SAIL: Summation-bAsed Incremental Learning for Information-Theoretic Text Clustering

Jie Cao, Zhiang Wu, Junjie Wu, *Member, IEEE*, and Hui Xiong, *Senior Member, IEEE*

***Abstract*—Information-theoretic clustering aims to exploit information-theoretic measures as the clustering criteria. A common practice on this topic is the so-called Info-Kmeans, which performs K-means clustering with KL-divergence as the proximity function. While expert efforts on Info-Kmeans have shown promising results, a remaining challenge is to deal with high-dimensional sparse data such as text corpora. Indeed, it is possible that the centroids contain many zero-value features for high-dimensional text vectors, which leads to infinite KL-divergence values and creates a dilemma in assigning objects to centroids during the iteration process of Info-Kmeans. To meet this challenge, in this paper, we propose a Summation-bAsed Incremental Learning (SAIL) algorithm for Info-Kmeans clustering. Specifically, by using an equivalent objective function, SAIL replaces the computation of KL-divergence by the incremental computation of Shannon entropy. This can avoid the zero-feature dilemma caused by the use of KL-divergence. To improve the clustering quality, we further introduce the variable neighborhood search scheme and propose the V-SAIL algorithm, which is then accelerated by a multithreaded scheme in PV-SAIL. Our experimental results on various real-world text collections have shown that, with SAIL as a booster, the clustering performance of Info-Kmeans can be significantly improved. Also, V-SAIL and PV-SAIL indeed help improve the clustering quality at a lower cost of computation.**

***Index Terms*—Information-theoretic clustering, KL-divergence, K-means distance, multithreaded parallel computing, variable neighborhood search (VNS).**

Manuscript received December 11, 2011; revised April 18, 2012 and July 28, 2012; accepted July 29, 2012. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 71072172 and 61103229, in part by the Key Project of Natural Science Research in Jiangsu Provincial Colleges and Universities under Grant 12KJA520001, and in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK2012863. The work of J. Wu was supported in part by NSFC under Grants 70901002, 71171007, 71031001, and 70890080, in part by the Foundation for the Author of National Excellent Doctoral Dissertation of PR China under Grant 201189, in part by the Program for New Century Excellent Talents in University, and in part by the Ph.D. Programs Foundation of Ministry of Education of China under Grant 20091102120014. The work of H. Xiong was supported in part by the NSFC under Grant 71028002, and in part by the National Science Foundation under Grants CCF-1018151 and IIP-1069258. *(Corresponding author: J. Wu.)* A preliminary version of this work has been published as a short paper in ACM SIGKDD 2008 [1]. This paper was recommended by Editor L. Shao.

J. Cao and Z. Wu are with Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing 210003, China (e-mail: caojie690929@163.com; zawuster@gmail.com).

J. Wu is with Beijing Key Laboratory of Emergency Support Simulation Technologies for City Operations, the School of Economics and Management, Beihang University, Beijing 100191, China (e-mail: wujj@buaa.edu.cn).

H. Xiong is with the Management Science and Information Systems Department, Rutgers University, NJ 07102 USA (e-mail: hxiong@rutgers.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2012.2212430

I. Introduction

CLUSTER ANALYSIS is a fundamental task in various domains such as data mining [2], information retrieval [3], image and video processing [4]–[7], etc. Recent years have witnessed an increasing interest in information-theoretic clustering [8]–[11], since information theory [12] can be naturally adapted as the guidance for the clustering process. For instance, the clustering analysis can be treated as the iteration process of finding a best partition on data in a way such that the loss of mutual information due to the partitioning is the least [8].

This paper is focused on the problem of K-means clustering with KL-divergence [13] as the proximity function, which is called Info-Kmeans. To better understand the theoretic foundation of Info-Kmeans, we present an organized study of two different views on the objective functions of Info-Kmeans. First, we derive the objective function of Info-Kmeans from a probabilistic view. In this regard, we know that the probabilistic view takes several assumptions on data distributions, and the goal of Info-Kmeans is to maximize the likelihood function on multinomial distributions. In contrast, the information-theoretic view has no prior assumption on data distributions. In this case, the objective function of Info-Kmeans is to find the best partition on data so that the loss of mutual information is minimized. The above indicates that the information-theoretic view on Info-Kmeans is more appealing, since we do not need to make any assumption on data distributions. As a result, in this paper, we take the information-theoretic view on Info-Kmeans.

While Info-Kmeans has the sound theoretic foundation, there are some challenging issues with it from a practical viewpoint. For example, people have shown that, for text clustering, the performance of Info-Kmeans is poorer than that of the spherical K-means, which has the cosine similarity as the proximity function [10] and is the de facto benchmark of text clustering. Indeed, for high-dimensional sparse text vectors, Info-Kmeans often has some difficult scenarios. For example, the centroids in sparse data usually contain many zero-value features. This creates infinite KL-divergence values, which leads to a challenge in assigning objects to the centroids during the iteration process of Info-Kmeans. A traditional way to handle this zero-feature dilemma is to smooth the sparse data by adding a very small value to every instance in the data [14]. In this way, there is no instance having zero feature values. While this smoothing method can avoid the zero-feature dilemma for Info-Kmeans, it can also degrade the clustering performance, since the real values and the sparseness of data have been changed [15], [16].

As an alternative to the smoothing technique, we propose a Summation-bAsed Incremental Learning (SAIL) algorithm for

Info-Kmeans clustering in this paper. Our main contributions lie in the following aspects. First, SAIL can avoid the zero-feature dilemma of Info-Kmeans by replacing the computation of KL-divergence for the "instance—centroid" distances, by the incremental computation of the Shannon entropy for the centroids alone. Second, the distance computation is further refined to make use of the sparseness of text vectors to improve the efficiency of SAIL. Third, the variable neighborhood search (VNS) meta-heuristic [17], [18] is introduced to improve the clustering quality of SAIL, and the resulting V-SAIL variant is further speeded up by a multithreaded parallel computing scheme, which results in the novel PV-SAIL variant.

Our experimental results on various real-world text corpora have shown that, with SAIL as a booster, the clustering performance of Info-Kmeans can be significantly improved. Indeed, for most of the text collections, SAIL produces clustering results competitive to or better than the state-of-the-art spherical K-means algorithm in CLUTO [19]. Some settings such as feature weighting, instance weighting, and bisecting have been shown to have varied effects on SAIL, but SAIL without these settings shows more robust results. Moreover, V-SAIL further improves the clustering quality of SAIL by searching around the neighborhood of the solution using the VNS scheme, and PV-SAIL effectively lowers the high computational cost of V-SAIL by using the multithreaded parallel computation.

The remainder of this paper is organized as follows. Section II gives two theoretical views of Info-Kmeans. Section III introduces the so-called zero-feature dilemma in Info-Kmeans clustering. Sections IV and V introduce in detail the SAIL algorithm and the two variants: V-SAIL and PV-SAIL. Section VI shows the experimental results. Finally, we present related work in Section VII and conclude our work in Section VIII.

II. Theoretical Overviews of Info-Kmeans

To better understand the theoretical foundation of Info-Kmeans, in this section, we provide an organized study of two different views, i.e., the probabilistic view and the information-theoretic view, on the objective function of Info-Kmeans.

A. Global Objective of Info-Kmeans

K-means [20] is a prototype-based, simple partitional clustering technique which attempts to find the user-specified K clusters. These clusters are represented by their centroids (a cluster centroid is typically the arithmetic mean of the data objects in that cluster). K-means has an objective function:

$$obj: \quad \min \sum_{i=1}^{K} \sum_{x \in c_i} \pi_x \text{dist}(x, m_i) \tag{1}$$

where c_i denotes cluster i, m_i is the centroid of c_i, $\text{dist}(\cdot)$ is the distance function, and π_x is the weight of x given $\sum_x \pi_x = 1$. To reach the minimum, K-means employs a heuristic clustering process as follows. First, K initial centroids are selected. Then, the two-phase iterations are launched. In the first phase, every point in the data is assigned to the closest centroid, and each collection of points assigned to a centroid forms a cluster; then in the second phase, the centroid of each cluster is updated based on the points assigned to the cluster. This process is repeated until no point changes clusters.

As we know, different distance functions lead to different types of K-means. Our focus in this paper is on Info-Kmeans. Let $D(x\|y)$ denote the KL-divergence [13] between two discrete distributions x and y, we have

$$D(x\|y) = \sum_i x_i \log \frac{x_i}{y_i}. \tag{2}$$

It is easy to observe that, in most cases, $D(x\|y) \neq D(y\|x)$, and $D(x\|y) + D(y\|z) \geq D(x\|z)$ cannot be guaranteed. Hence, D is not a *metric*. If we let "dist" be D in (1), we have the objective function of Info-Kmeans as follows:

$$obj: \quad \min \sum_k \sum_{x \in c_k} \pi_x D(x\|m_k) \tag{3}$$

where each instance x has been normalized to a discrete distribution before clustering. To further understand Info-Kmeans, we take two different views on (3) as follows.

B. Probabilistic View of Info-Kmeans

In this subsection, we first derive the objective function of Info-Kmeans from a probabilistic view. Specifically, the objective function can be derived by maximizing the "partitioned" likelihood function of the EM algorithm, i.e., the crisp version of EM [10].

Assume that we have a text collection $\mathbb{D}$, which consists of K crisp partitions in multinomial distributions with different parameters, i.e., $\theta_1, \ldots, \theta_K$, respectively. Let random variables X and Y denote the text and the term, respectively. Let $n(x, y)$ denote the number of occurrences of term y in document x, and $n(x) = \sum_y n(x, y)$. Then, we have

Theorem 1: Let $L = P(\mathbb{D}|\Theta) = \Pi_x p(x|\Theta)$ be the likelihood function. Let $B = \sum_x n(x)$, and $A = -\sum_x n(x) H(p(Y|x))$, where H is the Shannon entropy [12]. Then, we have

$$\frac{A - \log L}{B} = \sum_k \sum_{x \in c_k} p(x) D\left(p(Y|x) \| p(Y|\theta_k)\right) \tag{4}$$

where $p(x) = n(x)/\sum_x n(x)$ and $p(y|x) = n(x, y)/n(x)$.

Proof: By definition

$$\begin{aligned} \log L = \sum_x \log p(x|\Theta) &\overset{a}{=} \sum_k \sum_{x \in c_k} \log p(x|\theta_k) \\ &\overset{b}{=} \sum_k \sum_{x \in c_k} \sum_y n(x, y) \log\left(p(y|\theta_k)\right) \\ &= \sum_k \sum_{x \in c_k} n(x) \sum_y p(y|x) \log p(y|\theta_k) \end{aligned}$$

where "a" reflects the "crisp" property of the modified EM model, and "b" follows the multinomial distribution, i.e., $p(x|\theta) = \Pi_y p(y|\theta)^{n(x,y)}$.

