Distributed SNN Clustering for large document collections

December 26, 2016

Abstract

Your abstract.

1 Introduction

As a consequence of the explosive growth of the WEB, the integration of search engines such as Google into personal computers and mobile devices, and the wide use of social networks, the task of clustering text data, i.e. Tweets or documents in a search engine, has each time an increasing importance because of the necessity of unraveling the underlying categories in large volumes of text data. Nowadays the generation of large amounts of documents surpasses the computational power of personal computers and even of high performance computers. As an example, it is estimated that the amount of WEB pages that popular search engines such as Yahoo! and Google index is higher than the tens of a billion [WWWSize, 2016]. It is therefore of great interest to develop algorithmic techniques capable of automatically organize, classify and summarize document collections distributed in multiple machines of a network, and that also perform efficiently in modern hardware, particularly within parallel processing frameworks in multi-core architectures. In real problems such as *Collection Selection* for distributed document databases, in which for a given query the computer node containing the most suitable sub-collection must be selected to answer it [Crestani and Markov, 2013], the challenges related to the scalability and efficiency of *Knowledge Discovery* methods have become very important. Traditional algorithms often assume that the whole dataset is loaded into main memory (RAM) and thus every document can be accessed at any time with no access latency. In scenarios where the size of the collection is much bigger than the amount of available RAM either because of the number of documents or the length of each document, this ideal loading process is unfeasible due to real constraints in the storage or computational capabilities.

Often, text data is structured in digital collections of documents whose length (number of characters) is variable, e.g. WEB pages or the content generated by users in social networks such as Twitter or Facebook. In order to enable the processing of these collections, in a first stage of preprocessing a set of words occurring in the documents are extracted and sorted in lexicographical order; this word set is referred to as the Vocabulary. Then, the content of every document in the collection is represented as a vector in which each dimension denotes a specific word of the Vocabulary and its value in a document vector is given by a function of the number of occurrences of the word within the document and the number of documents in which it appears. As a natural consequence of the lexical richness of every language, the size of the Vocabulary, and in turn the dimensionality of the vector space onto which a document is represented, is far bigger than the data size that traditional clustering algorithms manage. Because of this, the task of automatic document clustering has high computational and storage (RAM and secondary memory) costs. Along with this, when the number of documents is large (tens or hundreds of thousand and even millions of items) then traditional techniques for processing and clustering documents, and current computational capabilities of a single machine and also high performance machines are insufficient. Even in the most favorable scenario the storage and computational power tackle the challenge but the response time are excessive.

There exist three approaches successfully applied to the construction of clustering algorithms capable of processing large volumes of data. The first one introduces constraints on the number of passes allowed on a document (related to the amount of time it is loaded into main memory) [Yi et al., 2014, Indra et al., 2014]. The second one exploits current multi-core architectures to perform parallel processing of the data, which does not solve the massive volume problem [Zhang et al., 2013]. The last one combines the computational power of a single machine together with the scalable storage capability of a distributed system by partitioning the dataset into several independent machines connected through a network [Sarnovsky et al, 2016].

The last above mentioned approach seems promising since it allows to exploit the local capabilities of single computers with multi-core architectures without sacrificing scalability to large data volumes because of its distributed design. Within this path there are two contexts regarding the data generation scenario. On the one hand, in some problems where the dataset is large but collected in a centralized fashion, the strategy employed consists in partitioning the collection into several machines or nodes of a network. This scheme leads to the transmission of a lot of data during the execution of the algorithm [Nagwani, 2015]. On the other hand, there are some problems where the data is generated in a distributed fashion and where besides it is not feasible to centralize the data because of high transmission costs or privacy issues [Jagannathan et al., 2005, Liu et al., 2012], e.g. search engines work with document collections originated and stored in different geographical locations.

This document is structured as follows: First, a review of the literature on scalable and distributed data clustering methods is presented. Next, the proposed method is shown. Finally,the experimental design along with the attained results and the final remarks are presented.

2 Distributed Clustering Algorithms

As far as we know from the literature, most of the existing efforts for the construction of clustering techniques capable of operating in scenarios where the data is distributed have been focused on low dimensional data (less than 100 attributes) in contrast with document datasets in which a document vector for a small collection may have about 10^4 attributes. Nevertheless, the main advances in distributed data clustering are detailed below, specially highlighting those contributions focused on methods capable of dealing with high dimensional data.

2.1 Main contributions on parallel algorithms

In [Xu et al., 1999] a parallel version of DBSCAN algorithm is presented (PDBSCAN). The authors present the 'shared-nothing' architecture with multiple computers interconnected through a network. A fundamental component of a shared-nothing system is its distributed data structure. They introduce the dR*-tree, a distributed spatial index structure in which the data is spread among multiple computers and the indexes of the data are replicated on every computer. A performance evaluation shows that PDBSCAN offers nearly linear speedup and has excellent scaleup and sizeup behavior. The authors in [Dhillon and Modha, 1999] present an algorithm that exploits the inherent data-parallelism in the kmeans algorithm. They analytically show that the speedup and the scaleup of our algorithm approach the optimal as the number of data points increases. The implementation of this proposal is done on an IBM POWERparallel SP2 with a maximum of 16 nodes. On typical test data sets, nearly linear relative speedups are observed, for example, 15.62 on 16 nodes, and essentially linear scaleup in the size of the data set and in the number of clusters desired. For a 2 gigabyte test data set, the implementation drives the 16 node SP2 at more than 1.8 gigaflops. Another scalable approach based on secondary memory consists in designing algorithms capable of working within the MapReduce framework[1]. In this context it is possible to highlight the contributions made by [Das et al., 2007] in which they propose an implementation of the EM algorithm and also by [Ene et al., 2011] in which they tackle the K-Median problem by using MapReduce. In [Das et al., 2007] the authors present an approach to filter recommendations for users of Google News in order to generate personalized recommendations in a collaborative way. They generate recommendations using three approaches: collaborative filtering using MinHash clustering, Probabilistic Latent Semantic Indexing (PLSI), and covisitation counts. The recommendations are combined from different algorithms using a linear model. The authors claim that the approach is content agnostic and consequently domain independent, making it easily adaptable for other applications and languages with minimal effort. In [Ene et al., 2011] the authors design clustering algorithms that can be used in MapReduce, still one of the most popular programming environment for processing large datasets. They focus on the practical and popular clustering problems, k-center and k-median. A fast clustering algorithms with constant factor approximation guarantees is developed. The algorithms use sampling to decrease the data size and they run a time consuming clustering algorithm such as local search or Lloyd's algorithm on the resulting data set. The proposed algorithms have sufficient flexibility to be used in practice since they run in a constant number of MapReduce rounds. The experiments in the paper show that proposed algorithms' solutions are similar to or better than the other algorithms' solutions. Another parallel approach for K-Means is presented by [Bahmani et al., 2012] and it is called K-Means++. The initialization of the k means algorithm is crucial, and the authors claim that the proposed algorithm obtains an initial set of centers that is provably close to the optimum

