Scalable k-NN based text clustering

Alessandro Lulli*†, Thibault Debatty§‖, Matteo Dell'Amico‡, Pietro Michiardi‖, Laura Ricci*†
*University of Pisa, Italy {surname}@di.unipi.it
†ISTI, CNR, Pisa, Italy {name.surname}@isti.cnr.it
‡Symantec Research Labs, matteo_dellamico@symantec.com
§Royal Military Academy, Brussels, Belgium, {name.surname}@rma.ac.be
‖EURECOM, Campus SophiaTech, France, {name.surname}@eurecom.fr

***Abstract*—Clustering items using textual features is an important problem with many applications, such as root-cause analysis of spam campaigns, as well as identifying common topics in social media. Due to the sheer size of such data, algorithmic scalability becomes a major concern. In this work, we present our approach for text clustering that builds an approximate k-NN graph, which is then used to compute connected components representing clusters. Our focus is to understand the scalability / accuracy tradeoff that underlies our method: we do so through an extensive experimental campaign, where we use real-life datasets, and show that even rough approximations of k-NN graphs are sufficient to identify valid clusters. Our method is scalable and can be easily tuned to meet requirements stemming from different application domains.**

I. Introduction

Data clustering and analysis is a fundamental task, that consists in finding groups of related data items, according to a definition of similarity that is application specific.

In this work, we focus on a particular data clustering task, which involves grouping text data items. The application domain of our work stems from the objective of analyzing SPAM campaigns: for instance, we focus on data collected by Symantec Research Labs, that perform root-cause analysis of large scale SPAM email campaigns originated from bot networks. In this *adversarial* context, data clustering is even more challenging, because spammers manipulate text to avoid SPAM emails being clustered the same campaign. Hence, the similarity metrics used for clustering must cope with text mangling, which require *non-metric* distances that disregard typos, character swapping, and other techniques to avoid detection. In addition, data clustering is challenging due to the large volume of data: this calls for the design of scalable algorithms, capable of ingesting millions of data points and cluster them in meaningful ways.

Current approaches fall short in addressing the above challenges. For example, the widespread K-means clustering algorithm requires text data to be transformed into d-dimensional vectors to operate correctly. However, such transformations generally imply high-dimensional vectors, which render distance functions problematic. Similarly, approaches for text clustering based on frequency analysis of shingles suffers from high-dimensionality problems. Moreover, the techniques discussed above are not amenable to non-metric spaces, which is a requirement for the application domain we study. As a consequence, clustering can perform poorly. Finally, designing scalable clustering algorithms is hard. For example, K-means – albeit easy to parallelize – suffers from a large runtime when K is large, and requires a large number of similarity computations. Frequency based methods are also difficult to scale, as it is generally cumbersome to parallelize frequent itemset mining algorithms.

In this work we present a clustering approach that addresses the above concerns: *i)* it produces high quality clusters that are easy to interpret, *ii)* it accommodates any kind of similarity metrics, and *iii)* it is scalable. The gist of our approach consists in building an *approximate* k-NN graph of the input text data, and compute its connected components, which identify data clusters. In particular, we aim at understanding the trade-off that exists between accuracy, scalability and, ultimately, clustering quality. To do so, we perform a thorough experimental evaluation of our method with real, large-scale datasets, using our implementation for the Apache Spark.

In summary, the contributions of our paper are as follows:

- We design and implement a *scalable* algorithm for text clustering, which works in an "adversarial" setting, and that produces high quality, and interpretable clusters.
- We perform a detailed experimental analysis, where we show the impact of the parameters that govern the degree of approximation of our method. Our results indicate that even rough approximations are sufficient to obtain high quality clusters.
- We use real-life datasets and evaluate the overall clustering quality of our approach both using traditional metrics and with the help of domain experts through manual investigation, highlighting the interpretability of clustering results.