Meanwhile, A can be transformed into

$$A = \sum_k \sum_{x \in c_k} n(x) \sum_y p(y|x) \log p(y|x).$$

If we substitute the transformed A and $\log L$ into the left-hand side of (4), we can easily get the right-hand side. Hence, we complete the proof. □

Let us compare (4) with (3). If we let $\pi_x \equiv p(x)$, $x \equiv p(Y|x)$, and $m_k \equiv p(Y|\theta_k)$, we have $obj \Leftrightarrow \min(A - \log L)/B \Leftrightarrow \max \log L$. This implies that, if we take the probabilistic view of the objective function, Info-Kmeans aims to maximize the likelihood function on multinomial distributions. This probabilistic view of Info-Kmeans requires a series of assumptions: $p(x) = n(x)/\sum_x n(x)$, $p(y|x) = n(x,y)/n(x)$, and $p(y|\theta_k) = p(x)p(y|x)/\sum_{x \in c_k} p(x)$. However, in the experimental section, we will show these assumptions may degrade the performance of Info-Kmeans.

C. Information-Theoretic View of Info-Kmeans

Here, we derive the objective function in (3) from an information-theoretic point of view. We begin by introducing an important lemma as follows. Given a set of discrete probabilistic distributions $\{p_1, p_2, \ldots, p_n\}$ and the weights $\{\pi_1, \pi_2, \ldots, \pi_n\}$, we have

Lemma 1 (Cover&Thomas, 2006):

$$\sum_{i=1}^{n} \pi_i D\left(p_i \| \sum_{i=1}^{n} \pi_i p_i\right) = H\left(\sum_{i=1}^{n} \pi_i p_i\right) - \sum_{i=1}^{n} \pi_i H(p_i). \quad (5)$$

Now, given a text collection $\mathbb{D}$, we want to partition $\mathbb{D}$ into K clusters without overlapping. Let random variables X, Y, and C denote the text, the term, and the cluster, respectively. Let x, y, and c be the corresponding instances with $p(x)$, $p(y)$, and $p(c)$ being the probabilities of occurrences. Furthermore, we assume that $p(c) = \sum_{x \in c} p(x)$. Then, we have the following theorem:

Theorem 2: Let $I(X,Y)$ be the mutual information between two random variables X and Y, then

$$I(X,Y) - I(C,Y) = \sum_k \sum_{x \in c_k} p(x) D\left(p(Y|x) \| p(Y|c_k)\right). \quad (6)$$

The proof of Theorem 2 can be found in [9]. Theorem 2 formulates the information-theoretic view of Info-Kmeans; that is, Info-Kmeans tries to find the best partition on data so that the loss of mutual information due to the partitioning is minimized.

D. Discussions

In summary, we have two different views on Info-Kmeans, as described by (6) and (4), respectively. Both views lay the theoretic foundations of Info-Kmeans. Nonetheless, the information-theoretic view seems to be more appealing, since there is no prior assumption for $p(x)$ and $p(x|c)$, which is, however, crucial for the probabilistic view. In fact, we can regard the probabilistic framework as a special case of the information-theoretic framework of Info-Kmeans. Moreover, we will show in experimental results that in most cases the assumption of $p(x)$ and $p(x|c)$ in the probabilistic view is harmful to the clustering accuracy.

III. Dilemma of Info-Kmeans

Though having sound theoretical foundations, Info-Kmeans has long been criticized for having performance inferior to the spherical K-means [21] on text clustering [10]. However, in this section, we highlight an implementation challenge of Info-Kmeans. We believe this challenge is one of the key factors that degrade the clustering performance of Info-Kmeans.

Assume that we use Info-Kmeans to cluster a text corpus. To optimize the objective in (3), we launch the two-phase iteration process—reassigning text vectors to the "nearest" centroids and updating the centroids according to the newly assigned text vectors subsequently. To this end, we must compute the KL-divergence between each text vector $p(Y|x)$ and each centroid $p(Y|c_k)$. In practice, we usually let

$$p(Y|x) = \frac{x}{n(x)} \qquad p(Y|c_k) = \frac{\sum_{x \in c_k} p(x) p(Y|x)}{\sum_{x \in c_k} p(x)}$$

where $n(x)$ is the sum of all the term frequencies of x, $p(x)$ is the weight of x, as in (4) and (6). Therefore, by (2), to compute $D(p(Y|x) \| p(Y|c_k))$, we should expect that all the feature values of x are positive real numbers. Unfortunately, however, this is not the case for high-dimensional text vectors, which are famous for the sparseness in high dimensionality.

To illustrate this, we observe the computation of KL-divergence in each dimension y. As we know, $D(p(Y|x) \| p(Y|c_k)) = \sum_y p(y|x) \log(p(y|x)/p(y|c_k))$. To simplify the notations, hereinafter, we denote $p(y|x) \log(p(y|x)/p(y|c_k))$ by D_y. Then, the different combinations of $p(y|x)$ and $p(y|c_k)$ values can result in four scenarios as follows:

1) *Case 1*: $p(y|x) > 0$ and $p(y|c_k) > 0$. In this case, the computation of D_y is straightforward, and the result can be any real number.
2) *Case 2*: $p(y|x) = 0$ and $p(y|c_k) = 0$. In this case, we can simply omit this feature, or equivalently let $D_y = 0$.
3) *Case 3*: $p(y|x) = 0$ and $p(y|c_k) > 0$. In this case, $\log(p(y|x)/p(y|c_k)) = \log 0 = -\infty$, which implies that the direct computation is infeasible. However, by the L' Hospital's rule [22], $\lim_{x \to 0^+} x \log(x/a) = 0 (a > 0)$. Thus, we can let $x \doteq p(y|x)$ and $a \doteq p(y|c_k)$, and thus have $D_y = 0$.
4) *Case 4*: $p(y|x) > 0$ and $p(y|c_k) = 0$. In this case, $D_y = +\infty$, which is hard to handle in practice.

We summarize the above four cases in Table I. As can be seen, for Cases 1 and 2, the computation of D_y is logically reasonable. However, the computation of D_y in Case 3 is actually questionable; that is, it cannot reveal any difference between $p(Y|x)$ and $p(Y|c_k)$ in dimension y, although $p(y|c_k)$ may deviate heavily from zero. Also, it implies that the differences of various centroids in dimension y have been omitted.

Nevertheless, the most difficult case to handle is Case 4. On one hand, it is hard to do computations with $+\infty$ in practice. On the other hand, it is obvious that if there is some

TABLE I
FOUR CASES IN KL-DIVERGENCE COMPUTATIONS

Case	i	ii	iii	iv
$p(y\|x)$	>0	0	0	>0
$p(y\|c_k)$	>0	0	>0	0
D_y	$\in \mathbb{R}$	0	0	$+\infty$

dimension y of Case 4, the total KL-divergence of $p(Y|x)$ and $p(Y|c_k)$ is infinite. This does not work for high-dimensional sparse text vectors, because the centroids of such data typically contain many zero-value features. Therefore, we will have big challenges in assigning instances to the centroids. We call this problem the "zero-feature dilemma."

One way to solve the above dilemma is to smooth the sparse data sets. For instance, we can add a very small positive value to the entire data set so as to avoid having any zero feature value. While this smoothing technique facilitates the computations of KL-divergence, it indeed changes the sparseness property of the data sets. We will demonstrate in the experimental section that this method actually degrades the clustering performance of Info-Kmeans.

In summary, there is a need to develop a new implementation scheme for Info-Kmeans which should be able to avoid the zero-feature dilemma.

IV. SAIL ALGORITHM

In this section, we propose a new variant, named SAIL, for Info-Kmeans. To illustrate it, we first present the concept of K-means distance and use it to simplify the objective function of Info-Kmeans. Then, we refine the computations in SAIL to further improve the efficiency. Finally, we present the algorithmic details.

A. SAIL: Theoretical Foundation

We begin by briefly introducing the notion of *K-means distance*. A detailed study of the K-means distance is available in [23], [24].

Definition 1 (K-means Distance): Let $\mathcal{S} \subseteq \mathbb{R}^d$ be a non-empty open convex set. A continuously differentiable function $f : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_+$ is called a K-means distance, if there exists some continuously differentiable convex function $\phi : \mathcal{S} \mapsto \mathbb{R}$ such that

$$f(x, y) = \phi(x) - \phi(y) - (x - y)^t \nabla\phi(y).$$

Note that different ϕ can lead to different instances of K-means distance. Therefore, the K-means distance is actually a family of infinite distance functions. More importantly, the K-means distance is the only choice for the K-means algorithm with centroids of arithmetic means. That is

Theorem 3: Let $\mathcal{S} \subseteq \mathbb{R}^d$ be a non-empty open convex set. Assume $f : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}_+$ is a continuously differentiable function satisfying: 1) $f(x, x) = 0$, $\forall x \in \mathcal{S}$ and 2) $f_{y_j}(x, y)$ is continuously differentiable on x_l, $1 \le j, l \le d$. Then, f can be used for K-means with centroids of arithmetic means, if and only if f is a K-means distance.

TABLE II
SOME K-MEANS DISTANCES

$\phi(x)$	dom(ϕ)	$f(x, y)$	Distance
$\|x\|^2$	$\mathbb{R}^d$	$\|x - y\|^2$	squared Euclidian distance
$\|x\|$	$\mathbb{R}^d_{++}$	$\|x\| - x^t y/\|y\|$	cosine distance
$-H(x)$	$\mathbb{R}^d_{++}$	$D(x\|\|y)$	KL-divergence

Note: $x \in \mathbb{R}^d_{++}$ means: $x_l \ge 0$, $\forall 1 \le l \le d$, and $x \ne \mathbf{0}$.

The details and proofs can be found in [23]. Note that we have special interests in K-means with centroids of arithmetic means because such K-means is much simpler and more efficient than K-means with centroids of medians or medoids. These merits are particularly important for large-scale, high-dimensional text clustering. It is also interesting to note that the K-means distance is a family of distance functions with different ϕ, including the well-known Bregman divergence induced by strictly convex ϕ [25]. We list some popular K-means distances in Table II.