[1]Hadoop MapReduce is a software solution that enables the construction of applications capable of processing large amounts of data (e.g. Terabytes) in a parallel fashion over big computer clusters.

solution. A major downside of the k-means++ is its inherent sequential nature, which limits its applicability to massive data: one must make k passes over the data to find a good initial set of centers. In this work the authors show how to drastically reduce the number of passes needed to obtain, in parallel, a good initialization. This is unlike prevailing efforts on parallelizing k-means that have mostly focused on the post-initialization phases of k-means. The proposed initialization algorithm k-means—— obtains a nearly optimal solution after a logarithmic number of passes, and then shows that in practice a constant number of passes suffices.

Additionally, the EM-Tree proposed by [De Vries et al., 2015] is very interesting since it allows to process very large datasets, specifically the authors show that it can handle hundred of millions of WEB pages. They present a scalable algorithm that clusters hundreds of millions of web pages into hundreds of thousands of clusters. It does this on a single mid-range machine using efficient algorithms and compressed document representations. It is applied to two web-scale crawls covering tens of terabytes. ClueWeb09 and ClueWeb12 contain 500 and 733 million web pages and were clustered into 500,000 to 700,000 clusters. Previous approaches clustered a sample that limits the maximum number of discoverable clusters. The proposed EM-tree algorithm uses the entire collection in clustering and produces several orders of magnitude more clusters than the existing algorithms. Fine grained clustering is necessary for meaningful clustering in massive collections where the number of distinct topics grows linearly with collection size. These fine-grained clusters show an improved cluster quality when assessed with two novel evaluations using ad hoc search relevance judgments and spam classifications for external validation. These evaluations solve the problem of assessing the quality of clusters where categorical labeling is unavailable and unfeasible.

2.2 Approaches capable of dealing with high dimensional data

[Kargupta et al., 2001] propose a method to obtain the Principal Components (PCA) over heterogeneous and distributed data. Based on this contribution on dimensionality reduction they also propose a clustering method that works over high dimensional data. Once the global principal components are obtained by using the distributed method and transmitted to each node, local data are projected onto the components and then a traditional clustering technique is applied. Finally, a central node integrates the local clusters in order to obtain a global clustering model.

Several years later, [Liang et al., 2013] present another algorithm for principal components extraction over distributed data. To this end, each node computes PCA over its local data and transmit a fraction of them to a central node. This node uses the received components to estimate the global principal components, which are later transmitted to each node. After this, in every node, local data are projected onto the global components and the projected data are used for computing a coreset by means of a distributed algorithm. The global coreset built from the local projected data will be finally used to obtain a global clustering model.

[Li et al., 2003] propose an algorithm called D-CoFD for high dimensional and distributed data that also is capable of dealing with homogeneous and heterogeneous environments.

2.3 Density based approaches

[Januzaj et al., 2003] propose a distributed data clustering technique in which local nodes build models and transmit a set of representatives of each cluster to a central node. In this node a centralized clustering method is applied over the representatives and then the resulting model is re-transmitted to the local nodes to update their models.

[Klusch et al., 2003] propose a clustering technique based on density estimates.

[Januzaj et al., 2004] present a scalable version of DBSCAN that is also capable of operating over distributed collections. First, the best local representatives a selected depending on the number of points that each one represents and then those chosen points are sent to a central node. This central node clusters the received local representatives into a single new model, which is re-transmitted to the other nodes so they can improve their local group structure.

2.4 Approaches based on parametric models

[Merugu and Ghosh, 2003] propose a clustering method that combines local parametric models, each one built on a single node, into a general one. The main contribution of this work consists in a method capable of dealing with distributed data that also considers the privacy of the local data in each node, since it transmits a summary of each local data and not raw points.

[Kriegel et al., 2005] present a technique that fits a Gaussian mixture model in each node using the EM algorithm. Finally, all these Gaussian mixture models are integrated into a general parametric model.

2.5 Approaches based on representative points

[Forman and Zhang, 2000] extend centroid based techniques to identify groups over distributed data. Specifically, they extend K-Means, K-Harmonic-Means and EM.

[Qi et al., 2008] propose an approximate K-Median clustering technique that works over stresaming data, i.e. data is continuously collected.

[Balcan et al., 2013] address the distributed clustering problem by using centroid based techniques that use a novel coreset construction method that works for distributed data.

[Naldi and Campello, 2014] tackle the problem of the parameter selection of the K-Means clustering algorithm by using evolutionary algorithms. Additionally, they propose two strategies inspired in evolutionary algorithms for clustering distributed data using centroids.