II. Related Work

Data clustering has been widely studied in the literature, with nuances ranging from graph theoretic and data mining principles [13], [5] to experimental approaches [30], [35].

One of the most popular algorithm for clustering is K-means [21], which is a simple approach that can be used to perform clustering, where K indicates the number of clusters the algorithm produces. Since K-means operates on d-dimensional vectors, text clustering requires a *transformation phase* to encode sentences and words into vectors [17], [32]. For example, Mikolov et al. [24] present an efficient implementation of the continuous bag-of-words and skip-gram

architectures for computing vector representations of words. In our work, we use such approach as a **baseline** to which we compare our method, and show that it suffers from the underlying inability to accept *non-metric* distance measures, which are essential to detect similarity between mangled sentences, and from its poor scalability.

Alternative approaches search for frequent terms in the dataset to identify clusters [7], [6], [23]: the idea is to find subsets of frequent term sets, which are a proxy for clusters, and map data items containing elements of such subsets to the same cluster. Such approaches scale poorly, and do not take into account similarity metrics resilient to mangling.

Other approaches aim to optimize the computation of pairwise similarity between text items using matrix computations [25]. In this category, recently, Lin et al. [19] present an optimized algorithm to retrieve clustering of text data from a similarity matrix, using cosine similarity. In general, such approaches do not accommodate non-metric similarity measures and are difficult to scale, although recent work [8] has shown the benefits of approximate matrix operations, which scale better than exact, all-pair similarity computations.

An approach that targets goals that are similar to ours is Triage [31], which addresses the same application domain we target in this paper. However, the focus of Triage is on multi-feature data items, and not on scalability: the authors mainly address problems related to information fusion, by defining a method to merge several different distance metrics operating on text, categorical and numerical values. Recently, a parallel version of Triage has been proposed [29], which partially addresses scalability issues. However, the approach still computes all-pair similarity among representative, prototype items, with an $O\left(n^2\right)$ complexity that still makes handling very large data sets difficult.

III. k-NN BASED CLUSTERING

We present our approach for text clustering, which is based on a scalable, randomized algorithm, and recognizes the role played by *approximation* which constitutes an important contribution to our analysis of the trade-off that exists between clustering quality and the scalability of our method.

The problem of text clustering we consider is particularly challenging due to the application scenario we study. We face an adversarial setting in which text data is generated such that finding similar items is cumbersome: SPAM campaigns introduce text mangling, spelling errors and generally variations on some baseline text which makes SPAM items belonging to the same campaign appear different one from each other. As a consequence, we need to use a similarity metric between items that can overcome, or at least mitigate, the problem.

Similarity Metric. There exist numerous similarity metrics in the vast literature on the subject of this work. In particular, for text data, the Hamming distance and Levenshtein distance have been extensively used to determine the similarity among text items. In this work, for the reasons illustrated above, we choose the Jaro-Winkler [14], [33] similarity metric which, simply stated, counts the common characters between two strings even if they are misplaced, misspelled, and mangled by a "short" distance. Note that, the Jaro-Winkler metric has a codomain in $[-1, 1]$.

Given the choice of the similarity metric we use in this work, the wide spectrum of techniques to find clusters of similar text items reduces to few methods. This is the main driving factor that steers the algorithmic design choices we make in this work.

A. Phase 1: k-NN Graph Construction

Algorithm 1: k-NN construction

```
1 procedure Map(Node n, NeighborList(n))
2     forall the u ∈ NeighborList(n) ∪ n do
3         forall the v ∈ NeighborList(n) ∪ n \ u do
4             EMIT(u, (v, SIMILARITY(u, v)))
5         end
6     end
7 procedure Reduce(Node n,List[(Node u,
  Similarity s)] l)
8     orderedList = ORDERDESC(l).LIMIT (k)
9     EMIT(n, orderedList)
```

In the first phase of our method, we build a k-NN graph. Essentially, the construction of a k-NN graph is the process of building a directed graph from a set of items V, with vertex set equals to V and an edge from each $v \in V$ to its k most similar items in V under a given similarity measure. In this work, we limit our attention to text features: for example, we extract the subject of a SPAM email as the only representative feature of the item. Considering additional, heterogeneous features is outside the scope of this work, and we defer it to an extension of our approach.