In Table II, the squared Euclidean distance is the most widely used for a variety of clustering applications [2]. The cosine distance, derived from a convex but not strictly convex ϕ, is equivalent to the cosine similarity used for the so-called spherical K-means [21], which is often considered as the state-of-the-art method for text clustering [19]. Our focus in this paper, i.e., the KL-divergence for Info-Kmeans, also belongs to the family of K-means distance. Specifically, according to Definition 1, KL-divergence can be rewritten as

$$D(x\|y) = -H(x) + H(y) + (x - y)^t \nabla H(y). \tag{7}$$

Based on $D(x\|y)$ in (7), we now lay the theoretical foundation of the SAIL algorithm, a new variant of Info-Kmeans. Specifically, we have the following theorem:

Theorem 4: Given $p(c_k) = \sum_{x \in c_k} p(x)$, the objective function of Info-Kmeans:

$$O_1 : \min \sum_k \sum_{x \in c_k} p(x) D\left(p(Y|x)\|p(Y|c_k)\right)$$

is equivalent to

$$O_2 : \min \sum_k p(c_k) H\left(p(Y|c_k)\right). \tag{8}$$

Proof: By (7), we have

$$D\left(p(Y|x)\|p(Y|c_k)\right) = -H\left(p(Y|x)\right) + H\left(p(Y|c_k)\right) + \left(p(Y|x) - p(Y|c_k)\right)^t \nabla H\left(p(Y|c_k)\right).$$

As a result

$$\sum_k \sum_{x \in c_k} p(x) D\left(p(Y|x)\|p(Y|c_k)\right) = \underbrace{\sum_k p(c_k) H\left(p(Y|c_k)\right)}_{(a)} - \underbrace{\sum_x p(x) H\left(p(Y|x)\right)}_{(b)} - \underbrace{\sum_k \sum_{x \in c_k} p(x) \left(p(Y|x) - p(Y|c_k)\right)^t \nabla H\left(p(Y|c_k)\right)}_{(c)}.$$

Since $p(Y|c_k) = \sum_{x \in c_k} p(x)p(Y|x)/p(c_k)$, we have

$$\sum_{x \in c_k} p(x) \left(p(Y|x) - p(Y|c_k)\right) = 0.$$

Accordingly

$$(c) = \sum_k \nabla H\left(p(Y|c_k)\right)^t \sum_{x \in c_k} p(x) \left(p(Y|x) - p(Y|c_k)\right) = 0.$$

Moreover, (b) is a constant given the data set and the weights for the instances. Thus, the goal of O_1 is equivalent to minimize (a), which completes the proof. □

Note that the equivalent objective O_2 given in (8) is right the objective function of SAIL. In other words, Theorem 4 reduces the computation of KL-divergence between instances and centroids to the computation of Shannon entropy of centroids only. As the zero features of a centroid can be **safely skipped** during the entropy computation, SAIL can avoid the zero-feature dilemma in information-theoretic clustering of high-dimensional sparse texts. Some real-world examples can be found in the experimental section below. For instance, Tables IV and V give four real-world text collections with poor clustering results due to the zero-feature dilemma of Info-Kmeans. In contrast, Tables VI and VII show the significantly improved clustering quality using SAIL on the same text collections, which clearly verifies the effectiveness of SAIL in solving the zero-feature dilemma. More details can be found in Section VI.

B. SAIL: Computational Issues

Now, based on the objective function in (8), we establish the computational scheme for SAIL. The major concern here is the efficiency issue.

Generally speaking, SAIL is a greedy scheme which updates the objective-function value "instance by instance." That is, SAIL first selects an instance from the text collection and assigns it to the most suitable cluster. Then, the objective-function value and other related variables are updated immediately after the assignment. The process will be repeated until some stopping criterion is met.

Apparently, to find the suitable cluster is the critical point of SAIL. To illustrate this, suppose SAIL randomly selects $p(Y|x')$ from a cluster $c_{k'}$. Then, if we assign $p(Y|x')$ to cluster c_k, the change of the objective-function value will be (see equation at the bottom of the page) where $(a) - (b)$ and $(c) - (d)$ represent the two parts of changes on the objective-function value due to the movement of $p(Y|x')$ from cluster $c_{k'}$ to cluster c_k. Then, x' will be assigned to the cluster c with the smallest Δ, i.e., $c = \arg\min_k \Delta_k$.

The computation of Δ_k in (9), shown at the bottom of the page, has two appealing properties. First, it only relates to the changes that occurred within the two involved clusters c_k and $c_{k'}$. Other clusters remain unchanged and thus have no contribution to Δ_k. Second, the computations of both $\sum_{x \in c} p(x)$ and $\sum_{x \in c} p(x)p(Y|x)$ have additivity, which can facilitate the computation of Δ_k. Indeed, these two summations, incrementally updated during the clustering, are the key elements of SAIL.

The computation of Δ_k in (9), however, still suffers from the high costs of computing Shannon entropy in (a) or (c). Let us take the entropy computation in (a) for example. Since the denominator changes from $p(c_{k'})$ to $p(c_{k'}) - p(x')$, every dimension in the numerator $\sum_{x \in c_{k'}} p(x)p(Y|x) - p(x')p(Y|x')$ will have a new value, which requires us to compute the logarithm for each dimension. These computations are indeed a huge cost for text vectors in high dimensionality. Therefore, here comes the question: Can we make use of the high sparseness of text vectors to further improve the computational efficiency of SAIL?

The answer is YES. To illustrate this, recall SAIL's objective function O_2 in (8). Let $S(k, y)$ denote $\sum_{x \in c_k} p(x)p(y|x)$. Since $p(Y|c_k) = \sum_{x \in c_k} p(x)p(Y|x)/p(c_k)$, we have

$$\begin{aligned}
&\sum_k p(c_k) H\left(p(Y|c_k)\right) \\
&\quad = -\sum_k \sum_y \sum_{x \in c_k} p(x)p(y|x) \\
&\qquad \times \left(\log \sum_{x \in c_k} p(x)p(y|x) - \log p(c_k)\right) \\
&\quad = \sum_k \left(\sum_{x \in c_k} p(x) \sum_y p(y|x)\right) \log p(c_k) \\
&\qquad - \sum_k \sum_y S(k, y) \log S(k, y) \\
&\quad = \sum_k p(c_k) \log p(c_k) - \sum_k \sum_y S(k, y) \log S(k, y). \qquad (10)
\end{aligned}$$

$$\begin{aligned}
\Delta_k &= O_2(\text{new}) - O_2(\text{old}) \\
&= \underbrace{\left(p(c_{k'}) - p(x')\right) H\left(\frac{\sum_{x \in c_{k'}} p(x)p(Y|x) - p(x')p(Y|x')}{p\left(c_{k'}\right) - p(x')}\right)}_{(a)} - \underbrace{p(c_{k'}) H\left(\frac{\sum_{x \in c_{k'}} p(x)p(Y|x)}{p(c_{k'})}\right)}_{(b)} \\
&\quad + \underbrace{\left(p(c_k) + p(x')\right) H\left(\frac{\sum_{x \in c_k} p(x)p(Y|x) + p(x')p(Y|x')}{p(c_k) + p(x')}\right)}_{(c)} - \underbrace{p(c_k) H\left(\frac{\sum_{x \in c_k} p(x)p(Y|x)}{p(c_k)}\right)}_{(d)} \qquad (9)
\end{aligned}$$

Accordingly, if we move x' from cluster $c_{k'}$ to cluster c_k, the change of the objective-function value will be (See equation at the bottom of the page) where $S^+(k,y) = S(k,y) + p(x')p(y|x')$, and $S^-(k',y) = S(k',y) - p(x')p(y|x')$.

According to (11), shown at the bottom of the page, only the non-empty features of $p(Y|x')$ have contributions to Δ_k in (a) or (b), and thus will trigger the expensive computations of logarithm. Considering that a text vector $p(Y|x')$ is often very sparse, i.e., has many empty features, the computational saving due to (11) will be significant. As a result, we adopt (11) rather than (9) for the computation of Δ_k in SAIL and give the comparative results in the experimental section.

DISCUSSION. It is clear that SAIL differs from the traditional K-means. Indeed, SAIL is an incremental algorithm while the traditional K-means usually employs the batch-learning mode. Furthermore, SAIL also differs from the traditional incremental K-means; that is, to decide the assignment of each selected instance, SAIL does not compute the KL-divergence values between the instance and all the centroid vectors. Instead, it computes and updates the Shannon entropies of the centroids. This computation is supported by the two incrementally maintained summations for each cluster $c : p(c) = \sum_{x\in c} p(x)$ and $p(Y|c) = \sum_{x\in c} p(x)p(Y|x)$. That is why we call this method the SAIL algorithm.

C. SAIL: Algorithmic Details

In this subsection, we present the main process and the implementation details of SAIL. Figs. 1 and 2 show the pseudocodes of the SAIL algorithm.

Line 1 in Fig. 1 is for data initialization. First of all, text collection $\mathbb{D}$ is loaded into the memory. To save the space, we use the triple $\langle rowID, columnID, value\rangle$ to record each non-empty element in $\mathbb{D}$. To fast locate the triples of an instance, we further use an array to store the index of the first feature of each instance. There are two methods for assigning the weights of instances; that is, $\pi_x = n(x)/\sum_x n(x)$, or simply, $\pi_x = 1$. The second one is much simpler and is the default setting in our experiments. The preprocessing of $\mathbb{D}$ includes the row and column modeling, e.g., $tf - idf$, to smooth the instances and assign weights to the features. Then, we normalize the instance x to $x = p(Y|x)$ where $p(y|x) = n(x,y)/n(x)$ for each feature y.

Lines 2–5 in Fig. 1 show the clustering process. Line 3 is for initialization, where $label_{n\times 1}$ contains the cluster labels of all instances, and $cluSum_{K\times(d+1)}$ stores the summations of the weights and weighted instances in each cluster. That is,

$[objVal^*, label^*, \pi]$ =**SAIL**$(\mathbb{D}, K, reps, maxIter)$

Input: $\mathbb{D}$: text collection
K: the number of clusters
$reps$: the number of repeated clusterings
$maxIter$: the max number of iterations in a clustering
Output: $objVal^*$: the optimal objective-function value
$label^*$: the cluster labels of documents
π: the vector of document weights
Variable: $cluSum$: storing $\sum_{x\in c_k}\pi_x$ and $\sum_{x\in c_k}\pi_x x$, $\forall\ k$

Procedure
1. Load and Preprocess $\mathbb{D}$, and $\forall\ x \in \mathbb{D}$, normalize x;
2. **for** $i = 1 : reps$
3. Initialize $label[i]$, then $objVal[i]$ and $cluSum[i]$;
4. LocSearch$(\mathbb{D}, \pi, objVal[i], label[i], cluSum[i], maxIter)$;
5. **end for**
6. $t = \arg\min_i objVal[i]$;
7. **return** $objVal^* = objVal[t]$, $label^* = label[t]$, π;

Fig. 1. Pseudocodes of SAIL.

