Towards information-theoretic K-means clustering for image indexing

Jie Cao\textsuperscript{a}, Zhiang Wu\textsuperscript{a}, Junjie Wu\textsuperscript{b,}* Wenjie Liu\textsuperscript{c}

\textsuperscript{a} Jiangsu Provincial Key Laboratory of E-Business, Nanjing University of Finance and Economics, Nanjing, China
\textsuperscript{b} Department of Information Systems, School of Economics and Management, Beihang University, Beijing, China
\textsuperscript{c} Department of Computer Science \& Technology, Nanjing University, Nanjing, China

1. Introduction

Image indexing aims to identify images based on their attributes and provide access to useful groupings of images. Without image indexing, most of the images will remain buried in the databases, never seen by the users [25]. In general, there are two types of approaches for image indexing: (1) The concept-based approaches, in which image attributes and semantic content are identified and described verbally by human indexers. (2) The content-based approaches, in which features of images are automatically identified and extracted by computer software. Cluster analysis, which provides insight into the data by dividing the objects into groups (clusters) of objects, shows great potential in grouping and summarizing images for content-based image indexing, and therefore absorbs much research attention [19]. Indeed, clustering similar images is equivalent to looking for those graph representations that are similar to each other in a database. Fig. 1 shows such an example, where cluster analysis helps to retrieve three types of Oxford landmarks. Nevertheless, clustering a large set of images is still widely unexplored and remains one of the most challenging problems in structural pattern recognition [19,12,24,6].

Recent years have witnessed an increasing interest in information-theoretic clustering [3,4,28,10], as information theory [2] can be naturally adopted as the guidance for the clustering process. This type of algorithms usually treats clustering as the iterative process of finding the best partitioning on data in a way such that the loss of
mutual information due to the partitioning is the least [3].

Our study in this paper is focused on employing Info-Kmeans, a K-means clustering algorithm [5] with KL-divergence [13] as the distance function, for the clustering of high-dimensional sparse data extracted from images, texts, genes, etc. For instance, images are often preprocessed by the popular bag-of-features (BOF) model [26], which treats images as collections of unordered appearance descriptors extracted from local patches, and quantizes these descriptors into discrete “visual words”. By doing this, an image can be characterized by a vector of visual words, which is usually in high dimensionality and extremely sparse [18,7,8].

While being rooted in information theory, Info-Kmeans still faces some challenging issues from a practical viewpoint. For example, research on text clustering has shown that, the performance of Info-Kmeans is poorer than that of the spherical K-means, which takes the cosine similarity as the proximity function and becomes the de facto benchmark of text clustering [27]. We argue that, however, this largely attributes to the zero-feature dilemma of Info-Kmeans. That is, for high-dimensional sparse data, the centroids of Info-Kmeans usually contain many zero-value features. This creates infinite KL-divergence values and leads to a challenge in assigning objects to the centroids during the iterative process of Info-Kmeans. A traditional solution is to smooth the sparse data by adding a small value to every data object. But this smoothing technique can also degrade the clustering performance of Info-Kmeans, since the true values and data sparseness have been changed severely.

In light of this, 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, we identify the zero-feature dilemma of Info-Kmeans, and point out that this dilemma is the primary reason that degrades the performance of Info-Kmeans on high-dimensional data. Second, we propose an objective-equivalent algorithm SAIL for Info-Kmeans clustering, which can avoid the zero-feature dilemma by replacing the computation of KL-divergence between instances and centroids by the computation of the centroid entropies only. Third, the entropy computation in SAIL is further refined to make use of the data sparseness to improve the efficiency of SAIL. Finally, the Variable Neighborhood Search (VNS) meta-heuristic [9] is introduced to improve the clustering quality of SAIL, provided that the clustering efficiency is not a major concern.

Our experimental results on various benchmark 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 even slightly better than the results by the state-of-the-art spherical K-means algorithm: CLUTO. 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, with the help of VNS, V-SAIL further improves the clustering quality of SAIL by searching around the neighborhood of the solution. Finally, SAIL and V-SAIL are employed for the real-world high-dimensional, noisy image data clustering. Experimental results show that after noise removal using the CosMiner algorithm [22], SAIL and V-SAIL can successfully recognize nine out of 11 landmarks from the extremely sparse Oxford_5K data set.

