop_import_z_224>(file, datumReader);
    sqoop_import_z_224 data = new sqoop_import_z_224();

    String uri = "/home/sujit/workspace/cs5604/ebola_S_AVRO/
    tiny_ebola/ebola_tiny.seq";
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    Path path = new Path(uri);
    Text key = new Text();
    Text value = new Text();
    SequenceFile.Writer writer = null;

    writer = SequenceFile.createWriter(fs, conf, path, key.
    getClass(), value.getClass());

    System.out.println("Prepare to wait...");
    while (dataFileReader.hasNext()) {
      data = dataFileReader.next();
      key.set(data.getId().toString());
      value.set(data.getText().toString());
      writer.append(key, value);
      //System.out.println(key + " : " + value);
    }
    dataFileReader.close();
    IOUtils.closeStream(writer);
    System.out.println("Done");
    return 0;
  }

  public static void main(String[] args) {
    int result = 0;
    System.out.println("Starting Avro to Sequence file generation");
    try {
      result = ToolRunner.run(new Configuration(), new AvroToSequenceFiles(), args);
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.exit(result);
  }
}
```

Following code is used to perform clustering job on Hadoop cluster. The input to clustering_cs5604s15.sh script is HDFS sequence file generated from Avro output in above code listing. The output of clustering as of now is statistics in case of the streaming k-means and cluster dump with top terms in case of the fuzzy k-means. Our implementation for the cluster labeling is still under progress. Once cluster labeling is complete the final output will be produced in Avro format.

```bash
#!/bin/bash

# Author: Sujit Thumma
# inspired from https://github.com/apache/mahout/blob/master/examples/bin/cluster-reuters.sh
# Clustering job for CS5604 class

### Usage ###
if [ "$1" == "" ] || [ "$1" == "--help" ] || [ "$1" == "--?" ]; then
    echo "This script clusters tweet and webpage data in HDFS"
    echo "TODO: more help on usage"
    exit
fi

### change directory to script path ###
SCRIPT_PATH=${0%/*}
echo "$SCRIPT_PATH"
if [ "$0" != "$SCRIPT_PATH" ] && [ "$SCRIPT_PATH" != "" ]; then
        cd $SCRIPT_PATH
fi

### set environment variables ###
MAHOUT=mahout
HADOOP=hadoop

### check for required packages ###
#if [ ! -e $MAHOUT ]; then
```

```bash
#     echo "Can't find mahout driver in $MAHOUT, cwd `pwd`, exiting.."
#     exit 1
#fi

algorithm=( streamingkmeans fuzzykmeans )
if [ -n "$3" ]; then
  choice=$1
else
  echo "Please select a number to choose the corresponding clustering algorithm"
  echo "1. ${algorithm[0]} clustering"
  echo "2. ${algorithm[1]} clustering"
  read -p "Enter your choice : " choice
fi

echo "ok. You chose $choice and we'll use ${algorithm[$choice-1]} Clustering"
clustertype=${algorithm[$choice-1]}

### check for input and output directories ###
INPUT_DIR=$1
OUTPUT_DIR=$2

$HADOOP fs -ls $INPUT_DIR
if [ $? -eq 0 ]; then
    echo ""
else
    echo "hadoop fs -ls $INPUT_DIR failed"
    exit 1
fi

$HADOOP fs -mkdir $OUTPUT_DIR
if [ $? -eq 0 ]; then
    echo "creating directory $OUTPUT_DIR"
else
    echo "hadoop fs -mkdir $OUTPUT_DIR failed"
    echo "possible reasons- directory not empty"
    exit 1
fi

### get input avro files and convert to sequence file ###
# as of now we have all the sequence files ready. Ignore till final submission
# write another script for this when cleaned data arrives