LocSearch$(\mathbb{D}, \pi, objVal, label, cluSum, maxIter)$

1. **for** $j = 1 : maxIter$
2. **for** $l = 1 : n$
3. Randomly select without replacement an x from $\mathbb{D}$;
4. **for** $s = 1 : K$
5. $\Delta objVal(k)$ = TestAssign$(para)$;
6. // $para \doteq \langle x, \pi_x, cluSum, label(x), k, objVal\rangle$
7. **end for**
8. $k^* = \arg\min_k(\{\Delta objVal(k), k = 1, \cdots, K\})$;
9. Update $objVal$, $label$ and $cluSum$ accordingly;
10. **if** $label$ is unchanged
11. **break**;
12. **end if**
13. **end for**
14. **end for**

Fig. 2. LocSearch subroutine.

for $k = 1, \ldots, K$, $cluSum(k, 1:d) = \sum_{x\in c_k}\pi_x p(Y|x)$, and $cluSum(k, d+1) = \sum_{x\in c_k}\pi_x$, where n, d, and K are the numbers of instances, features, and clusters, respectively. Two initialization modes are employed in our implementation, i.e., "random label" and "random center" (the default one), and the required variables are then computed.

The "LocSearch" subroutine in Line 4 performs clustering for each instance. As shown in Fig. 2, it traverses all instances at random as a round. In each round, it assigns each instance to the cluster with the heaviest drop of the objective-function value and then updates the values of the related variables. The computational details have been given in Section IV-B.

$$\Delta_k = (p(c_k)+p(x'))\log(p(c_k)+p(x')) - p(c_k)\log p(c_k) + (p(c_{k'})-p(x'))\log(p(c_{k'})-p(x')) - p(c_{k'})\log p(c_{k'})$$
$$+ \underbrace{\sum_{\{y|p(y|x')\neq 0\}} S(k,y)\log S(k,y) - S^+(k,y)\log S^+(k,y)}_{(a)} + \underbrace{\sum_{\{y|p(y|x')\neq 0\}} S(k',y)\log S(k',y) - S^-(k',y)\log S^-(k',y)}_{(b)} \quad (11)$$

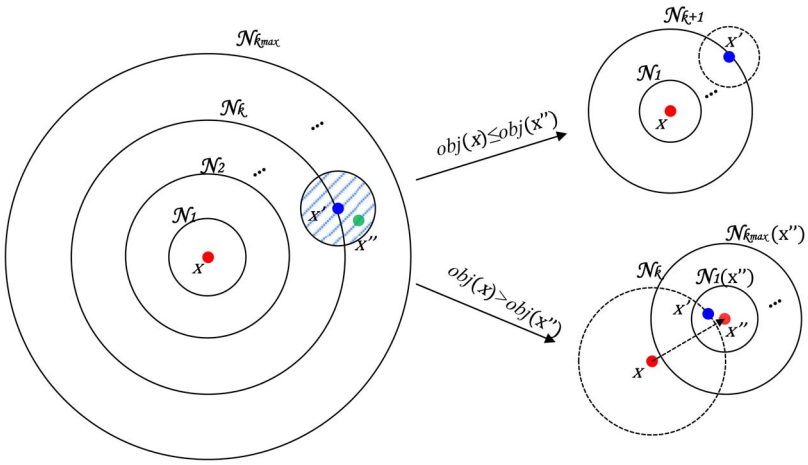

Fig. 3. Illustration of VNS.

Lines 10–12 in Fig. 2 show the stopping criterion in addition to *maxIter*; that is, if no instance changes its label after a round, we stop the clustering. Finally, Lines 6–7 in Fig. 1 choose and return the best clustering result among the *reps* clusterings.

Next, we briefly discuss the convergence issues of SAIL. Since the objective-function value decreases continuously after reassigning each instance, and the combinations of the labels assigned to all instances are limited, SAIL guarantees to converge after limited iterations. However, due to the complexity of the feasible region, SAIL often converges to a local minima or a saddle point. That is why we usually do multiple clusterings in SAIL and choose the one with a lowest objective-function value.

SAIL also preserves the most important advantage of Info-Kmeans—low computational costs. Specially, the space and time requirements are $O((n+K)\bar{d})$ and $O(IKn\bar{d})$, respectively, where I is the number of iterations for convergence (NIC), and $\bar{d}$ is the average number of non-empty features for each instance. As K is often small and I is typically not beyond 20 (please refer to the empirical results in the experimental section), the computational complexity of SAIL is often very low—just as the classic K-means algorithm. Also, by employing (11) rather than (9) for SAIL, we can take advantage of the sparseness of text collections to further improve the efficiency of SAIL.

V. Beyond SAIL: Enhancing SAIL via VNS and Parallel Computing

SAIL introduced in the previous section can be viewed as a combinatorial optimization algorithm. Therefore, like most iterative scheme, SAIL is apt to converge to some local minima or saddle points, particularly for text vectors in high dimensionality. To meet this challenge, further study is in critical need to help SAIL jump out of the inferior points. Here, we propose to use the VNS scheme [17], [18], and establish the VNS-enabled SAIL algorithm: V-SAIL.

VNS is a meta-heuristic for solving combinatorial and global optimization problems. The idea of VNS is to conduct a systematic change of neighborhood within the search [18]. Fig. 3 schematically illustrates the search process of VNS. That is, VNS first finds an initial solution x and then shakes in the kth neighborhood $\mathcal{N}_k$ to obtain x'. Then, VNS centers the search around x' and finds the local minimum x''. If x'' is better than x, x is replaced by x'', and VNS starts to shake in the first neighborhood of x''. Otherwise, VNS continues to search in the $(k+1)$th neighborhood of x. The set of neighborhoods are often defined by metric function in the solution space, e.g., Hamming distance, Eular distance, k-OPT operator, etc. As VNS is often time-consuming, to set stopping conditions is very important, which can be the size of the neighborhood set, the maximum CPU time, and/or the maximum number of iterations.

$[objVal^*, label^*, \pi]$ =**V-SAIL**$(para, k_{max})$

Input: $para \doteq \langle \mathbb{D}, K, reps, maxIter \rangle$, as for SAIL
k_{max}: the number of neighborhood structures
Output: $objVal^*$: the optimal objective-function value
$label^*$: the cluster labels of documents
π: the vector of document weights
Variable: $\mathcal{N}_i$: the ith neighborhood with a Hamming distance $i \times \|\mathbb{D}\|/k_{max}$ from the current label

Procedure
1. $[objVal, label, \pi]$ = SAIL$(\mathbb{D}, K, reps, maxIter)$, $i = 1$;
2. **while** $(i++ \leq k_{max})$
3. $label' =$ Shaking$(\mathcal{N}_i, label)$;
4. Update $objVal'$ and $cluSum'$ accordingly;
5. LocSearch$(\mathbb{D}, \pi, objVal', label', cluSum', maxIter)$;
6. **if** $objVal' < objVal$
7. $objVal = objVal'$, $label = label'$, $i = 1$;
8. **end if**
9. **if** the stopping condition is met
10. **break**;
11. **end if**
12. **end while**
13. **return** $objVal^* = objVal$, $label^* = label$, π;

Fig. 4. Pseudocodes of V-SAIL.

A. V-SAIL Algorithm

Fig. 4 shows the pseudocodes of the V-SAIL algorithm. Generally speaking, V-SAIL is a two-stage algorithm employing

a **c**lustering step and a **r**efinement step. In **c**-step, the SAIL algorithm is called to generate a clustering result. This result serves as the starting point of the subsequent **r**-step, which employs the VNS scheme to refine the clustering result to a more accurate one. It is interesting to note that the "LocSearch" subroutine of SAIL is also called in VNS as a heuristic method to search for local minima. Some important details are as follows.

Lines 2–12 in Fig. 4 describe the **r**-step of V-SAIL. In Line 3, for the ith neighborhood $\mathcal{N}_i$, the "Shaking" function is called to generate a solution $label'$ in $\mathcal{N}_i$ that has a Hamming distance $H_i = i \times |\mathbb{D}|/k_{max}$ to the current solution $label$. More specifically, "Shaking" first randomly selects H_i instances and then changes their labels at random in $label$, which results in a new solution $label'$. Apparently, $label'$ tends to deviate $label$ more heavily as the increase of i. The idea is that once the best solution in a large region has been found, it is necessary to explore an improved one far from the incumbent solution.

In Line 5, the "LocSearch" subroutine of SAIL is called to search for a local minimum solution initialized on $label'$. This well demonstrates that SAIL is not only a clustering algorithm, but also a combinatorial optimization method. Then, in Lines 6–8, if the minimum is smaller than the current $objVal$, we update the related variables, and set the solution as the new starting point of VNS. The stopping condition in Line 9 is very simple; that is, we stop VNS either if a maximum repeat-time of calling "Shaking" is met or when a given CPU time is due.

In summary, V-SAIL well combines the complemental advantages of SAIL and VNS. That is, VNS can help SAIL avoid inferior solutions, while SAIL can help VNS fast locate a good local minimum. We will show the empirical results in the experimental section.

B. PV-SAIL Algorithm

V-SAIL may find a better clustering result by searching inside the neighborhoods via the VNS scheme. The critical problem is, however, VNS usually has a high computational cost. As indicated by Fig. 4, V-SAIL searches inside the neighborhoods one by one and gets back to the first neighborhood after finding a better solution. This can make the computational time uncontrollable. That is why we introduce a "hard" stopping criterion in Line 9. However, the problem remains unsolved in another way—given a time constraint, V-SAIL can only search a limited number of neighborhoods, which prevents it from further enhancing the clustering accuracy.

To meet this challenge, we propose a multithreaded scheme for V-SAIL: PV-SAIL, which aims to parallel V-SAIL by fully exploiting the power of the multicore CPU. In general, PV-SAIL is based on the central memory to reduce the cost of communications. The subthreads are invoked in multiple rounds. In each round, the subthreads simultaneously search inside the neighborhoods in different Hamming distances, and the best result will be adopted as the new solution for the search in the next round. Fig. 5 shows the pseudocodes of the PV-SAIL algorithm. Some notable details are as follows.

Lines 1–13 show the process of the main thread. Note that to avoid unpredictable errors, we let the main thread wait for the termination of all subthreads in Line 7. Lines 14–23 describe the process of each subthread, which is similar to the **r**-step in V-SAIL. Note that to expand the search space for a better result, we let the subthreads search within the neighborhoods in different Hamming distances simultaneously, as indicated by $t = j$ in Line 14. Also, we do not set a hard stopping criterion in the subthreads, since the parameter k_{max} in Line 15 is often set to a small value, say not beyond 5 in our experiments. Finally, the best way to set $nSub$ is to keep consistency with the number of CPU cores. For instance, for most computers having a four-core CPU and if $k_{max} = 5$, PV-SAIL can search within 20 neighborhoods at short notice. This greatly increases the probability of finding a better clustering result.