The naïve approach to build a k-NN graph consists in finding all-pairs similarity among all items of a dataset, then select the k most similar items to each item. Clearly, this "brute-force" approach is not scalable, as it requires $O(n^2)$ similarity computations, where n is the number of items in the dataset. Note that the "brute-force" algorithm produces *exact* k-NN graphs, which we use in this work as a baseline to determine the approximation quality of our method.

In this work, we design a parallel version of the NNDescent algorithm [12], which is an elegant, and widely used method to build approximate k-NN graphs through an iterative procedure.[1] From NNDescent, our approach inherits the capability of using *arbitrary* similarity metrics, including Jaro-Winkler. Although alternative approaches to build k-NN graphs exist, for example using locality sensitive hashing (LSH) [34], [26], such methods do not extend to arbitrary similarity metrics.[2]

The main idea behind the k-NN graph algorithm we use in the first phase of our method is to iteratively improve an

[1] Although an Hadoop MapReduce version of NNDescent is discussed in [12], we are not aware of any experimental validation of it. Moreover, in our work we use Spark, a more efficient MapReduce framework geared toward iterative algorithms.

[2] We are currently working on an extension of this work to include modern LSH-based algorithms that can support arbitrary similarity metrics [15], [10], [9] as well.

initial, random k-NN graph, by "swapping" the neighborhood of each node, searching for similar candidates among its two-hop neighborhood. Increasing the number of iterations allows the algorithm to converge to better approximations of the k-NN graph, at the cost of higher convergence time.

Algorithm 1 illustrates the pseudo-code of a generic iteration of our parallel k-NN graph algorithm, which we cast through the MapReduce programming model. Note that the output of the algorithm is a *weighted* k-NN graph, where each edge is labelled with the similarity measure between its end vertexes. First, we initialize the algorithm by building a random, undirected k-NN graph:[3] each node of the graph (*i.e.* a data item) is assigned k random neighbors. In the "map phase", the algorithm accepts as input the random k-NN graph, explores the two-hop neighborhood of each node and computes the similarity among each pair, as shown the `Map` procedure. Note the *vertex-centric* nature of the algorithm: each vertex n considers a couple of its neighbours (u,v) and "unselfishly" computes the similarity between them. The result of this computation is then sent to the interested nodes through the `EMIT` operation. In the reduce phase, `Reduce`, each node selects its top-k similar nodes, and produces a new approximated k-NN graph, that is used as an input for the next iteration of the algorithm. Note that `NeighborList` is composed of `(Node, Similarity)` pairs.

In this work we are particularly interested in the role of the number of iterations of the algorithm, which determines its approximation quality. We claim that even rough approximations of the k-NN graph are sufficient for the ultimate goal of our clustering method. Intuitively, the existence of a path between similar items on the k-NN graph is sufficient for the last phase of the clustering algorithm we propose.

B. k-NN Graph Pruning

The iterative procedure to build an approximate k-NN graph may induce neighboring relations among text items that have a low pairwise similarity. Indeed, the algorithm necessarily outputs the k most similar neighbors for each item: any skew in the distribution of the pairwise similarities may produce a k-NN graph in which some nodes are only "loosely" similar. We thus introduce a *pruning phase*, that uses a parameter $\theta \in [0, 1]$ to determine a cut-off similarity value, below which edges between any pair of nodes are eliminated. The pruning phase inspects each node of the k-NN graph, and prunes such edges.

Choosing an appropriate threshold θ determines the final output of our clustering method: as $\theta \to 1$ clustering is *strict*, which leads to a large number of small clusters of essentially identical items; as $\theta \to 0$ clustering is *loose*, leading toward the degenerate case of a single, giant cluster.