2. Problem definition

K-means [14] is a prototype-based, simple partitional clustering algorithm, which attempts to find the user-specified $K$ clusters via a two-phase iterative process. K-means has an objective function:

$$\text{obj} : \min \sum_{k \in C} \sum_{x \in c_k} \pi_x \text{dist}(x, m_k),$$

where $c_k$ denotes cluster $k$, $m_k$ denotes the centroid of $c_k$ (a centroid is the arithmetic mean of the cluster members), $\text{dist}(\cdot)$ is the distance function, and $\pi_x > 0$ is the weight of $x$.

2.1. Information-theoretic K-means

As we know, different distance functions lead to different types of K-means. Let $D(x|y)$ denote the KL-divergence [13] between two discrete distributions
x and y, we have
\[ D(x|y) = \sum_{i} x_i \log \frac{x_i}{y_i}. \] (2)

It is easy to note that in most cases \( D(x|y) \neq D(y|x) \), and that \( D(x|y) + D(y|z) \geq D(x|z) \) cannot be guaranteed. So \( D \) is not a metric. If we let “dist” be \( D \) in Eq. (1), we have the objective function of information-theoretic K-means clustering (Info-Kmeans) as follows:
\[ O_1 : \min_{\{c_t\}} \sum_{k \neq c_t} \sum_{x} p_x D(x|m_k^t), \] (3)
where each instance \( x \) is normalized to a discrete distribution, and \( m_k = \sum_{x} p_x x/\sum_{x} p_x \) is the arithmetic mean of instances assigned to cluster \( c_k \). Let \( \lfloor \cdot \rfloor \) denote the sum of feature values of a vector. Since \( |x| = 1 \), we have \( |m_k| = 1, \forall k \).

It has been pointed out that Info-Kmeans actually aims to minimize the loss of mutual information between the instance variable and the feature variable after the clustering [3]. This illustrates why Info-Kmeans belongs to the family of information-theoretic clustering.

2.2. The problem of Info-Kmeans

Though having clear physical meaning, Info-Kmeans has long been criticized for performing relatively poorly on high-dimensional sparse data [28]. In this section, however, we highlight an implementation challenge of Info-Kmeans. We believe this challenge is one of the major factors that degrade the performance of Info-Kmeans.

Assume that we use Info-Kmeans to cluster a text corpus. To this end, we must compute the KL-divergence between each text vector \( x \) and each centroid \( m_k \). In practice, we usually let \( x = x/|x| \), and then compute \( D(x|m_k^t) \) by Eq. (2). Note that Eq. (2) implies that all the feature values of \( x \) are positive real numbers. Unfortunately, however, this is not the case for high-dimensional data, which are usually very sparse in their feature space.

To illustrate this, we observe the computation of KL-divergence in the \( t \)th dimension. Let \( D_i \) denote \( x_i \log (x_i/m_{k_i}^t) \), we have four scenarios as follows:

1. Case 1: \( x_i > 0 \) and \( m_{k_i} > 0 \). In this case, the computation of \( D_i \) is straightforward, and the result can be any real number.
2. Case 2: \( x_i = 0 \) and \( m_{k_i} = 0 \). In this case, we can simply omit this feature, or equivalently let \( D_i = 0 \).
3. Case 3: \( x_i = 0 \) and \( m_{k_i} > 0 \). In this case, \( \log (x_i/m_{k_i}) = \log 0 = -\infty \), which implies that the direct computation is infeasible. However, by the L’Hospital’s rule [1], \( \lim_{x \to 0^+} x \log (x/a) = 0 (a > 0) \). So we can let \( x_i = \epsilon_i \) and \( a = m_{k_i} \), and thus have \( D_i = 0 \).
4. Case 4: \( x_i > 0 \) and \( m_{k_i} = 0 \). In this case, \( D_i = +\infty \), which is hard to handle in practice.

We summarize the four cases in Table 1. In general, for Cases 1 and 2, the computation of \( D_i \) is logically reasonable. However, the computation of \( D_i \) in Case 3 is somewhat weird; that is, it cannot reveal any difference between \( x_i \) and \( m_{k_i} \), although \( m_{k_i} \) may deviate heavily from zero.

Nevertheless, the most difficult case is Case 4. It will lead to infinite \( D \) and hinder the instance from being properly assigned. This is particularly true for high-dimensional sparse data, since the centroids of such data typically contain many zero-value features. We call this problem the “zero-feature dilemma”.