### perform mahout clustering and output results ###
if [ "x$clustertype" == "xstreamingkmeans" ]; then
    echo "Step1: seq2sparse" && \
  $MAHOUT seq2sparse \
    -i ${INPUT_DIR}/input-seqdir/ \
    -o ${OUTPUT_DIR}/output-seqdir-sparse-streamingkmeans -ow --
   maxDFPercent 85 --namedVector \
  && \
  echo "Step2: Mahout streamingkmeans" && \
  $MAHOUT streamingkmeans \
    -i ${OUTPUT_DIR}/output-seqdir-sparse-streamingkmeans/tfidf-
   vectors/ \
    --tempDir ${OUTPUT_DIR}/tmp \
    -o ${OUTPUT_DIR}/output-streamingkmeans \
    -sc org.apache.mahout.math.neighborhood.FastProjectionSearch
    \
    -dm org.apache.mahout.common.distance.CosineDistanceMeasure
   \
    -k 10 -km 130 -ow \
  && \
  echo "Step3: qualcluster for streaming kmeans" && \
  $MAHOUT qualcluster \
    -i ${OUTPUT_DIR}/output-seqdir-sparse-streamingkmeans/tfidf-
   vectors/part-r-00000 \
    -c ${OUTPUT_DIR}/output-streamingkmeans/part-r-00000    \
    -o ${OUTPUT_DIR}/output-streamingkmeans-cluster-distance.csv
    \
    && \
  cat ${OUTPUT_DIR}/output-streamingkmeans-cluster-distance.csv
  echo "Check streamingkmeans results in $OUTPUT_DIR/output-
   streamingkmeans-cluster-distance.csv"
elif [ "x$clustertype" == "xfuzzykmeans" ]; then
    echo "Step1: Running seq2sparse for fuzzykmeans" && \
  $MAHOUT seq2sparse \
    -i ${INPUT_DIR}/input-seqdir/ \
    -o ${OUTPUT_DIR}/output-seqdir-sparse-fkmeans -ow --
   maxDFPercent 85 --namedVector \
  && \
  echo "Step2: Mahout fkmeans" && \
  $MAHOUT fkmeans \
    -i ${OUTPUT_DIR}/output-seqdir-sparse-fkmeans/tfidf-vectors/
    \
    -c ${OUTPUT_DIR}/output-fkmeans-clusters \
    -o ${OUTPUT_DIR}/output-fkmeans \
    -dm org.apache.mahout.common.distance.CosineDistanceMeasure
   \
    -x 10 -k 20 -ow -m 1.1 \
  && \
  echo "Step3: clusterdump for fkmeans" && \
  $MAHOUT clusterdump \
    -i ${OUTPUT_DIR}/output-fkmeans/clusters-*-final \
    -o ${OUTPUT_DIR}/output-fkmeans/clusterdump \
    -d ${OUTPUT_DIR}/output-seqdir-sparse-fkmeans/dictionary.
   file-0 \
    -dt sequencefile -b 100 -n 20 -sp 0 \
    && \
#  cat ${WORK_DIR}/reuters-fkmeans/clusterdump
  echo "check fkmeans results in $OUTPUT_DIR/output-fkmeans/
    clusterdump"
else
     echo "uknown cluster type: $clustertype"
fi
echo "Done!"

### TODO: cluster labeling ###
### TODO: <doc ID, cluster ID, cluster label> ###
### TODO: put the final output to Avro schema ###
```

B.0.18 The Cluster Labeling Process

The most common cluster labeling method is to use the most frequent or central phrases in a document cluster as labels. We treat document cluster labeling as a ranking problem of the keywords in each cluster.

Keyword selection

We incorporate The keyphrase extraction algorithm KEA [?] method to identify important terms (words) from the documents. KEA is an algorithm for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary. KEA is implemented in Java and is platform independent. It is an open-source software distributed. KEA gets a directory name and processes all documents in this directory that have the extension ".txt". Therefore we write first convert the AVRO files into txt, before feeding it to KEA to get the key words for each cluster. For each candidate phrase KEA computes 4 feature values; first, TFxIDF is a measure describing the specificity of a term for this document under consideration, compared to all other documents in the corpus. Candidate phrases that have high TFxIDF value are more likely to be keyphrases/keywords. Second, the first occurrence is computed as the percentage of the document proceeding the first occurrence of the term in the document. Terms that tend to appear at the start or at the end of a document are more likely to be keywords. Thirst, the length of a phrase is the number of its component words. Finaly, the node degree of a candidate phrase is the number of phrases in the candidate set that are semantically related to this phrase.

Kea first needs to create a model that learns the extraction strategy from manually indexed documents. This means, for each document in the input directory there must be a file with the extension ".key" and the same name as the corresponding document. This file should contain manually assigned keywords, one per line. Given the list of the candidate phrases, Kea marks those that were manually assigned as positive example and all the rest as negative examples. By analyzing the feature values for positive and negative candidate phrases, a model is computed, which reflects the distribution of feature values for each phrase. As we can not manually assign an keys (.key files) for the documents, we used the words from the first line of each document at the .key file. Later we planned to enhance this method using the result from the clustering method, Map-Reduce our put, which give the most frequent words in the cluster as input for .key files for KEA. When extracting keywords from new documents, KEA takes the model and feature values for each candidate phrase and computes its probability of being a keywords. Phrases with the highest probabilities are selected into the final set of keywords. The user can specify the number of keywords that need to be selected. We select the highest 10 probability keywords for each cluster to output from KEA.

Installation and Testing KEA:

```

a) Download the complete package and unzip it.

b) Set KEAHOME to be the directory which contains this README.

  e.g.: export KEAHOME=/home/olena/kea-5.0_full

c) Add $KEAHOME to your CLASSPATH environment variable.