C. Phase 2: Connected Components

The second and last phase of our approach uses the pruned k-NN graph, and outputs its connected components, that we use as a proxy for identifying clusters of similar items. Recall that the problem of finding the connected components of a graph amounts to searching for sub-graphs in which any two vertexes are connected to each other by paths.

Finding connected components in large-scale graphs using scalable algorithms is a well studied and understood problem, as shown in the rich literature on the subject [27], [18], [22].

In this work we use a parallel implementation of the Cracker algorithm [22], wherein each node is tagged with the smallest node identifier of its component called seed node identifier. The key idea of Cracker is to reduce the graph size in each step of the computation, by employing a technique inspired by the "contraction algorithm" to compute minimum cuts of a graph [16]. At termination, all nodes tagged with the same seed identifier are grouped in the same component.

IV. Experimental Setup

This section provides details about our experimental setup, including datasets used, evaluation metrics, parameters and system environment.

Experimental platform. All the experiments have been conducted on a cluster running Ubuntu Linux consisting of 17 nodes (1 master and 16 slaves), each equipped with 12 GB of RAM, a 4-core CPU and a 1 Gbit interconnect.

To implement our approach and the baseline method we use for our comparative analysis using Apache Spark [1]: our source code is publicly available[4].

Evaluation Metrics. We now discuss the metrics we use to analyse the parameter space of our approach, and for its global validation in terms of clustering quality. Also, we manually investigate the clusters we obtain, using domain knowledge to evaluate the goodness of clustering.

We study the role of the parameters of our approach using the following metrics:

- **N. of clusters**: measures the number of clusters identified by the clustering algorithm. If not otherwise stated, we only consider clusters to be "useful" if they have more than 1,000 elements;
- **Largest cluster size**: measures the size of the largest cluster identified by the algorithm.

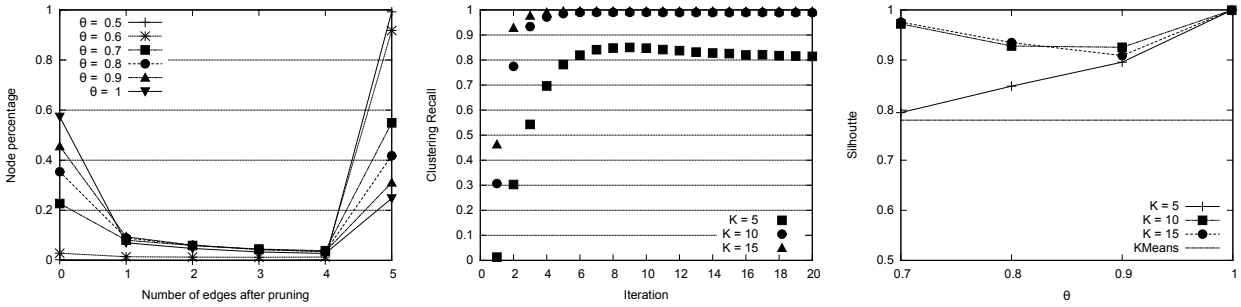
We compute clustering quality using well-known metrics [11], [20], that we report below:

- **Silhouette** [28]: constitutes an aggregate metric, that takes into account the inter- and intra-cluster pairwise similarity between items. Higher values are preferred.
- **Recall**: this metric relates two data clustering obtained by different methods. Using clustering C as a reference, we compute the recall of clustering D by computing the fraction of items that belong to the same cluster in both C and D. In particular, we use as a reference the exact clustering we obtain with the "brute force" approach to compute the k-NN graph. Higher values of recall are preferred.

It is important to notice that computing the above metrics is computationally as hard as computing the clustering we intend

[3] We omit the pseudo-code of this phase, as it is trivial.