$[objVal^*, label^*, \pi]$ =**PV-SAIL**$(para, nSub, maxInv)$

Input: $para \doteq \langle \mathbb{D}, K, reps, maxIter, k_{max} \rangle$ as in V-SAIL
$nSub$: the number of subthreads
$maxInv$: the max number of invocation of subthreads
Output: $objVal^*$: the optimal objective-function value
$label^*$: the cluster labels of documents
π: the vector of document weights

Procedure
Main thread:
1. $[objVal, label, \pi] =$ SAIL$(\mathbb{D}, K, reps, maxIter)$;
2. Create subthreads: $subThread[j]$, $j = 1, 2, \cdots, nSub$;
3. **for** $i = 1 : maxInv$
4. **for** $j = 1 : nSub$
5. $[objVal[j], label[j]] = SubThread[j]$.start();
6. **end for**
7. $SubThread[j]$.wait(), $j = 1, 2, \cdots, nSub$;
8. $j^* = \arg\min_j \{objVal[j], j = 1, \cdots, nSub\}$;
9. **if** $objVal[j^*] < objVal$
10. $objVal = objVal[j^*], label = label[j^*]$;
11. **end if**
12. **end for**
13. **return** $objVal^* = objVal$, $label^* = label$, π;
Subthread: // take $SubThread[j]$ for example
14. $t = j, objVal[j] = objVal, label[j] = label$;
15. **while** $(t++ < j + k_{max})$
16. $label' =$ Shaking$(\mathcal{N}_t, label)$;
17. Update $objVal'$ and $cluSum'$ accordingly;
18. LocSearch$(\mathbb{D}, \pi, objVal', label', cluSum')$;
19. **if** $objVal' < objVal[j]$
20. $objVal[j] = objVal', label[j] = label', t = j$;
21. **end if**
22. **end while**
23. **return** $objVal[j]$, $label[j]$;

Fig. 5. Pseudocodes of PV-SAIL.

In summary, PV-SAIL is a multithreaded version of V-SAIL, which aims to improve the clustering quality of V-SAIL given a limited time.

VI. Experimental Results

In this section, we demonstrate the effectiveness of SAIL and its variants for text clustering.

A. Experimental Setup

Data sets. For our experiments, we use a number of real-world text collections. Some characteristics of these data sets are shown in Table III, where "CV" is the coefficient of

TABLE III
EXPERIMENTAL TEXT DATA SETS

ID	Data	#Instance	#Feature	#Class	CV	Density
1	classic	7094	41681	4	0.547	0.0008
2	cranmed	2431	41681	2	0.212	0.0014
3	fbis	2463	2000	17	0.961	0.0799
4	k1a	2340	21839	20	1.004	0.0068
5	k1b	2340	21839	6	1.316	0.0068
6	la1	3204	21604	6	0.493	0.0048
7	la2	3075	31472	6	0.516	0.0048
8	la12	6279	31472	6	0.503	0.0047
9	new3	9558	83487	44	0.580	0.0029
10	ohscal	11162	11465	10	0.266	0.0053
11	re0	1504	2886	13	1.502	0.0179
12	reviews	4069	126373	5	0.640	0.0015
13	sports	8580	126373	7	1.022	0.0010
14	tr11	414	6429	9	0.882	0.0438
15	tr12	313	5804	8	0.638	0.0471
16	tr23	204	5832	6	0.935	0.0661
17	tr31	927	10128	7	0.936	0.0265
18	tr41	878	7454	10	0.913	0.0262
19	tr45	690	8261	10	0.669	0.0340
20	wap	1560	8460	20	1.040	0.0167

variation [26] used to characterize the class imbalance of the data sets, and "Density" is the ratio of non-zero feature values in each text collection. A large CV indicates a severe class imbalance, and a small density indicates a high sparseness.

Clustering tools. In the experiments, we employ four types of clustering tools. The first one is SAIL and its variants V-SAIL and PV-SAIL, coded by ourselves in C++. The other three are well-known software packages for K-means clustering, including MATLAB v7.1 [27], CO-CLUSTER v1.1 [28], and CLUTO v2.1.1 [19].

The MATLAB implementation of K-means is a batch-learning version, which computes the distances between instances and centroids. We extend it to include KL-divergence such that it can work as the traditional Info-Kmeans.

CO-CLUSTER is a C++ program which implements the information-theoretic co-clustering algorithm [9]. While still computing the KL-divergence between instances and centroids, it provides additional methods, e.g., annealing and local search, to improve the clustering performance.

CLUTO is a software package for clustering high-dimensional data sets. Specifically, its K-means implementation with cosine similarity as the proximity function shows superior performance in text clustering [21]. In the experiments, we compare CLUTO with SAIL on a number of real-world text data sets.

Note that the parameters of the four K-means implementations are set to match one another for the purpose of comparison, and the cluster number K is set to match the number of true clusters indicated by the class labels of each data set.

Validation measures. Many recent studies use the external validation measure: Normalized Mutual Information (NMI), to evaluate the clustering performance [10]. For consistency, we also use NMI in our experiments, which is computed as: $NMI = I(X,Y)/\sqrt{H(X)H(Y)}$, where the random variables X and Y denote the cluster and class sizes, respectively, $I(X,Y)$ is the mutual information between X and Y, and $H(X)$ and $H(Y)$ are the Shannon entropies of X and Y, respectively. The value of NMI is in the interval: [0, 1], and a larger value indicates a better clustering result. To further verify the validity of clustering, we also include the well-known statistical measure: normalized Rand index (R_n) [29]. R_n is the corrected-for-chance version of the Rand index based on the multivariate hypergeometric distribution (MHD), which is computed as: $R_n = (R - E(R))/(1 - E(R))$, where R is the Rand index value, and $E(R)$ is the expected Rand index value given MHD. A positive R_n implies a higher clustering quality than the expected baseline level. The larger the R_n value is, the better the clustering result. More computational details about NMI and R_n can be found in [30] with a contingency-table view.

TABLE IV
CLUSTERING RESULTS OF MATLAB INFO-KMEANS

Data	NMI	CV_0	CV_1
tr23	0.035	0.935	2.435
tr45	0.022	0.669	3.157

TABLE V
CLUSTERING RESULTS OF CO-CLUSTER

Data	Search	Annealing	NMI	CV_0	CV_1
tr12	batch	1.0	0.058	0.638	0.295
		0.5	0.040	0.638	0.374
		none	0.031	0.638	0.376
	local	1.0	0.045	0.638	0.334
		0.5	0.059	0.638	0.339
		none	0.048	0.638	0.461
tr31	batch	1.0	0.007	0.936	0.426
		0.5	0.011	0.936	0.362
		none	0.010	0.936	0.405
	local	1.0	0.014	0.936	0.448
		0.5	0.009	0.936	0.354
		none	0.011	0.936	0.365

B. Impact of Zero-Feature Dilemma

Here, we demonstrate the negative impact of the zero-feature dilemma to Info-Kmeans. Since the MATLAB implementation of Info-Kmeans can handle infinity (denoted as INF in MATLAB), we select `tr23` and `tr45` data sets and apply MATLAB Info-Kmeans on them. The clustering results are shown in Table IV, where CV_0 and CV_1 indicate the distributions of the cluster sizes before and after clustering, respectively.

As indicated by the close-to-zero NMI values, the clustering performance of MATLAB Info-Kmeans is extremely poor. Also, by comparing the CV_0 and CV_1 values, we found that the distributions of the resulting cluster sizes are much more skewed than the distributions of the class sizes. In fact, for both data sets, nearly all the documents have been assigned to one cluster! This clearly verifies our analysis in Section III; that is, Info-Kmeans will face the serious zero-feature dilemma when clustering highly sparse text data.

Furthermore, we test CO-CLUSTER on `tr12` and `tr31` data sets. Table V shows the clustering results, where "Search" and "Annealing" indicate the search modes and annealing parameters, respectively. As can be seen, the use of the annealing technique and different search modes does not improve the clustering performance. The near-to-zero NMI values indicate the poor clustering performance. This again verifies that to

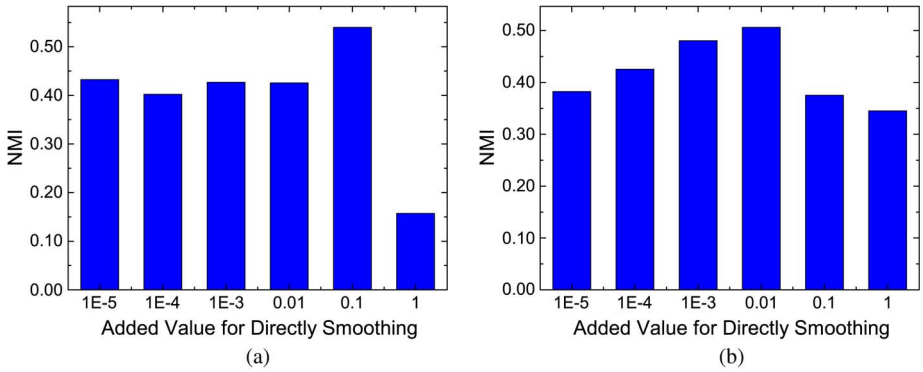

Fig. 6. Effect of data smoothing. (a) Data set: `tr11`. (b) Data set: `tr45`.

directly compute KL-divergence is infeasible for sparse data. Another interesting observation is that, the clusters produced by CO-CLUSTER are much more balanced than the clusters produced by MATLAB Info-Kmeans, as indicated by the much smaller CV_1 values.

C. Comparison of SAIL and the Smoothing Technique

Here, we illustrate the effect of the smoothing technique by using MATLAB Info-Kmeans. Fig. 6(a) and (b) shows the clustering results on data sets `tr11` and `tr45` with increasing added values.

One observation is that data smoothing indeed improves the clustering quality of Info-Kmeans, from nearly zero to about 0.3 (measured by NMI). This implies that the smoothing technique does help Info-Kmeans get out of the zero-feature dilemma, although the performance is still far from satisfactory. Another interesting observation is that the optimal added value (OAV) is varied for different data sets. For instance, while $OAV \approx 0.1$ for `tr11`, $OAV \approx 0.01$ for `tr45`. This indicates that to have the optimal smoothing effect is very difficult. Nevertheless, a general rule is that we should avoid setting extreme values. A tiny value may not take effect on the zero-feature dilemma, but a large value may change the nature of data and thus lead to poor clustering results. Fig. 6(b) well illustrates this point, where an added value smaller or larger than 0.01 will degrade the clustering quality.