**Problem definition:** Design a new information-theoretic K-means algorithm, which can avoid the zero-feature dilemma and is particularly suitable for high-dimensional sparse-data clustering.

3. The SAIL algorithm

In this section, we propose a new algorithm called Summation-bAsed Incremental Learning (SAIL), for Info-Kmeans clustering.

3.1. SAIL: theoretical foundation

Let \( H(x) \) denote the Shannon entropy of a discrete distribution \( x \). We first have the following lemma:

**Lemma 1.**
\[ D(x|y) = -H(x) + H(y) + (x-y)^T \nabla H(y). \] (4)

**Proof.** Since \( H(y) = -\sum_i y_i \log y_i \), it is easy to have
\[ \nabla H(y) = -(\log y_1, \ldots, \log y_d)^T - \log e \times (1, \ldots, 1)^T. \]

Accordingly, we have
\[ -H(x) + H(y) + (x-y)^T \nabla H(y) = \sum_{i=1}^d x_i \log (x_i/y_i) - \log e \times (x_i/y_i)^T. \] (a)

Since (a) = \( D(x|y) \) and (b) = 0 provided that \( \sum_{i=1}^d x_i = \sum_{i=1}^d y_i = 1 \), the lemma follows. \( \square \)

Based on \( D(x|y) \) in Eq. (4), we now derive SAIL, a new variant of Info-Kmeans. Specifically, we have the following theorem:

**Theorem 1.** Given \( \pi_{ct} = \sum_{x \in c_x} \pi_x \), the objective function of Info-Kmeans \( O_1 \) in Eq. (3) is equivalent to
\[ O_2 : \min_{\{c_t\}} \sum_k \pi_{ct} H(m_k^t). \] (5)

**Proof.** By Eq. (4), we have \( D(x|m_k^t) = -H(x) + H(m_k^t) + (x-m_k^t)^T \nabla H(m_k^t) \). As a result,
\[ \sum_{k \neq c_t} \sum_{x} p_x D(x|m_k^t) = \sum_k \pi_{ct} H(m_k^t) - \sum_x \pi_x H(x). \] (a)

\[ \sum_{k \neq c_t} \sum_{x} p_x D(x|m_k^t) = \sum_k \pi_{ct} H(m_k^t) + \sum_k \pi_x H(x). \] (b)
Since \( m_k = \sum_{x \in c_k} \pi_x(x-m_k)^2 \) \( \nabla H(m_k) \).

(c)

O2 in Eq. (5) gives the objective function of SAIL. That is, by replacing the computations of KL-divergence between instances and centroids by the computations of centroid entropies only, SAIL can avoid the zero-feature dilemma in information-theoretic clustering of highly sparse data.

### 3.2. SAIL: computational issues

Here we specify the computational details of SAIL. The major concern is the efficiency issue.

Generally speaking, SAIL is a greedy scheme which updates the objective-function value “instance by instance”. That is, SAIL firstly selects an instance from data 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 \( x \) from cluster \( c_k \). Then, if we assign \( x \) to cluster \( c_k \), the change of the objective-function value will be

\[
A_k = O_2(\text{new}) - O_2(\text{old}) = (\pi_{c_k} - \pi_x) \log \frac{\pi_{c_k}}{\pi_x} - \pi_{c_k} \log \frac{\sum_{x \in c_k} \pi_x}{\pi_{c_k}} + \pi_{c_k} \log \frac{\sum_{x \in c_k} \pi_x}{\pi_x}
\]

\[
= \underbrace{(\pi_{c_k} - \pi_x) \log \frac{\sum_{x \in c_k} \pi_x}{\pi_x}}_{(a)} - \underbrace{\pi_{c_k} \log \pi_{c_k}}_{(b)} - \underbrace{\pi_{c_k} \log \pi_x}_{(c)} + 
\]

\[
\underbrace{\pi_{c_k} \log \pi_{c_k}}_{(d)} - \underbrace{\pi_{c_k} \log \pi_x}_{(c)} + 
\]

\[
\underbrace{(\pi_{c_k} + \pi_x) \log \frac{\sum_{x \in c_k} \pi_x + \pi_x}{\pi_x}}_{(b)} - \underbrace{(\pi_{c_k} + \pi_x) \log \pi_{c_k}}_{(a)} - \underbrace{\pi_{c_k} \log \pi_x}_{(c)} + 
\]

\[
\underbrace{(\pi_{c_k} + \pi_x) \log \pi_x}_{(c)}.
\]

where (a), (b) and (c), (d) represent the two parts of changes of the objective-function value due to the movement of \( x \) from cluster \( c_k \) to cluster \( c_k \). Then \( x \) will be assigned to the cluster \( c \) with the smallest \( A_k \).