2.6 Hierarchical clustering approaches

A hierarchical algorithm that works on distributed and heterogeneous data is presented by [Johnson and Kargupta, 2000]. This method assumes that data is vertically partitioned, thus all nodes contain the same set of points, but characterized by different features. At the beginning, a dendrogram is generated in each node, and then it is transmitted to a central node. Finally, this node combines all the dendrograms into a global model.

[Jin et al., 2015] propose a hierarchical clustering algorithm for distributed data that builds the clusters by incrementally processing the data points. The authors re-state the hierarchical clustering problem as a Minimum Spanning Tree construction problem over a graph. In order to integrate several local models they also propose a technique for combining multiple minimum spanning trees, assuming that these trees were obtained from disjoint subgraphs of the complete original graph. This mixture procedure iterates until a single tree is obtained, which in turn denotes the hierarchical clustering originally pursued.

3 Proposal

In this work we present a distributed clustering algorithm based on *Shared-Nearest-Neighbor* (SNN) clustering. This method automatically identifies the number of underlying groups along with a set of representative points for each one. We pose that this method is able to deal with collections arbitrarily distributed across a Master/Worker architecture as depicted in figure 1 and the overall algorithm operates in two stages: The first one starts when the data is randomly partitioned and distributed into several nodes, then each one generates a set of representative points per cluster, i.e. *core-points*, and finally, transmits back a sample set of these *core-points* to the master node. In the second stage, the central node joins the sample points received and then starts a centralized SNN clustering over them. Finally, the set of representative points is labeled and also a new set of *core-points* that summarizes the overall collection is obtained.

Parameters of the algorithm

The proposed method comprehends three parameters, namely k, Eps, MinPts and ω. The value of parameter k determines the size of the neighborhood over which the SNN similarity measure is going to be computed. In the SNN space, the value of Eps denotes the similarity threshold beyond which two points are considered as close. Additionally, a point is identified as a *core-point* when the number of points close to it in the SNN space surpasses the value of parameter MinPts. Finally, the value of the parameter ω rules the size of the sample set of *core-points* that is going to be transmitted.

Initial stage

This stage starts by randomly partitioning and distributing the dataset into several worker nodes. In this primary stage, after a worker node n_i receives its data chunk $\mathcal{D}_i$, it starts to identify the core-points by following procedure SnnCorePoints described in algorithm 1. Once its core-point set $\mathcal{C}_i$ is built, an attempt to assign a cluster label to each point $p \in \mathcal{C}_i$ is performed.

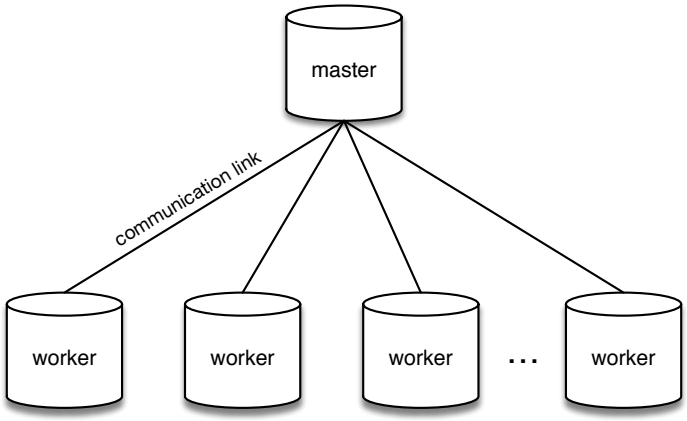


Figure 1: Network architecture employed by the proposed algorithm

Initially, all points are unlabeled. Then, iteratively, each point p is tagged with a new cluster label l_p and subsequently all remaining points located within a radius of Eps from p in the SNN space are labeled with l_p.

Once all core-points are labeled, a weighted sample from each cluster is drawn. The weight of a point follows the expression:

$$\frac{1 - (N_l/N_c)}{2 \cdot N_l}$$

where N_c denotes the number of labeled core-points and N_l the number of points labeled with label l. The expression shown above denotes that the sampling weight of a core-point is inverse to its group size as it is depicted in figure 2. The aim of this expression is to build a sample of core-points capable of representing both small and large groups alike without a bias to larger clusters which is a serious problem for clustering algorithms operating under unbalanced data scenarios. The overall procedure is described in algorithm 2.

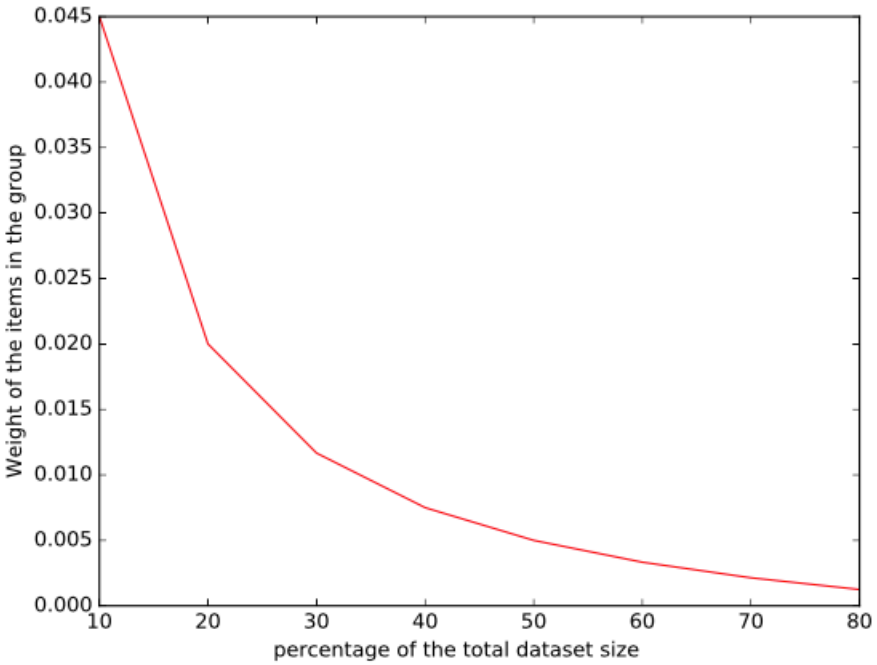


Figure 2: Weight of each item in a group vs the relative size of the group in comparison with the total dataset size.