[4] https://github.com/alessandrolulli/knnMeetsConnectedComponents



(a) Edges Removed with k=5. (b) Iterations vs. Clustering recall. (c) Silhouette (higher is better).

Fig. 1: Impact of the pruning phase threshold θ. Impact of number of *iterations* for the k-NN graph construction phase of our algorithm. Clustering quality in terms of clustering Silhouette for the *full* Symantec dataset.

to evaluate. For this reason, we resort to uniform sampling: instead of computing the all-to-all pairwise similarity between items, we pick items uniformly at random, with a sampling rate of 1%, increasing it up to 10% for small clusters.

The dataset. The main dataset we use in our evaluation consists of a subset of SPAM emails collected by Symantec Research Labs, between 2010-10-01 and 2012-01-02, which is composed by $3,886,371$ email samples. Each item of the dataset is formatted according to JSON and contains the common features of an email, such as: subject, sending date, geographical information, the bot-net used for the SPAM campaign as labeled by Symantec systems, and many more. For instance, a subject of an email in the dataset is "19.12.2011 Rolex For You -85%" and the sending day is "2011-12-19".

In this work, we are interested in identifying clusters of SPAM emails using subjects alone, as they constitute a compact description of the email.

V. Results

In this section we present our result, and we organize it as follows. First, we analyze the parameter space of our algorithm, and discuss the impact of such parameters on the metrics we defined above. Then, we focus on clustering quality, and compare the performance of our approach to that of the *baseline* algorithm we discuss in section II. Finally, we study the clustering scalability.

Analysis of the parameter space. First, we summarize the parameters underlying our algorithm and discuss about their role. Our approach has 3 main parameters: k, the number of neighbors to construct the k-NN graph; the number of *iterations* of the first phase of the algorithm; and θ, the pruning threshold.

The experimental results we show in this section are obtained with a sampled version of the Symantec dataset, and account for 800,000 data items. A sampled dataset allows us to execute the "brute force" method to compute the k-NN graph.

In what follows, we let the number of *iterations* and θ to be free parameters, and instead select a few representative values for k. We chose k to be small, *i.e.*, we allow a few neighbors per node in the k-NN graph.

Impact of the pruning threshold θ. We now discuss how the pruning mechanism modifies the k-NN graph, and what is the impact on clustering. As discussed in section III, as θ tends to 0, pruning is less effective, and the k-NN graph tends to have a single giant component. Instead, when θ tends to one, only very similar neighbors survive pruning, and the k-NN graph is fractioned in a large number of small clusters.

Figure 1a shows the fraction of nodes for which a given number of edges are removed after pruning, as a function of θ. For values of $\theta < 0.8$, pruning is less effective, as the number of pruned edges is small. Instead, for $\theta > 0.9$, a large fraction of nodes remain with one or fewer edges after the pruning phase. This translates in sizes of the largest clusters to approach the entire dataset, for $\theta = 0.5$ already, or to be extremely small, for $\theta = 1$.

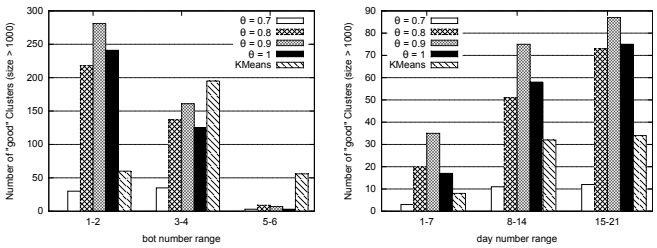
Overall impact of approximation. We now study the impact of the k-NN graph approximation on clustering quality, by analysing the deviation of our approach from the results obtained from an *exact* k-NN graph computed using the "brute force" approach. Our results indicate that approximate k-NN graphs obtained with a low k and few *iterations* are sufficient to obtain data clustering that is practically indistinguishable from that obtained by an onerous $O(n^2)$ k-NN graph construction phase.