We then compare the performance between the smoothing technique and SAIL on all the data sets. The parameters of SAIL are set as follows: 1) $\pi_x = 1$, for any $x \in \mathbb{D}$; 2) no row or column modeling; 3) the initialization mode is "random center"; 4) $reps = 1$, but we repeat SAIL five times for each data set and return the average score. **Unless otherwise stated, these are the default settings of SAIL in our experiments**.

Tables VI and VII show the results in terms of NMI and R_n, respectively. As can be seen, the clustering results of SAIL are consistently superior to the results of the smoothing technique. For some data sets such as `tr11`, `tr12`, `tr41`, and `tr45`, SAIL takes the lead with a wide margin. Note that for the smoothing method, we try a series of added values, i.e., 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, and select the best one for comparison.

In summary, while the data smoothing technique can improve the performance of Info-Kmeans on sparse data, it changes the data integrity and has difficulty in setting the OAV. In contrast, SAIL with default settings can lead to consistently better clustering performance, and therefore is a better solution for the zero-feature dilemma.

TABLE VI
PERFORMANCE COMPARISON IN TERMS OF NMI

ID	Data	Smoothing	CLUTO		SAIL	V-SAIL
			no IDF	IDF		
1	classic	N/A	0.450	0.561	0.691	0.702
2	cranmed	N/A	0.900	0.976	0.990	0.990
3	fbis	0.544	0.579	0.603	0.606	0.624
4	k1a	N/A	0.574	0.549	0.594	0.606
5	k1b	N/A	0.589	0.583	0.648	0.634
6	la1	N/A	0.360	0.571	0.595	0.599
7	la12	N/A	0.396	0.000	0.594	0.600
8	la2	N/A	0.355	0.443	0.530	0.600
9	new3	N/A	0.617	0.618	0.644	0.653
10	ohscal	N/A	0.347	**0.606**	0.430	0.436
11	re0	0.360	0.411	**0.666**	0.434	0.431
12	reviews	N/A	**0.651**	**0.646**	0.581	0.587
13	sports	N/A	0.442	0.583	0.685	0.644
14	tr11	0.342	0.628	0.111	0.640	0.656
15	tr12	0.312	0.634	0.401	0.645	0.675
16	tr23	0.193	0.271	**0.493**	0.385	0.404
17	tr31	0.485	0.411	0.543	0.545	0.591
18	tr41	0.445	0.551	0.244	0.645	0.693
19	tr45	0.464	0.554	0.364	0.618	0.708
20	wap	0.395	0.567	**0.679**	0.584	0.597

Note: (1) The value larger than the ones in both "SAIL" and "V-SAIL" is in bold.
(2) The value larger than "SAIL" but smaller than "V-SAIL" is in italic.
(3) "N/A" is due to out-of-memory.

D. Comparison of SAIL and Spherical K-means

In the literature, people have shown that spherical K-means usually have better clustering quality than traditional K-means [10]. The CLUTO version of spherical K-means [19] even shows superior performance on text collections, which makes it the benchmark for text clustering. However, we would like to show in this experiment that the performance of SAIL is comparable to the spherical K-means in CLUTO.

In this experiment, the parameter settings in CLUTO are as follows: clmethod = direct, crfun = i2, sim = cosine, colmodel = none, ntrials = 10. For SAIL, we use the default settings. Tables VI and VII show the results in terms of NMI and R_n, respectively, where CLUTO without IDF (also denoted as CLUTOwtIDF) indicates the clustering results of spherical K-means. As can be seen, SAIL shows consistently higher

TABLE VII
PERFORMANCE COMPARISON IN TERMS OF R_n

ID	Data	Smoothing	CLUTO		SAIL	V-SAIL
			no IDF	IDF		
1	classic	N/A	0.300	0.439	0.538	0.548
2	cranmed	N/A	0.950	0.990	0.997	0.997
3	fbis	0.315	0.381	0.383	0.397	0.446
4	k1a	N/A	0.322	0.390	0.412	0.418
5	k1b	N/A	**0.548**	**0.545**	0.441	0.415
6	la1	N/A	0.234	0.497	0.553	0.556
7	la12	N/A	0.234	0.518	0.543	0.554
8	la2	N/A	0.240	0.553	0.486	0.554
9	new3	N/A	0.382	0.374	0.410	0.421
10	ohscal	N/A	0.224	**0.357**	0.309	0.321
11	re0	0.199	0.204	0.194	0.236	0.201
12	reviews	N/A	**0.690**	*0.570*	0.477	0.577
13	sports	N/A	0.496	0.532	0.608	0.521
14	tr11	0.246	*0.479*	**0.565**	0.478	0.503
15	tr12	0.260	0.438	**0.545**	0.483	0.504
16	tr23	0.090	0.169	0.230	0.255	0.266
17	tr31	0.420	0.419	**0.532**	0.426	0.525
18	tr41	0.285	0.422	*0.566*	0.500	0.567
19	tr45	0.339	0.432	**0.710**	0.517	0.646
20	wap	0.178	0.278	**0.410**	0.396	0.393

Note: (1) The value larger than the ones in both "SAIL" and "V-SAIL" is in bold.
(2) The value larger than "SAIL" but smaller than "V-SAIL" is in italic.
(3) "N/A" is due to out-of-memory.

clustering quality on nearly all data sets. For some data sets, for example `classic`, `la1`, `la12`, `la2`, `sports`, and `tr31`, SAIL even shows dominant advantages. This demonstrates that SAIL is particularly suitable for text clustering, even compared with the state-of-the-art methods.

As it is reported that feature weighting often takes effect on the spherical K-means [21], we also compare the clustering performance between SAIL and CLUTO with IDF (also denoted as CLUTOwIDF) [31], where the parameter: colmodel = idf. Table VI shows the comparison result measured by NMI. Two observations are notable as follows. First, although CLUTOwIDF improves the clustering quality of CLUTOwtIDF in 12 out of 20 data sets, it still shows poorer performance than SAIL in 15 out of 20 data sets. Second, for some data sets, such as `la12`, `tr11`, `tr12`, `tr41`, and `tr45`, the IDF scheme actually seriously degrades the clustering performance of CLUTO. These observations imply that feature weighting is indeed an X-factor for the spherical K-means. In contrast, SAIL with default settings shows consistent clustering performance and therefore is more robust in practice. The situation is similar in Table VII where R_n is employed, although the gap between SAIL and CLUTOwIDF is narrowed—they show comparable clustering performance.

In summary, compared with the benchmark spherical K-means algorithm, SAIL shows merits in providing competitive and robust clustering results on a number of real-world text collections.

E. Inside SAIL

In this subsection, we take a further step to explore the properties of SAIL.

The efficiency. As mentioned in Section IV-B, we employ (11) instead of (9) for the computation of SAIL. Fig. 7 shows the runtime comparison, where "SAIL_old" represents SAIL using (9). As can be been, by using (11), SAIL improves the clustering efficiency greatly. For instance, for the large-scale data sets `ohscal`, `reviews`, and `sports`, SAIL is faster than SAIL_old by roughly two orders of magnitude. Note that the vertical axis of Fig. 7 represents the average time consumed by running one iteration of SAIL.

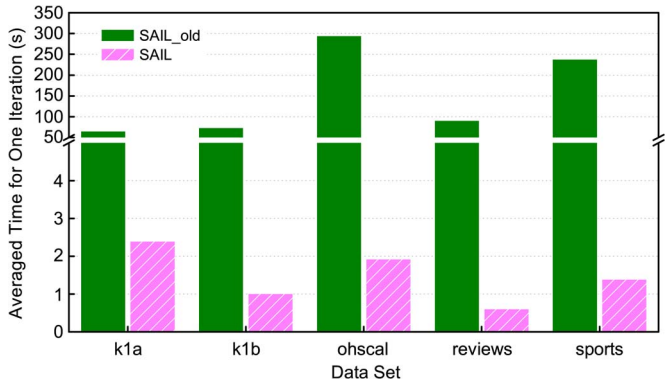

Fig. 7. Runtime comparison between SAIL and SAIL_old.

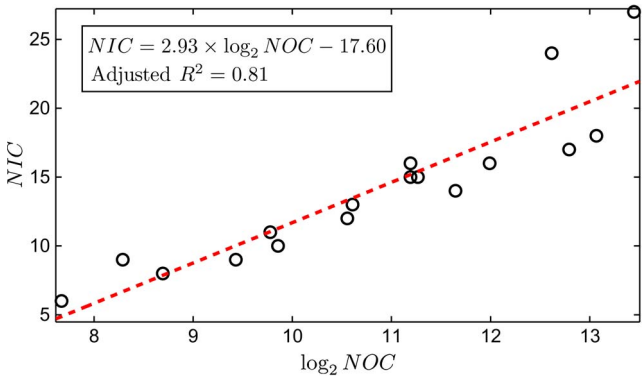

Fig. 8. Convergence of SAIL.

Fig. 8 then shows the convergence of SAIL, which characterizes the relationship between the number of instances (NOI) and the NIC. In the figure, each circle represents the situation of a data set. As can be seen, NIC is typically smaller than 20, except for data sets `la12`, `la2`, and `ohscal`, whose NIC values are larger than 20 but smaller than 30. Also, there is a significant linear correlation between NIC and $\log_2$ NOI, which indicates that NOI can only lead to a logarithmic increase in NIC. Note that NIC in Fig. 8 is the average value of five runs of SAIL, and we have deleted the outliers `cranmed`(NIC = 3), `new3`(NIC = 30), and `la12`(NIC = 29) for a better fit in Fig. 8.

The stability. As a variant of K-means, the clustering result of SAIL also depends on the generation of initial seeds, which brings the stability concern. To illustrate this, the clustering performances of SAIL in five runs with different initial seeds are summarized by box plots in Fig. 9. As can be seen, SAIL shows great stability to different initial seeds on most of the data sets.

The scalability. The algorithmic scalability is essential for large-scale text data analysis. To this end, we take two very large text collections without class labels, i.e. the Enron

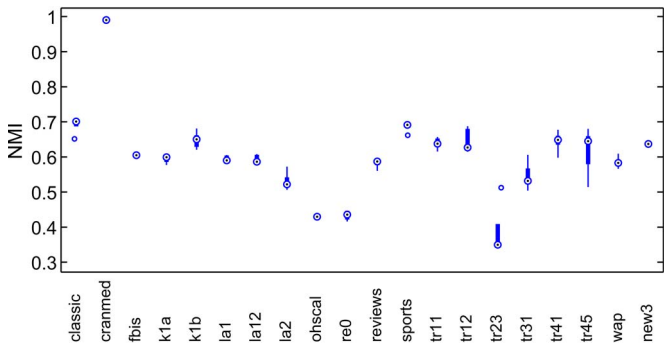

Fig. 9. Stability of SAIL.