Final stage

This stage starts when all worker nodes transmit their weighted sample of core-points to the master node and then this node joins all these points into a single dataset $\mathcal{S}$. Then the procedure SnnCorePoints is applied over $\mathcal{S}$, obtaining a new core-point set $\mathcal{C}$, and the same labeling step applied in the initial stage in each worker is performed over this set. After this step, for each point $q \in \mathcal{S}$ not chosen for the core-point set $\mathcal{C}$, its nearest neighbor $p \in \mathcal{C}$ is found. If the SNN similarity between these points is lower than Eps, then q is marked as noise, otherwise label l_p is assigned to q. Finally, the set of labeled points in $\mathcal{S}$ (i.e. discarding the noisy subset) is returned. A thorough description appears in algorithm 3.

Algorithm 1: Identification of core-points from a dataset

Function `SnnCorePoints` ($\mathcal{D}$, k, Eps, MinPts) **:**
- **foreach** $p \in \mathcal{D}$ **do**
  - compute $KNN_k(p)$
- **foreach** $p \in \mathcal{D}$ **do**
  - **foreach** $q \neq p \in \mathcal{D}$ **do**
    - $\mathsf{SNN}_k(p,q) \leftarrow \#(KNN_k(p) \cap KNN_k(q))$
  - $\mathsf{density}(p) \leftarrow \#\{q \neq p \in \mathcal{D} | \mathsf{SNN}_k(p,q) > \mathsf{Eps}\}$
  - **if** $\mathsf{density}(p) > \mathsf{MinPts}$ **then** Mark p as **core-point**
- $\mathcal{C} \leftarrow$ all points marked as **core-point**
- **return** $\mathcal{C}$, SNN_k

Algorithm 2: Selection of representative points executed in a node

Function `SampleLocalData` ($\mathcal{D}$, k, Eps, MinPts, $\omega \in [0,1]$) **:**
- $\mathcal{C}, \mathsf{SNN}_k \leftarrow$ `SnnCorePoints` ($\mathcal{D}$, k, Eps, MinPts)
- **foreach core-point** $p \in \mathcal{C}$ **do**
  - **if** p *is not visited* **then**
    - Assign a new cluster label l_p to p
    - mark p as visited
    - **foreach** *Not visited* $q \neq p \in \mathcal{C}$ **do**
      - **if** $\mathsf{SNN}_k(p,q) > \mathsf{Eps}$ **then**
        - Assign label l_p to q
        - mark q as visited
- $N_c \leftarrow$Number of labeled **core-points**
- $S \leftarrow \emptyset$
- **foreach** *cluster* l *having size* N_l **do**
  - $S \leftarrow$ **Add** $\omega \cdot N_l$ **points sampled from group** l **with weight** $\dfrac{1-(N_l/N_c)}{2 \cdot N_l}$
- **return sampled data** S

4 Methodology and Experimental results

In this section, we assess the performance of the proposed distributed algorithm and contrasts its effectiveness against a centralized version and against an efficient and widely used SNN-graph bisection method[Zhao et al., 2002], both of them use the complete dataset loaded into main memory.

Initially, a proof of concept was made by applying the algorithm over the synthetic dataset shown in figure 3 which was proposed by [Guha et al., 1998]. Since this dataset contains points in two dimensions, the discriminative capability of the proposal in presence of high dimensional data is not assessed. In spite of that, the effective-

Algorithm 3: Procedure executed by the master node to generate the list of representative points per group

Input: $\mathcal{D}$ denotes the local subset of the data, k, Eps, MinPts, $\omega \in [0, 1]$
Partition the dataset $\mathcal{D}$ into M subsets $\{D_1, D_2, \ldots D_M\}$
On each *node* n_i **execute**
  $S_i \leftarrow$ `SampleLocalData(`D_i`)/* remote procedure call on node` n_i `*/`
$S \leftarrow [S_1, S_2, \ldots S_M]$
$\mathcal{C}, \mathsf{SNN}_k \leftarrow$ `SnnCorePoints(`$\mathcal{S}$, k, Eps, MinPts`)`
foreach core-point $p \in \mathcal{C}$ **do**
  if p *is not visited* **then**
    Assign a new cluster label l_p to p
    mark p as visited
    foreach *Not visited* $q \neq p \in \mathcal{C}$ **do**
      if $\mathsf{SNN}_k(p, q) >$ Eps **then**
        Assign label l_p to q
        mark q as visited
foreach $p \in S \setminus \mathcal{C}$ **do**
  $n_p \leftarrow \arg\max_{q \in \mathcal{C}} \mathsf{SNN}_k(p, q)$
  if $\mathsf{SNN}_k(p, n_p) <$ Eps **then** Mark p as **noise**
  else Assign label l_{n_q} to p
Output: $\{l_p, p \in S \mid l_p \neq$ **noise**$\}$