Figure 1b shows how clustering recall varies as a function of the k-NN *iterations*. As shown in the Figure, $k = 10$ and 5 *iterations* are sufficient to obtain a clustering which is *essentially identical* to that obtained with the *exact* k-NN graph. Even a very low value of $k = 5$ settles to a 0.8 recall, after roughly 10 *iterations*. Even rough approximations of the k-NN graph, obtained with a small number of *iterations*, are sufficient for the algorithm to stabilize.

Analysis of the clustering quality. We now move to a global evaluation of the algorithm we present in this work, and compare clustering quality to the **baseline** algorithm described in section II. In particular, we use the efficient K-means implementation available in Spark's MLLib package [2], and the `word2vec` package [4], as illustrated in [3]. If not otherwise specified, we set $K = 1000$ such that the baseline algorithm output 1,000 clusters similarly to our

TABLE I: Symantec Dataset: Manual Investigation

cluster size	bot	days	subjects sample
7255	Grum, Unclassified	2011/12/14-2011/12/20	"17.12.2011 Rolex For You -73%" "15.12.2011 Rolex For You -89%" "19.12.2011 Rolex For You -85%"
4512	Rustock, Unclassified	2010/12/04-2010/12/06	"jadevnn, Alena (status-online) invites you for chat." "Hi zmes40, Alena (status-online) invites you for chat." "keumd,,Alena (status-online) invites you for chat."
4412	Rustock, Unclassified	2011/01/28-2011/02/01	"Re: User kilmernn" "Re: User anguinet" "Re: User hudnalli"
4116	Rustock, Unclassified	2011/03/12-2011/03/14	"tdwilkey, you have a new PRIVATE MESSAGE", "dbeltondd, you have a new PRIVATE MESSAGE" "bn, you have a new PRIVATE MESSAGE"
2992	Grum, Unclassified	2011/08/19-2011/08/23	"cseeberd@Amega.com VIAGRA ? 84% consensus!" "Maia@Amega.com VIAGRA ? 50% consensus!" "zelmo38dd@Amega.com VIAGRA ? 16% consensus!"



(a) Bot network identifiers. (b) Days.

Fig. 2: Clustering quality in terms of the unique number of features in each cluster output by clustering algorithms.

approach. Also, this configuration yields the best result in term of Silhouette metric.

We focus on the *full* Symantec dataset. In addition, in what follows and if not otherwise specified, we set the operating parameters of our algorithm as follows: $k = 10$, and 10 *iterations*, which are the parameters that offer a good trade-off between clustering quality, approximation quality, and algorithm runtime.

Figure 1c illustrates that the clustering Silhouette obtained by our approach is superior to the baseline algorithm, and this holds for all parameter choices. This is confirmed also in Figures 2a and 2b, which show the number of "good" clusters (with at least 1,000 items) as determined by domain knowledge metrics. Essentially, these figures report the number of clusters amenable to manual inspection of the results, as a function of features such as the number of bot-nets and the time-frame of a SPAM campaign. For example, in Figure 2a, domain experts can extract valuable information when the number of SPAM bots in a cluster is small, in the 1-2 range: in this case, our approach is superior to the baseline algorithm, which performs slightly better for the less interesting cases of 3-4 and 5-6 bots. Similarly, Figure 2b shows that the number of "good" clusters identified by our approach is always better than that of the baseline algorithm, and this is especially true for the 1-7 range, indicating clusters with emails spanning a 1 week time-frame.

Finally, we proceed with a manual inspection of the clusters we obtain with our approach, to further illustrate the "goodness" of the clustering we achieve, with $k = 10$, 5 *iterations* and $\theta = 0.9$. Table I illustrates a few email samples in clusters where both the number of bot-nets is less or equal to 2, and all emails are sent within one week time-frame. For instance, we obtain a cluster of 7255 emails sent from the Grum bot-net, between 2011/12/14 and 2011/12/20: the subjects of the email are related to a SPAM campaign involving a Rolex discount. Note that subjects are all related, albeit not identical.