  e.g. export CLASSPATH=$CLASSPATH:$KEAHOME
\item Hadoop-ready driver program to run Mahout clustering
    algorithms on hadoop cluster with large collections.
\item Visualization to judge how well the clustering is
    performing. The visualization need not be graphical. Textual
    representation is fine for the scope of the project.
\item Inputs from the LDA and Social Network team to improve
    clustering results.
d) Add $KEAHOME/lib/*.jar to your CLASSPATH environment variable
    .

  e.g. export CLASSPATH=$CLASSPATH:$KEAHOME/lib/commons-logging.
    jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/icu4j_3_4.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/iri.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/jena.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/snowball.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/weka.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/xercesImpl.jar
       export CLASSPATH=$CLASSPATH:$KEAHOME/lib/kea-5.0.jar


e) Compile and run TestKea

  e.g. javac TestKea
       java -Xmx526M TestKea
```

Build the keyphrase extraction model by running KEAModelBuilder:

```

java kea.main.KEAModelBuilder -l <name_of_directory>  -v  <none>
     -f <text>
```

To extract keyphrases for some documents, put them into an empty directory. Then rename them so that they end with the suffix ".txt"

Extract thee keyphrase extraction model by running KEAKeyphraseExtractor:

```

java KEAKeyphraseExtractor −l <name_of_directory>  −v <none> −f
   <text>
```

Keyword ranking

The keywords are ranked by the number of documents where they serve as important terms. It ranks the terms by the sum of their TF-IDF scores in each document cluster. The top 20 forms a label list. The top 10 are cluster labels.

Appendix C

File Inventory

Document clustering project file inventory list is provided in the Table C.1

File	Description
ClusteringReport.pdf	Final project technical report
ClusteringReport.zip	Latex source code for generating report (includes Readme.txt to reproduce report)
ClusteringPPT.pdf	Final project presentation in pdf format
ClusteringPPT.pptx	Final project presentation in editable pptx format
ClusteringCodeFiles.zip	Project source code and binaries (See C.2 for more details)

Table C.1: Inventory of files in Clustering project

File or Folder	***Description***
adhoc_scripts/ C.3	Various Python and Bash scripts that help in automation
bin/ C.4	Pre-compiled binary JAR files
cluster_evaluation/ C.5	Various scripts that help Clustering evaluation - Silhoutte and Confusion matrix
clustering_cs5604s15.sh	Main clustering script that queues K-Means jobs in Hadoop Cluster
cluster_labeling.sh	Main script to label clusters
data_preprocessing/ C.6	Includes code for Avro to Sequence file conversion and short to long URL expansion
Hierarchical_Clustering/ C.7	Code for cluster labeling and Hierarchical Clustering (includes JAR binary files)

Table C.2: Inventory of files in ClusteringCodeFiles.zip

File	Description
extract_docid_from_points.sh	Extracts document ID from Clustered points
lda-team-eval.sh	Wrapper script to compute row similarity
rows-similarity-computation.sh	Row similarity computation using Mahout
cluster_out_to_avro/cluster_out_to_avro.py	Cluster output to Avro file conversion

Table C.3: Inventory of files in adhoc_scripts directory

File	**Description**
AvroToSequenceFilesCleanedTweetDocId.jar	Converts Avro input to Sequence Files with Tweet schema
AvroToSequenceFilesCleanedWebpages.jar	Converts Avro input to Sequence Files with Web page schema

Table C.4: Inventory of files in bin directory

File	**Description**
confusion_matrix_for_clustering.py	Build confusion matrix for data sets with known labels
dictsize.sh	Dictionary size calculator for various data sets
silhoutte_concat_tweets.sh	Silhoutte score calculator for concatenated tweet data sets
silhoutte_evaluation_clustering.py	Main script to calculate Silhoutte scores
silhoutte.sh	Silhoutte score calculator wrapper script
sparsity.py	Sparsity index calculator
sparsity.sh	Wrapper script to calculate sparsity index
tweets.txt, webpages.txt	Data set names for input to above Bash scripts

Table C.5: Inventory of files in clustering_evaluation directory

File	**Description**
clustering/src/main/java/edu/vt/cs5604s15/clustering/	Source code for Avro to Sequence file conversion
shorturlexpansion/src/main/java/edu/vt/cs5604s15/shorturlexpansion/UrlExpander.java	Source code for short URL to long URL expansion

Table C.6: Inventory of files in data_preprocessing directory

File	**Description**
Hierarchical_clustering.sh	Bash script to perform hierarchical clustering
HClustering.zip	Source code and binary for hierarchical clusteirng
LabelCluster.zip	Source code and binary for cluster labeling
Merge.zip	Source code and binary for merging hierarchical cluster results