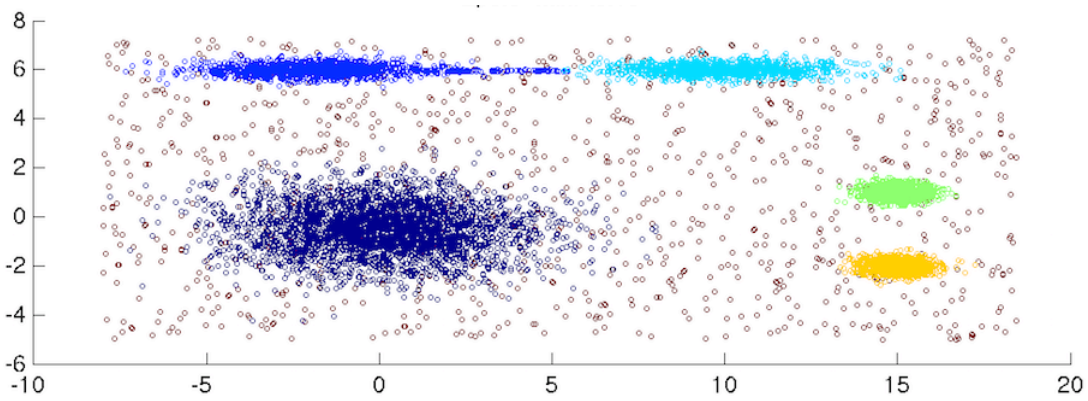


Figure 3: Toy example: Original data.

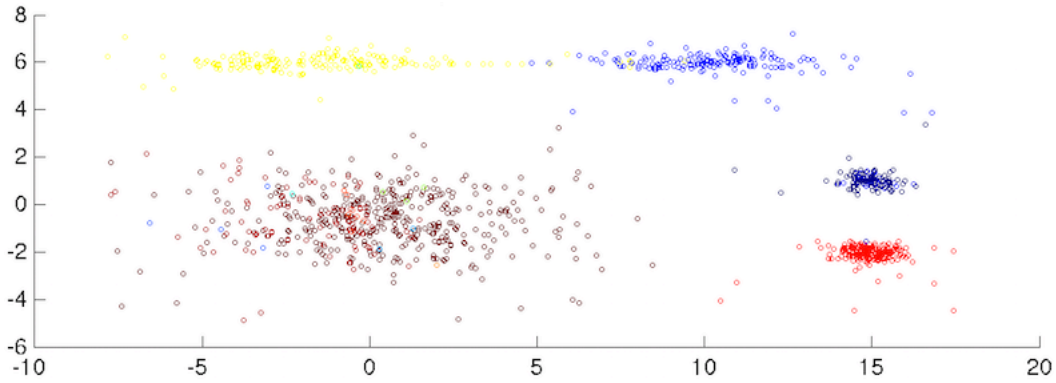


Figure 4: Toy example: Data after distributed snn clustering with $Eps = 50$.

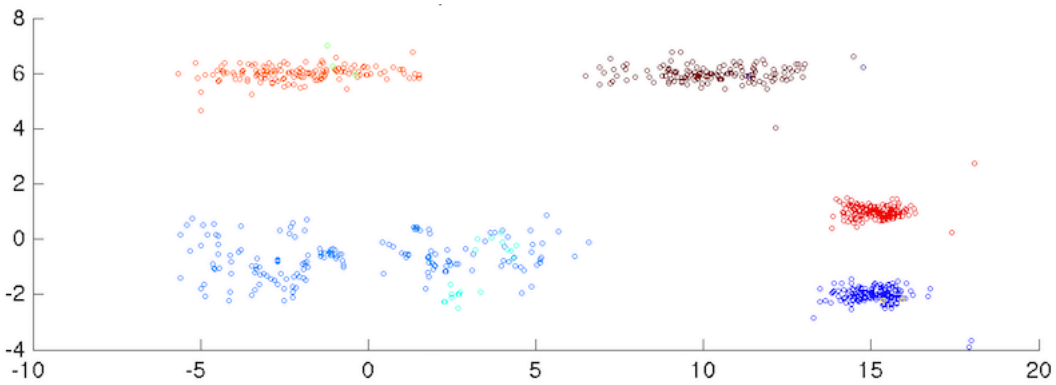


Figure 5: Toy example: Data after distributed snn clustering with $Eps = 60$.

ness of the distributed algorithm and its robustness to noise in the data are evaluated in a qualitative fashion. The obtained clusters in two independent runs with different parameters are depicted in figures 4 and 5.

By contrasting figure 3 against 4 and then against figure 5, it is noticed that the noise layer was removed by the clustering algorithm in both cases. Additionally, the increase in the Eps value had an impact onto the sensitivity of the algorithm to detect outliers, i.e. a higher value for this parameter was accompanied by an increasing rate of points marked as noise as it is specially noticed in the chain of outliers that connects the two ellipsoids in figure 3.

Datasets

The experiments were performed over 5 collections containing documents represented as vectors in high dimensional term spaces. By following the descriptions shown in table 1, firstly the 20-Newsgroup (M5 partition) collection contains 5000 newsgroup documents coming from different subjects, namely computers, motorcycles, baseball, science and politics. Then, the remaining document sets were extracted from the Tipster collection[2] and comprise several heterogeneous datasets such as *DOE* which contains short abstracts of the Department of Energy, and *FR* which contains reports of actions taken by U.S. government agencies. Also in table 1, the number of documents, the size of the vocabulary, the number of classes and the percentage of non zero values (a measure of the sparsity of the collection) are shown. As a final remark, all collections are quite sparse and also their document vectors are spanned onto high dimensional term spaces with tens of thousands of features.

In each one of the subfigures appearing in the Figure 6, similarities between all pair of documents in each collection are plotted by using a gray scale. Darker spots denote higher similarity values and also the documents were ordered consecutively by the group label. In all the figures, both axes contain the document numbers (starting from 0) and, regarding the previously mentioned document order, it is possible to observe the groups denoted as darker rectangular patches along the diagonal line.