TABLE II: Symantec dataset: breakdown of the algorithm runtime (in seconds)

k-NN graph phase	Iteration				
k	5	10	15	20	**CC**
5	675	2293	3185	4897	66
10	2281	4610	5320	7061	81
15	4061	8475	13594	18203	107

TABLE III: Symantec dataset: K-means, baseline algorithm runtime (in seconds)

	K	time
K-means	1000	3498 (1.53×)
	2000	10004 (4.39×)
	3000	28411 (12.46×)
	4000	56008 (24.55×)
Our approach, $k = 10$ and 5 iterations		**2281**

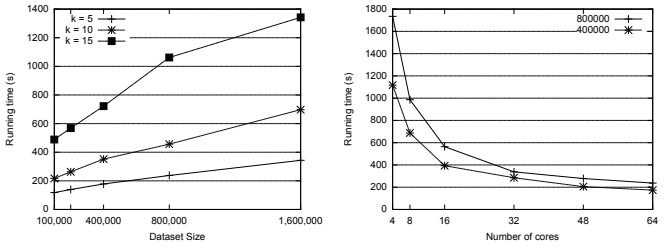
Analysis of algorithm scalability. We study the scalability of our approach and compare it to the baseline algorithm discussed earlier: first, we vary the dataset size maintaining the same number of compute machines that execute the parallel algorithms, then we keep the dataset size constant, and increase the level of parallelism by adding compute machines.

Figure 3a, shows the algorithm runtime with varying dataset sizes, using 5 different samples of the Symantec dataset of size 100,000, 200,000, 400,000, 800,000 and 1,600,000 emails respectively. All values plotted are the average of 5 independent executions. Our results indicate roughly a linear scalability with respect to the size of the dataset, an observation that holds irrespectively of the value of k.

Figure 3b shows the algorithm runtime as the number of cores we devote to the computation varies between 4 and 64, considering datasets 400,000 and 800,000 items; in both cases, our results indicate a quasi-linear speed-up, especially for the biggest dataset. For example, increasing doubling the number of cores from 8 to 16, for the large dataset, cuts almost in half the algorithm runtime.

Finally, Table II, reports the runtime breakdown of the k-NN graph construction phase, and of the connected component phase of our algorithm, for several values of k and for $\theta = 0.9$. Again, all values are the average of 5 independent executions.

As expected, the k-NN graph construction runtime increases both with k and with the *iterations* number, although more slowly than the worst scale asymptotic analysis and experi-



(a) Dataset sizes. (b) Number of cores.

Fig. 3: Scalability. Our approach scales roughly linearly.

mental results presented in [12].[5] Note that the first phase of our approach dominates the overall algorithm runtime, as computing the connected components is fast. Once the k-NN graph is built, it is possible to quickly proceed with various versions of the pruning phase (tuning θ for the application at hand) and obtain different clusters.

Table III illustrates the runtime of the baseline algorithm that uses K-means: the table reports the "slow-down" of the baseline algorithm with respect to our approach, when $k = 10$ and with 5 *iterations*, and for different values of K, the number of clusters K-means constructs. Our approach outperforms the baseline algorithm in terms of end-to-end clustering times, even for small values of K.

VI. Conclusion

Exploratory data analysis requires fundamental techniques to understand, describe and eventually extract value from large amounts of data. One of such techniques is data clustering. In this work, we presented a scalable approach for text data clustering, that accommodates arbitrary similarity measures and that produces high quality clusters.

To overcome the quadratic nature of typical approaches to text clustering, this work studied the role of approximation in establishing a trade-off between high clustering quality and fast algorithmic runtime. We showed, through a detailed experimental campaign, that our method does not require accurate representations of pairwise similarity across data items to produce high quality, interpretable clusters. We supported our claims using real traces covering adversarial applications aiming at identifying SPAM campaigns, and through manual inspection by domain experts of the clusters output by our algorithm.