Readme.txt	Provides further information on how to execute binaries

Table C.7: Inventory of files in Hierarchical_Clustering directory

Acknowledgement

We would like to thank our sponsor, US National Science Foundation, for funding this project through grant IIS - 1319578.

We would like to thank our class mates for extensive evaluation of this report through peer reviews and invigorating discussions in the class that helped us a lot in the completion of the project.

We would like to specifically thank our instructor Prof. Edward A. Fox and teaching assistants Mohamed Magdy, Sunshin Lee for their constant guidance and encouragement throughout the course of the project.

References

[1] Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn. “Data clustering: a review.” ACM Computing Surveys (CSUR) 31.3 (1999): 264-323.

[2] The Apache Mahout project’s goal is to build a scalable machine learning library. `http://mahout.apache.org/`

[3] Learning Mahout: Clustering. `http://sujitpal.blogspot.com/2012/09/learning-mahout-clustering.html` Last accessed: 02/19/2015

[4] Apache Hadoop. `http://hadoop.apache.org/` Last accessed: 02/19/2015

[5] Jain, Anil K. Data clustering: 50 years beyond K-means. Pattern recognition letters 31.8 (2010): 651-666.

[6] Berry, Michael W. Survey of text mining. Computing Reviews 45.9 (2004): 548.

[7] Beil, Florian, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.

[8] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schtze. Introduction to information retrieval. Vol. 1. Cambridge: Cambridge university press, 2008.

[9] Steinbach, Michael, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. KDD workshop on text mining. Vol. 400. No. 1. 2000.

[10] Zhao, Ying, and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. Proceedings of the eleventh international conference on Information and knowledge management. ACM, 2002.

[11] Dhillon, Inderjit S., Subramanyam Mallela, and Rahul Kumar. Enhanced word clustering for hierarchical text classification. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.

[12] Rosa, Kevin Dela, et al. Topical clustering of tweets. Proceedings of the ACM SIGIR: SWSM (2011).

[13] Zamir, Oren, and Oren Etzioni. Web document clustering: A feasibility demonstration. Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1998.

[14] Strehl, Alexander, Joydeep Ghosh, and Raymond Mooney. Impact of similarity measures on web-page clustering. Workshop on Artificial Intelligence for Web Search (AAAI 2000). 2000.

[15] Cooley, Robert, Bamshad Mobasher, and Jaideep Srivastava. Data preparation for mining world wide web browsing patterns. Knowledge and information systems 1.1 pp:5-32 (1999).

[16] Python package scikit-learn: different clustering algorithm implemented in python. http://scikit-learn.org/stable/modules/clustering.html#clustering Last accessed: 02/19/2015

[17] Python package collective.solr 4.0.3: Solr integration for external indexing and searching. https://pypi.python.org/pypi/collective.solr/4.0.3 Last accessed: 02/19/2015

[18] Python Package Pattern2.6: a web mining module for data mining (Google + Twitter + Wikipedia API, web crawler, HTML DOM parser), machine learning (vector space model, k-means clustering, Naive Bayes + k-NN + SVM classifiers) and network analysis (graph centrality and visualization). https://pypi.python.org/pypi/Pattern Last accessed: 02/19/2015

[19] Carrot2 is an Open Source Search Results Clustering Engine. It can automatically organize small collections of documents. http://project.carrot2.org/ Last accessed: 02/19/2015

[20] Apache SOLR and Carrot2 integration strategies. http://carrot2.github.io/solr-integration-strategies/ Last accessed: 02/19/2015

[21] Gruss, Richard; Morgado, Daniel; Craun, Nate; Shea-Blymyer, Colin, OutbreakSum: Automatic Summarization of Texts Relating to Disease Outbreaks. https://vtechworks.lib.vt.edu/handle/10919/51133 Last accessed: 02/19/2015

[22] Apache Solr: Quick Start Tutorial. http://lucene.apache.org/solr/quickstart.html. Last accessed 02/13/2015

[23] Apache Solr: SchemaXML Wiki. http://wiki.apache.org/solr/SchemaXml Last accessed: 02/13/2015

[24] Apache Maven: Downloads. http://maven.apache.org/download.cgi Last accessed: 02/13/2015

[25] Schwanke, R. W., and Platoff, M. A. Cross References are Features. Proc. ACM 2nd Intl. Workshop on Software Configuration Management, Princeton, N. J., October 1989, 86-95.

[26] M. Shtern and V. Tzerpos, On the comparability of software clustering algorithms, Intl Conf. on Program Compre., ICPC, pp. 64-67, 2010.

[27] Tonella, P., Ricca, F., Pianta, E., and Girardi, C. (2003, September). Using keyword extraction for web site clustering. In Web Site Evolution, 2003. Theme: Architecture. Proceedings. Fifth IEEE International Workshop on, 41-48.

[28] Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of computational and applied mathematics 20 (1987): 53-65.