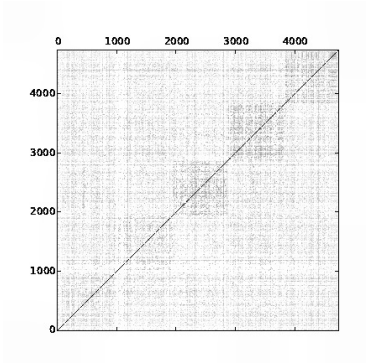
Dataset	Description	instances	$\lvert\mathcal{V}\rvert$	classes	%NNZ
20NG	20-newsgroup data, m5 partition.	4743	41223	5	0.167%
DOE	Department of Energy Abstracts	1664	15755	14	0.365%
FR	Federal Register notes	926	50427	14	1.104%
SJMN	San Jose Mercury News	908	23616	16	0.738%
ZF	Computer select disks by Ziff-Davis	3263	58398	25	0.360%

Table 1: Datasets employed.

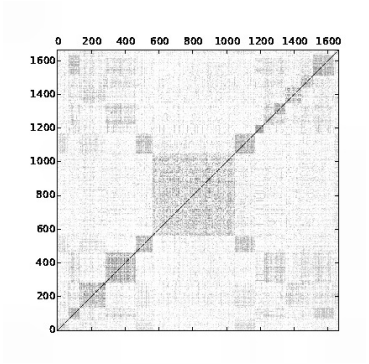
Document processing

To obtain a representative set of terms for each collection, i.e. its index terms, the documents are preprocessed as it is depicted in figure 7. This figure shows standard text processing steps commonly used in Information retrieval and text clustering tasks. Firstly, as documents within the Tipster collections come in SGML format, their content

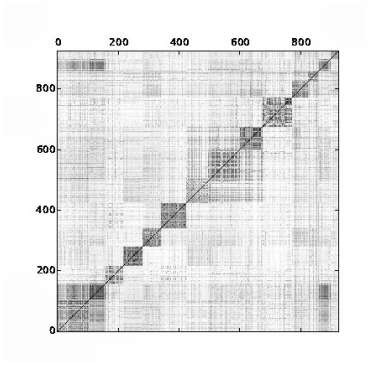
[2] `https://tac.nist.gov/data/data_desc.html#TIPSTER` (Last visit on November 8th, 2016.)



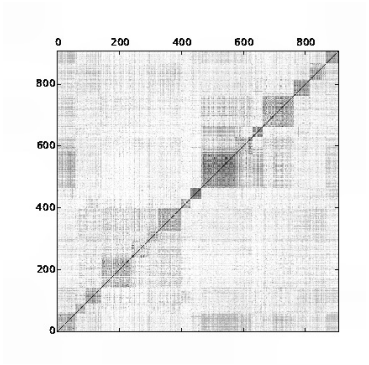
(a) The 20NG collection.



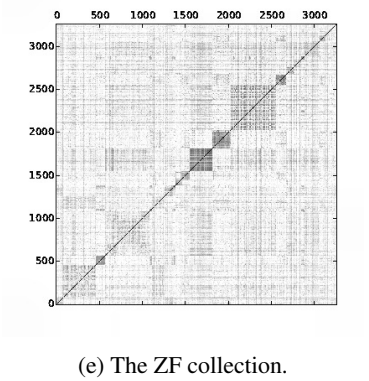
(b) The DOE collection.



(c) The FR collection.



(d) The SJMN collection.



(e) The ZF collection.

Figure 6: Pairwise cosine-similarity matrices

is extracted. Then, spacing and punctuation marks are removed in order to keep only the candidate terms of the vocabulary. After this step, the set of candidate terms is filtered by using a standard English stopword set, which aids to remove meaningless words (e.g. articles) and some highly frequent words. No other ad-hoc filter is applied to the documents in this work (e.g. stemming or removing very high and low frequency words) in order to keep the text content as similar as possible to the original version and specially to allow subsequent generalizations of the performance results of the proposal without any tie to a specific dataset. Finally, the selected terms within each collection are sorted and then the index term set or vocabulary is obtained.

After the vocabulary of each collection is built, the number of occurrences of its terms in each document are computed and used to build the vector representing each document. That is, the number of occurrences of a term i of the vocabulary $\mathcal{V}$ in the document j of the collection is denoted as $f_{i,j}$, also the number of documents of the collections that contain this term is denoted as N_i. By following this notation, the weight of term i within a document j is quantified by the expression:

$$w_{i,j} = \frac{f_{i,j}}{\max_{t \in \mathcal{V}} f_{t,j}} \cdot \log \frac{|\mathcal{V}|}{N_i}$$

In this way, the vector representing document j is made up by the weight of each term of the vocabulary in it.

Clustering quality measures

In this work, only external validation measures are employed, namely **Entropy**, **Purity**, **Adjusted-Rand-Index**, **Adjusted-Mutual-Information**, **Homogeneity**, **Completeness** and the **V-Measure** which consists in the harmonic mean between the Homogeneity and Completeness scores. These scores, with the exception of the Adjusted-Rand-Index which is within $[-1, 1]$, have positive values within $[0, 1]$. Also, with the exception of Entropy, for all of them larger values denote a better clustering quality.

Entropy, measures the extent in which all data from one class are spread into several groups and **Purity** measures the extent that a cluster contains only one class of data. **Adjusted-Rand-Index** measures the matching between the true labels and the ones produced by the algorithm under study by counting the proportion of point-pairs whose relationship is the same in both labelings. **Adjusted-Mutual-Information** accounts for the matching level between two clusterings by measuring the amount of information shared between the true labeling and the one proposed by an algorithm. Both scores are adjusted because of the original estimates are higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared between them. **Homogeneity** measures the extent in which the clusters contain only data points which are members of a single class. **Completeness** accounts for the extent in which all the data points that are members of a given class are elements of the same cluster.