Our next steps include the design of an LSH-based k-NN graph algorithm supporting arbitrary similarity metrics to push approximation even further, the extension of our method to a density-based approach and, ultimately, to take into account multiple, heterogeneous features of the data.

Acknowldegments

This work has been partially funded by the BigFoot project, in the framework FP7-ICT-ICT-2011.1.2 Call 8, Project No. 317858.

[5]Recall that we use a different parallel execution framework in our work, Spark, which is geared towards efficient execution of iterative algorithms.

References

[1] Apache spark. https://spark.apache.org.
[2] Apache spark machine learning library. https://spark.apache.org/mllib/.
[3] Clustering the News with Spark and MLLib. http://bigdatasciencebootcamp.com/posts/Part_3/clustering_news.html.
[4] Word2vector package. https://code.google.com/p/word2vec/.
[5] C. C. Aggarwal and C. Zhai. *Mining text data*. Springer Science & Business Media, 2012.
[6] H. Becker et al. Beyond trending topics: Real-world event identification on twitter. In *Proc. of ICWSM*, 2011.
[7] F. Beil et al. Frequent term-based text clustering. In *Proc. of ACM SIGKDD*, 2002.
[8] R. Bosagh-Zadeh and A. Goel. Dimension independent similarity computation. In *Journal of Machine Learning Research*, 2012.
[9] T. Debatty et al. Building k-nn graphs from large text data. In *Proc. of IEEE BigData*, 2014.
[10] T. Debatty et al. Scalable graph building from text data. In *Proc. ACM BigMine*, 2014.
[11] B. Desgraupes. Clustering indices. *University of Paris Ouest*, 2013.
[12] W. Dong et al. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proc. of ACM WWW*, 2011.
[13] S. Fortunato. Community detection in graphs. *Physics Reports*, 486, 2010.
[14] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7), 1995.
[15] J. Ji et al. Super-bit locality-sensitive hashing. In *Proc. of NIPS*, 2012.
[16] D. R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30, 1993.
[17] G. Karypis and E.-H. S. Han. Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In *Proc. of ACM CIKM*, 2000.
[18] R. Kiveris et al. Connected components in mapreduce and beyond. In *Proc. of ACM SOCC*, 2014.
[19] F. Lin and W. W. Cohen. A very fast method for clustering big text datasets. In *ECAI*, pages 303–308, 2010.
[20] Y. Liu et al. Understanding of internal clustering validation measures. In *Proc. of IEEE ICDM*, 2010.
[21] S. P. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2), 1982.
[22] A. Lulli et al. Cracker: Crumbling large graphs into connected components. In *Proc. of IEEE ISCC*, 2015.
[23] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1), 2004.
[24] T. Mikolov et al. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS*, 2013.
[25] M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3), 2006.
[26] A. Rajaraman et al. *Mining of massive datasets*, volume 77. Cambridge University Press Cambridge, 2012.
[27] V. Rastogi et al. Finding connected components in map-reduce in logarithmic rounds. In *Proc. of IEEE ICDE*, 2013.
[28] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 1987.
[29] Y. Shen et al. Mr-triage: Scalable multi-criteria clustering for big data security intelligence applications. In *Proc. of IEEE BigData*, 2014.
[30] M. Steinbach et al. A comparison of document clustering techniques. In *Proc. of KDD workshop on text mining*, 2000.
[31] O. Thonnard and M. Dacier. A strategic analysis of spam botnets operations. In *Proc. of ACM CEAS*, 2011.
[32] Z. Toh and W. Wang. Dlirec: Aspect term extraction and term polarity classification system. In *Proc. of SemEval*, 2014.
[33] W. E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*, 1999.
[34] Y.-m. Zhang et al. Fast knn graph construction with locality sensitive hashing. In *Proc. of ECML PKDD*, 2013.
[35] Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Journal of Machine Learning*, 55(3), 2004.