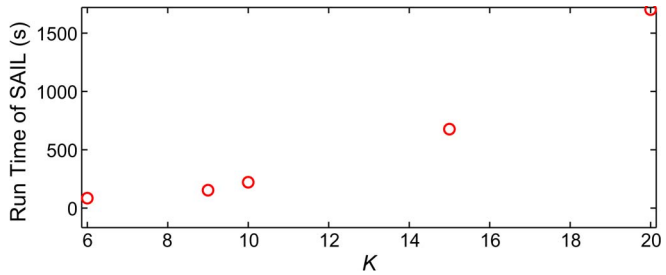

Fig. 10. Runtime trend of SAIL on Enron data.

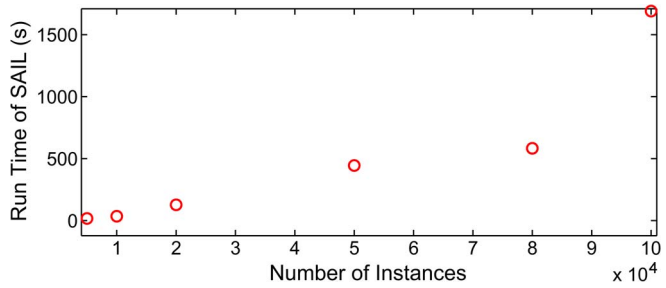

Fig. 11. Runtime trend of SAIL on NYTimes data.

email data (Enron)[1] and the NYTimes News Articles data (NYTimes)[2], for the scalability analysis of SAIL. Note that Enron contains 39 861 documents with 28 102 features, and the density is 0.0033. NYTimes contains 300 000 documents with 102 660 features, and the density is 0.0045. Fig. 10 shows the clustering results of SAIL on Enron along different numbers of clusters (K). Fig. 11 then shows the clustering results of SAIL on NYTimes with sampling sizes ranging from 5000 to 100 000. As can be seen from these two figures, SAIL shows good scalability to the number of clusters and the NOI.

Impact factors. We investigate other factors that may impact the performance of SAIL. Specifically, we introduce the feature weighting, text weighting, and bisecting schemes for SAIL, and observe the change of clustering performance. Table VIII shows the results. Note that "Default" represents SAIL with defaulting settings, "Feat. Wgt." represents SAIL using the IDF scheme, and "Text Wgt." represents SAIL for documents weighted by $p(x)$ in Theorem 1. As to "Bisecting," it represents a top-down divisive variant of SAIL. That is, it first divides all instances into two clusters using SAIL; and then repeatedly selects one cluster according to a certain criterion, and divides that cluster into two sub-clusters using SAIL again; the procedure will continue unless the desired K clusters are found. In our experiment, we use three cluster selection criteria for bisecting-SAIL, where "Size" chooses the largest cluster, "Obj" chooses the one with the largest objective-function value, and "Obj/Size" chooses the one with the largest average objective-function value.

As can be seen in Table VIII, one observation is that text weighting, bisecting with obj, and bisecting with obj/size often produce very poor clustering results (scores underlined), and therefore cannot be used for SAIL enhancement. In contrast, feature weighting and bisecting with size produce relatively robust results, and in some cases even improve SAIL slightly (scores in bold), and thus can be considered as valuable supplements to SAIL. Moreover, the bisecting scheme has another appealing merit; that is, the bisecting-SAIL is often more efficient than SAIL, as shown by Fig. 12. This is due to the decomposition effect of the bisecting scheme. Recall the time complexity of SAIL: $O(IKn\bar{d})$. In each run of bisecting-SAIL, K is reduced to 2, and more importantly, n is greatly reduced to the size of the selected cluster. In other words, there is no need to compare every point to every centroid, since to bisect a cluster, we just consider the points in that cluster and their distances to two centroids. Nonetheless, SAIL with default settings can produce better clustering results in most cases, and therefore should be the primary choice.

F. Performance of V-SAIL and PV-SAIL

Here, we illustrate the improvements of SAIL due to the introduction of the VNS and multithreaded schemes.

For the experiments of V-SAIL, we set $k_{max} = 12$, and return the average NMI values of five repetitions. The stopping criterion for V-SAIL is the maximum times for calling "LocSearch," which we set to 500. Tables VI and VII show the comparison results of V-SAIL and SAIL. As can be seen from the tables, V-SAIL generally achieves higher scores than SAIL. For some data sets, such as fbis, la2, tr31, tr41, and tr45, the improvements are quite satisfactory. Nonetheless, it is worth noting that for many data sets, V-SAIL only achieves slightly better clustering results than SAIL. This in turn implies that SAIL alone is capable of finding good enough solutions in many cases.

Next, we demonstrate the benefit of using a multithreaded scheme for V-SAIL. To this end, we compare PV-SAIL with V-SAIL by observing their objective-function values at each time point. Three large-scale data sets, i.e., ohscal, sports, and reviews, are selected for the comparison study. For V-SAIL, we set $k_{max} = 12$. Since the PC used for experiments has a four-core CPU, we run PV-SAIL with two, three, and four subthreads, respectively. In each subthread, we let $k_{max} = 3$. The stopping criterions for V-SAIL and PV-SAIL are removed for the comparison purpose. Fig. 13 shows the results. As can be seen, PV-SAIL typically obtains a lower objective-function value at each time point after the first call for SAIL (highlighted by a vertical line named "SAIL finished").

[1]http://www.cs.cmu.edu/~enron/
[2]http://ldc.upenn.edu/

TABLE VIII
IMPACT OF PARAMETER SETTINGS ON SAIL

Data	Default	Feat. Wgt. (IDF)	Text Wgt. (Prob.)	Bisecting Size	Bisecting Obj	Bisecting Obj/Size
classic	0.691	0.668	0.632	0.668	**0.729**	0.598
cranmed	0.990	**0.995**	0.990	0.990	0.990	0.990
fbis	0.606	**0.608**	0.554	0.569	0.569	0.579
k1a	0.594	0.551	**0.599**	0.548	0.567	0.532
k1b	0.648	0.568	0.607	0.538	0.554	0.416
la1	0.595	0.571	0.559	0.578	0.574	0.471
la12	0.594	0.578	0.329	0.559	0.554	0.484
la2	0.530	0.501	0.498	**0.549**	0.535	0.492
new3	0.644	0.612	0.057	0.588	0.592	0.356
ohscal	0.430	0.402	0.291	0.408	0.407	0.384
re0	0.434	0.360	0.408	0.401	0.407	0.252
reviews	0.581	0.508	0.338	0.545	0.570	0.542
sports	0.685	0.650	0.533	0.641	0.002	0.001
tr11	0.640	0.584	0.578	**0.642**	**0.658**	0.498
tr12	0.645	0.556	0.447	0.553	0.545	0.415
tr23	0.385	**0.403**	0.147	0.364	0.327	0.229
tr31	0.545	0.525	**0.550**	**0.616**	**0.550**	**0.569**
tr41	0.645	0.623	**0.684**	0.585	0.589	0.592
tr45	0.618	**0.685**	0.527	**0.630**	0.602	0.543
wap	0.584	0.543	**0.594**	0.558	0.548	0.549

Note: (1) Measured by *NMI*.
(2) The value larger than the one in "Default" is in bold.
(3) The value far smaller than the one in "Default" is underlined.

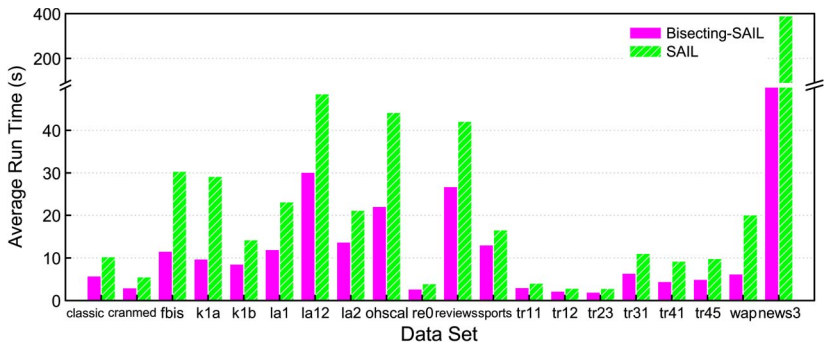

Fig. 12. Runtime comparison between SAIL and bisecting-SAIL.

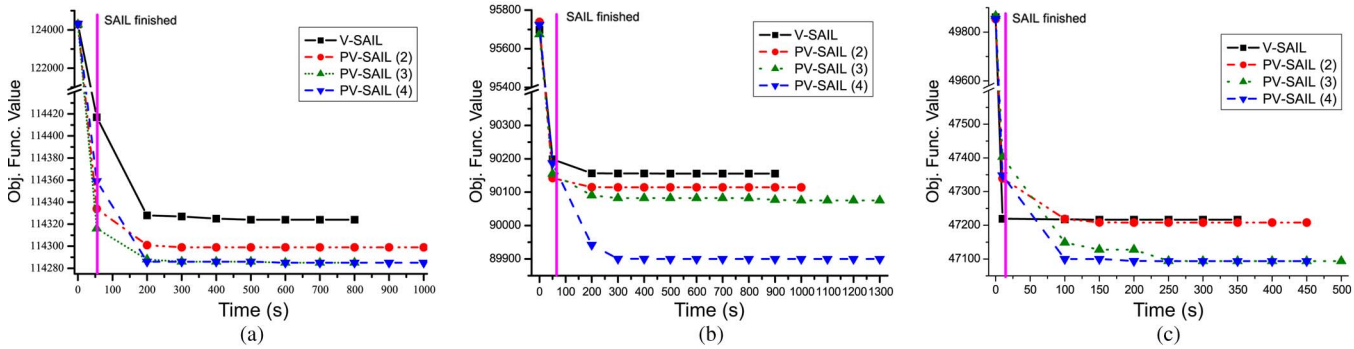

Fig. 13. Runtime comparison between PV-SAIL and V-SAIL. (a) Data set: `ohscal`. (b) Data set: `sports`. (c) Data set: `reviews`.

In summary, V-SAIL indeed improves the clustering performance of SAIL by using the VNS scheme. The high runtime cost of V-SAIL is then lowered by further introducing the multithreaded scheme, which results in the PV-SAIL algorithm.

VII. RELATED WORK

In the literature, great research efforts have been taken to incorporate information-theoretic measures into existing clustering algorithms, such as K-means [7]–[11], [14], [32]. However, the zero-feature dilemma remains a critical challenge.