Parameter tuning

In order to report the best performance found for each algorithm, its parameters must be tuned. Hence, for each algorithm a grid search is performed seeking the parameter values that allow to attain the best algorithm performance. The parameter grid used for the Centralized and Distributed SNN clustering algorithms consists of 288 configurations where the values of K are in $\{50, 70, 90, 110\}$, Eps are in $\{3, 5, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ and MinPts are in $\{5, 10, 15, 20, 25, 30\}$. For the SNN-Graph bisection algorithm the same values for K were tested but no difference in the performance was detected. In Tables 2 and 3 the parameters chosen after the tuning procedure are detailed.

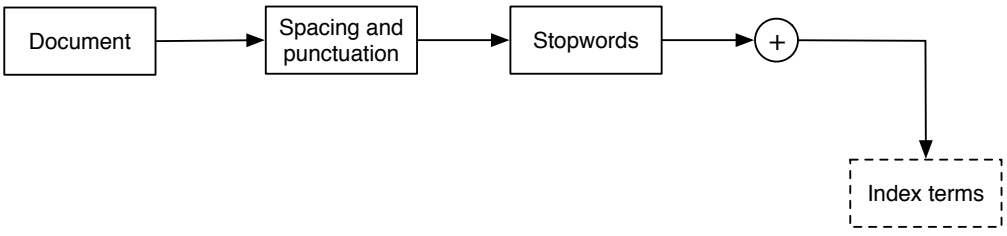


Figure 7: Text processing steps performed in each collection.

Dataset	K	Eps	MinPts
20NG	110	25	30
DOE	70	25	30
FR	50	25	20
SJMN	50	20	30
ZF	90	40	25

Table 2: Parameters of the centralized SNN-Clustering algorithm selected after the tuning procedure.

Dataset	K	Eps	MinPts
20NG	30	8	25
DOE	30	10	20
FR	30	10	5
SJMN	30	10	10
ZF	30	10	15

Table 3: Parameters of the distributed SNN-Clustering algorithm selected after the tuning procedure.

Results

The results shown in Table 4 demonstrate that the distributed SNN attains a comparable performance to the other two algorithms which also attained acceptable results. In most cases the proposal achieved better scores, nevertheless it is important to mention that a potential improvement is possible for the graph clustering algorithm by applying a better tuning strategy.

20NG and ZF datasets presented more difficulties to the centralized algorithms in terms of identifying homogeneous clusters.

	Dataset	Entropy	Purity	ARI	AMI	HOM	COM	VM
Graph Clust	20NG	0.2948	0.8307	0.6050	0.6421	0.6619	0.6425	0.6521
	DOE	0.2721	0.7139	0.4919	**0.7030**	0.7095	**0.7461**	0.7273
	FR	0.2524	0.7559	0.6185	0.7266	0.7375	0.7452	0.7413
	SJMN	0.2412	0.7544	0.5953	0.7367	0.7505	0.7657	0.7580
	ZF	0.4166	0.6028	0.3817	0.5444	0.5593	0.6015	0.5796
C-SNN	20NG	0.3525	0.6036	0.3432	0.3953	0.3990	0.4793	0.4355
	DOE	0.2865	0.6911	0.4197	0.6370	0.6476	0.6711	0.6591
	FR	0.1921	0.8531	0.7099	**0.7834**	0.7969	0.7919	0.7944
	SJMN	0.2024	**0.8282**	0.5846	0.6820	0.7732	0.6960	0.7326
	ZF	0.3330	0.6089	0.2845	0.5084	0.5750	0.5238	0.5482
D-SNN	20NG	**0.1710**	**0.8828**	**0.8400**	**0.8218**	**0.8262**	**0.9167**	**0.8691**
	DOE	**0.1074**	**0.9318**	**0.6816**	0.7029	**0.8794**	0.7227	**0.7934**
	FR	**0.0949**	**0.9134**	**0.7224**	0.7546	**0.8947**	**0.7784**	**0.8325**
	SJMN	**0.1802**	0.7656	**0.6360**	**0.7836**	**0.8052**	**0.8040**	**0.8046**
	ZF	**0.0117**	**0.9943**	**0.7809**	**0.7701**	**0.9882**	**0.7877**	**0.8766**

Table 4: Performance attained by the graph clustering algorithm, the centralized SNN algorithm and the distributed proposal over the text collections. The best values for each measure in each dataset appear in bold face.

5 Conclusions and future work

In this work, the hypothesis consists in that the proposal is able to deal with distributed text collections and then to recover the group structure underlying the whole dataset with a quality comparable to centralized algorithms successfully used in the clustering literature.

Beyond the labeling of each data point, the task that we found more important specially in BIgData scenarios is the cluster recovery task. Its aim is to summarize a data collection by reconstructing the group structure underlying

the set of data points.

6 Acknowledgment

Juan Zamora was supported by an intern postdoc project from Pontificia Universidad Católica de Valparaíso.

References

[Bahmani et al., 2012] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. (2012). Scalable K-Means ++. *Proceedings of the VLDB Endowment (PVLDB)*, 5:622–633.

[Balcan et al., 2013] Balcan, M. F., Ehrlich, S., and Liang, Y. (2013). Distributed k -Means and k -Median Clustering on General Topologies. *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 1–9.

[Crestani and Markov, 2013] Crestani, F., and Markov, I. (2013). Distributed Information Retrieval and Applications. *35th European Conference on IR Research*, 865–868.

[Das et al., 2007] Das, A., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM.