For instance, Dhillon *et al.* proposed information-theoretic K-means, which used the KL-divergence as the proximity function [8]. While the authors noticed the "infinity" values when computing the KL-divergence, they did not provide specific solutions to this dilemma. In addition, Dhillon *et al.* further extended information-theoretic K-means to the so-called information-theoretic co-clustering [9]. This algorithm is the 2-D version of information-theoretic K-means which monotonically increases the preserved mutual information by interwinding both the row and column clusterings at all stages. Again, however, there is no solution provided for handling

the zero-feature dilemma when computing the KL-divergence. Meila and Heckerman [33] compared hard and soft assignment strategies for text clustering using multinomial models from which Info-Kmeans can be derived. However, they also omitted the details of how to handle the zero-feature in computations.

Since many other proximity functions such as the squared Euclidean distance and the cosine similarity can also be used in K-means clustering [2], a natural idea is to compare their performances in practice. [10] is one of such studies. The authors argued that the spherical K-means produces better clustering results than Info-Kmeans. [21], [34], [35] also showed the merits of spherical K-means in text clustering. In many studies, the spherical K-means in CLUTO [19] has become the benchmark for text clustering. However, these studies did not tell us why Info-Kmeans shows inferior performance to spherical K-means and how to enhance Info-Kmeans.

Our paper indeed fills this crucial void by proposing the SAIL algorithm and its variants to handle the zero-feature dilemma and improving the clustering performance of Info-Kmeans to a level competitive to the spherical K-means.

VIII. Concluding Remarks

This paper studied the problem of exploiting KL-divergence for information-theoretic K-means clustering (denoted as Info-Kmeans for short). In particular, we revealed the dilemma of Info-Kmeans for handling high-dimensional sparse text data; that is, the centroids in sparse data usually contain zero-value features and thus lead to infinite KL-divergence values. This makes it difficult to use KL-divergence as a criterion for assigning objects to the centroids. To deal with it, we developed a SAIL algorithm, which can avoid the zero-feature dilemma by computing the Shannon entropy instead of the KL-divergence. The effectiveness of this replacement is guaranteed by an equivalent mathematical transformation in the objective function of Info-Kmeans. Moreover, we proposed two variants, i.e., V-SAIL and PV-SAIL, to further enhance the clustering ability of SAIL. Finally, as demonstrated by extensive text corpora in our experiments, SAIL can greatly improve the performance of Info-Kmeans on high-dimensional sparse text data. V-SAIL and PV-SAIL further improve the clustering performance of SAIL in terms of both quality and efficiency.

References

[1] J. Wu, H. Xiong, and J. Chen, "SAIL: Summation-based incremental learning for information-theoretic clustering," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 740–748.

[2] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Reading, MA: Addison-Wesley, 2005.

[3] E.-H. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore, "Webace: A web agent for document categorization and exploration," in *Proc. 2nd Int. Conf. Auton. Agents*, 1998, pp. 408–415.

[4] H. Zhou, D. Tao, Y. Yuan, and X. Li, "Object trajectory clustering via tensor analysis," in *Proc. 16th IEEE Int. Conf. Image Process.*, 2009, pp. 1945–1948.

[5] X. Gao, B. Wang, D. Tao, and X. Li, "A relay level set method for automatic image segmentation," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 2, pp. 518–525, Apr. 2011.

[6] X. Tian, D. Tao, and Y. Rui, "Complex object correspondence construction in two-dimensional animation," *IEEE Trans. Image Process.*, vol. 20, no. 11, pp. 3257–3269, Nov. 2011.

[7] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Non-negative patch alignment framework," *IEEE Trans. Neural Netw.*, vol. 22, no. 8, pp. 1218–1230, Aug. 2011.

[8] I. Dhillon, S. Mallela, and R. Kumar, "A divisive information-theoretic feature clustering algorithm for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1265–1287, Mar. 2003.

[9] I. Dhillon, S. Mallela, and D. Modha, "Information-theoretic co-clustering," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2003, pp. 89–98.

[10] S. Zhong and J. Ghosh, "Generative model-based document clustering: A comparative study," *Knowl. Inf. Syst.*, vol. 8, no. 3, pp. 374–384, Sep. 2005.

[11] C. Elkan, "Clustering documents with an exponential-family approximation of the dirichlet compound multinomial distribution," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 289–296.

[12] T. Cover and J. Thomas, *Elements of Information Theory,* 2nd ed. New York: Wiley, 2006.

[13] S. Kullback and R. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, Mar. 1951.

[14] N. Slonim and N. Tishby, "The power of word clusters for text classification," in *Proc. 23rd Eur. Colloq. Inf. Retrieval Res.*, 2001.

[15] T. Zhou, D. Tao, and X. Wu, "Manifold elastic net: A unified framework for sparse dimension reduction," *Data Mining Knowl. Discov.*, vol. 22, no. 3, pp. 340–371, 2010.

[16] X. Tian, D. Tao, and Y. Rui, "Sparse transfer learning for interactive video search reranking," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 8, no. 3, p. 26, Jul. 2012.

[17] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, Nov. 1997.

[18] P. Hansen and N. Mladenovic, "Variable neighborhood search: Principles and applications," *Eur. J. Oper. Res.*, vol. 130, no. 3, pp. 449–467, May 2001.

[19] G. Karypis, Cluto-Version 2.1.1. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/views/cluto

[20] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Stat. Probab.*, 1967, pp. 281–297.

[21] Y. Zhao and G. Karypis, "Criterion functions for document clustering: Experiments and analysis," *Mach. Learn.*, vol. 55, no. 3, pp. 311–331, 2004.

[22] L. Brand, *Advanced Calculus: An Introduction to Classical Analysis*. New York: Dover, 2006.

[23] J. Wu, H. Xiong, J. Chen, and W. Zhou, "A generalization of proximity functions for k-means," in *Proc. 7th IEEE Int. Conf. Data Mining*, 2007, pp. 361–370.

[24] J. Wu, H. Xiong, C. Liu, and J. Chen, "A generalization of distance functions for fuzzy c-means clustering with cetroids of arithmetic means," *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 3, pp. 557–571, Jun. 2012.

[25] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh, "Clustering with Bregman divergences," *J. Mach. Learn. Res.*, vol. 6, no. 10, pp. 1705–1749, Oct. 2005.

[26] W. Hendricks and K. Robey, "The sampling distribution of the coefficient of variation," *Ann. Math. Stat.*, vol. 7, no. 3, pp. 129–132, Sep. 1936.

[27] *K-means Clustering in Statistics Toolbox*, MathWorks, Natick, MA. [Online]. Available: http://www.mathworks.com

[28] I. Dhillon, *Co-Clustering Software, Version 1.1*, Department of Computer Science, University of Texas Austin, Austin, TX. [Online]. Available: http://www.cs.utexas.edu/~inderjit/software.shtml

[29] A. Jain and R. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[30] J. Wu, H. Xiong, and J. Chen, "Adapting the right measures for k-means clustering," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2009, pp. 877–886.

[31] H. Wu, R. Luk, K. Wong, and K. Kwok, "Interpreting TF-IDF term weights as making relevance decisions," *ACM Trans. Inf. Syst.*, vol. 26, no. 3, pp. 1–37, Jun. 2008.

[32] J. Wu, *Advances in K-means Clustering: A Data Mining Thinking*. New York: Springer-Verlag, 2012.

[33] M. Meila and D. Heckerman, "An experimental comparison of model-based clustering methods," *Mach. Learn.*, vol. 42, no. 1/2, pp. 9–29, Jan./Feb. 2001.

[34] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2000.

[35] A. Banerjee, I. Dhillon, J. Ghosh, and S. Sra, "Clustering on the unit hypersphere using von mises-fisher distributions," *J. Mach. Learn. Res.*, vol. 6, pp. 1345–1382, Dec. 2005.

Jie Cao received the Ph.D. degree from Southeast University, Nanjing, China, in 2002.

Currently, he is a Professor and the Director of Jiangsu Provincial Key Laboratory of E-Business at Nanjing University of Finance and Economics, Nanjing, China. He has been selected in the Program for New Century Excellent Talents in University and awarded with the Young and Mid-aged Expert with Outstanding Contribution in Jiangsu province. His main research interests include cloud computing, business intelligence, and data mining. He has published one book and more than 40 referred papers in various conferences and journals. He has taken charge of over 20 national projects and obtained the second prize in the Jiangsu Provincial Scientific and Technological Progress Award in 2008.

Zhiang Wu received the Ph.D. degree from Southeast University, Nanjing, China, in 2009.

Currently, he is an Associate Professor of Jiangsu Provincial Key Laboratory of E-Business at Nanjing University of Finance and Economics, Nanjing, China. He has authored more than 20 referred papers in international journals and conferences. His recent research interests include network computing, recommender systems, and data mining.

Dr. Wu is a member of the China Computer Federation and Association for Computing Machinery. He is a recipient of the Excellent Student Paper Award of the IFIP International Conference on Network Parallel Computing in 2007.

Junjie Wu (M'10) received the B.E. degree in civil engineering and Ph.D. degree in management science and engineering, in 2002 and 2008, respectively, both from Tsinghua University, Beijing, China.

Currently, he is an Associate Professor in the Information Systems Department, School of Economics and Management, Beihang University, Beijing, China, the Vice Director of Beijing Key Laboratory of Emergency Support Simulation Technologies for City Operations, and an Outside Researcher in the Research Center for Contemporary Management, Key Research Institute of Humanities and Social Sciences at Universities, Tsinghua University. His general area of research is data mining and complex networks. Currently, he is the Principal Investigator of three National Natural Science Foundation of China projects and one MOE project, and has published over 40 papers in refereed conference proceedings and journals, such as KDD, DMKD, and TKDE.

Dr. Wu is the recipient of the 2011 Program for New Century Excellent Talents in University and the 2010 National Excellent Doctoral Dissertation Award, both in China. He is a member of AIS and ACM.

Hui Xiong (SM'07) received the B.E. degree from the University of Science and Technology of China, Hefei, China, in 1995, the M.S. degree from the National University of Singapore, Singapore, in 2000, and the Ph.D. degree from the University of Minnesota, Minneapolis, in 2005.

Currently, he is an Associate Professor and the Vice Chair of the Management Science and Information Systems Department at Rutgers University. His general area of research is data and knowledge engineering, with a focus on developing effective and efficient data analysis techniques for emerging data intensive applications. He has published over 100 technical papers in peer-reviewed journals and conference proceedings. He is a co-editor of *Clustering and Information Retrieval* (Kluwer Academic Publishers, 2003) and a co-Editor-in-Chief of *Encyclopedia of GIS* (Springer, 2008). He is an Associate Editor of the *Knowledge and Information Systems* journal and has served regularly in the organization committees and the program committees of a number of international conferences and workshops.

Dr. Xiong is a Senior Member of the ACM.