[De Vries et al., 2015] De Vries, C. M., De Vine, L., Geva, S., and Nayak, R. (2015). Parallel streaming signature EM-tree: A clustering algorithm for web scale applications. In *Proceedings of the 24th International Conference on World Wide Web*, pages 216–226. ACM.

[Dhillon and Modha, 1999] Dhillon, I. S. and Modha, D. S. (1999). A data-clustering algorithm on distributed memory multiprocessors. *LargeScale Parallel Data Mining*, 1759(802):245–260.

[Ene et al., 2011] Ene, A., Im, S., and Moseley, B. (2011). Fast Clustering using MapReduce. *Kdd*, 681–689.

[Ertöz et al., 2003] Ertöz, L., Steinbach, M., and Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. *Proceedings of the SIAM International Conference on Data Mining*, 47–58.

[Forman and Zhang, 2000] Forman, G. and Zhang, B. (2000). Distributed data clustering can be efficient and exact. *ACM SIGKDD Explorations Newsletter*, 2(2):34–38.

[Guha et al., 1998] Guha, S., Rastogi, R. and Shim, K. (1998). CURE: an efficient clustering algorithm for large databases. *ACM SIGMOD Record*, 27(2):73–84.

[Han et al., 2011] Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[Indra et al., 2014] Indra, Z., Zamin, N., Jaafar, J. (2014). A clustering technique using single pass clustering algorithm for search engine. *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*, 1:182–187.

[Jagannathan et al., 2005] Jagannathan, G., and Wright, R. N. (2005). Privacy-preserving Distributed K-means Clustering over Arbitrarily Partitioned Data. *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 593–599.

[Januzaj et al., 2003] Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2003). Towards Effective and Efficient Distributed Clustering. *Workshop on Clustering Large Data Sets*, pages 49–58.

[Januzaj et al., 2004] Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2004). Scalable Density-Based Distributed Clustering. pages 231–244.

[Jin et al., 2015] Jin, C., Chen, Z., Hendrix, W., Agrawal, A., and Choudhary, A. (2015). Incremental, Distributed Single-linkage Hierarchical Clustering Algorithm Using Mapreduce. *Proceedings of the Symposium on High Performance Computing*, pages 83–92.

[Johnson and Kargupta, 2000] Johnson, E. and Kargupta, H. (2000). Collective, hierarchical clustering from distributed, heterogeneous data. *Lecture Notes in Computer Science*, 1759:221–244.

[Kargupta et al., 2001] Kargupta, H., Huang, W., Sivakumar, K., and Johnson, E. (2001). Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):422–448.

[Klusch et al., 2003] Klusch, M., Lodi, S., and Moro, G. (2003). Distributed clustering based on sampling local density estimates. *IJCAI International Joint Conference on Artificial Intelligence*, pages 485–490.

[Kriegel et al., 2005] Kriegel, H.-p., Kr, P., Pryakhin, A., and Schubert, M. (2005). Effective and Efficient Distributed Model-based Clustering.

[Li et al., 2003] Li, T., Zhu, S., and Ogihara, M. (2003). Algorithms for Clustering High Dimensional and Distributed Data. *Intelligent Data Analysis Journal*, 7(February):1–36.

[Liang et al., 2013] Liang, Y., Balcan, M.-f., and Kanchanapally, V. (2013). Distributed PCA and k-Means Clustering. *The Big Learning Workshop in NIPS 2013*, pages 1–8.

[Liu et al., 2012] Liu, J., Huang, J. Z., Luo, J., and Xiong, L. (2012). Privacy Preserving Distributed DBSCAN Clustering. *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, 177–185.

[Merugu and Ghosh, 2003] Merugu, S. and Ghosh, J. (2003). Privacy-preserving Distributed Clustering using Generative Models. *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 0–7.

[Nagwani, 2015] Nagwani, N. K. (2015). Summarizing large text collection using topic modeling and clustering based on MapReduce framework. *Journal of Big Data*, 2:1–18.

[Naldi and Campello, 2014] Naldi, M. C. and Campello, R. J. G. B. (2014). Evolutionary k-means for distributed data sets. *Neurocomputing*, 127:30–42.

[Qi et al., 2008] Qi, Z., Jinze, L., and Wei, W. (2008). Approximate clustering on distributed data streams. *Proceedings - International Conference on Data Engineering*, 00:1131–1139.

[Rousseeuw, 1987] Rousseeuw Peter J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

[Sarnovsky et al, 2016] Sarnovsky, M., Carnoka, N.. (2016). Distributed Algorithm for Text Documents Clustering Based on k-Means Approach. *Advances in Intelligent Systems and Computing*, 430:165–174.

[WWWSize, 2016] http://www.worldwidewebsize.com/ Accessed at 26th September 2016

[Xu et al., 1999] Xu, X., Jäger, J., and Kriegel, H. (1999). A fast parallel clustering algorithm for large spatial databases. *High Performance Data Mining*, 290:263–290.

[Yi et al., 2014] Yi, J., ZShang, L., Wang, J., Jin, R., Jain, A.K. (2014) A Single-Pass Algorithm for Efficiently Recovering Sparse Cluster Centers of High-dimensional Data. *Proceedings of the 31 st International Conference on Machine Learning, Beijing, China, 2014.*, 3:2112–2127.

[Zhang et al., 2013] Zhang, J., Wu, G., Hu, X., Li, S., Hao, S. (2013). A Parallel Clustering Algorithm with MPI – MKmeans . *Jouirnal of Computers*, 8(1):10–18.

[Zhao et al., 2002] Zhao, Y. and Karypis, G. (2002). Evaluation of hierarchical clustering algorithms for document datasets. *Proceedings of the eleventh international conference on Information and knowledge management*, 515–524.