The computation of \( A_k \) in Eq. (6) has two appealing properties. First, it only depends on the changes occurred within the two involving clusters \( c_k \) and \( c_k \). Other clusters remain unchanged and thus have no contribution to \( A_k \). Second, both \( \sum_{x \in c_k} \pi_x \) and \( \sum_{x \in c_k} \pi_x x \) have additivity, which can be incrementally updated for the computation of \( A_k \).

The computation of \( A_k \) in Eq. (6), however, is still time-consuming when computing Shannon entropy in (a) or (c). Let us take (a) for example. Since the denominator changes from \( \pi_x \) to \( \pi_x \), every non-zero feature in the numerator \( \sum_{x \in c_k} \pi_x x \) will have a new value, which requires us to update the logarithm of all these features, and thus leads to a huge cost for data instances in high dimensionality. To deal with this, in what follows, we refine the computational scheme of SAIL.

Recall SAIL’s objective function \( O_2 \) in Eq. (5). Let \( S_k \) denote \( \sum_{x \in c_k} \pi_x x \), and \( S^{\text{old}}_k \) the ith dimension of \( S_k \). Since \( m_k = \sum_{x \in c_k} \pi_x x / \pi_{c_k} \), we have

\[
\sum_{x \in c_k} \pi_x \log (\pi_x / \pi_{c_k}) = \sum_{x \in c_k} \pi_x \log \pi_x - \sum_{x \in c_k} \pi_x \log \pi_{c_k}
\]

\[
= \sum_{x \in c_k} \pi_x \log \pi_x - \sum_{x \in c_k} \pi_x \log \pi_{c_k} + \sum_{x \in c_k} \pi_x \log \pi_{c_k}
\]

\[
= \sum_{x \in c_k} \log \pi_x - \sum_{x \in c_k} \log \pi_{c_k}.
\]

Accordingly, if we move \( x \) from cluster \( c_k \) to cluster \( c_k \), the change of the objective-function value will be

\[
A_k = (\pi_{c_k} + \pi_x) \log \left( \frac{\pi_{c_k} + \pi_x}{\pi_{c_k}} \right) - \pi_{c_k} \log \pi_{c_k} + \underbrace{\sum_{x \in c_k} \pi_x \log \pi_{c_k}}_{(a)} + 
\]

\[
\underbrace{\sum_{x \in c_k} \pi_x \log \pi_{c_k}}_{(c)} + 
\]

\[
\underbrace{\sum_{x \in c_k} \log \pi_x - \sum_{x \in c_k} \log \pi_{c_k}}_{(b)}.
\]

where \( S^{\text{old}}_k = S_k + \pi_x x \) and \( S_k = S_k - \pi_x x \).

According to Eq. (8), only the non-zero features of \( x \) have contributions to \( A_k \) in (a) and (b), and thus will trigger the expensive computations of logarithm. Considering that a high-dimensional vector \( x \) is often very sparse, the computational saving due to Eq. (8) will be significant. As a result, we adopt Eq. (8) rather than Eq. (6) for the computation of \( A_k \) in SAIL.

### 3.3. SAIL: algorithmic details

Here we present the implementation details of SAIL. Figs. 2 and 3 show the pseudocodes of SAIL.

Lines 1–3 in Fig. 2 are for data initialization. Two methods, i.e., \( \pi_x = |x| / \sum_{x \in D} |x| \), or simply \( \pi_x = 1 \), are provided to assign weights to instances. The second one is much simpler and is the default setting in our experiments. The preprocessing of \( D \) in line 2 includes the row and column modeling, e.g., tf-idf, to initialize the instances and assign weights to the features. Then in line 3, we normalize the instance \( x \) to \( x / |x| \).

Lines 4–12 in Fig. 2 show the clustering process. Line 5 is for initialization, where \( \text{label}_{in.x} \) contains the cluster labels of all instances, and \( \text{clusSum}_{k,(d+1)} \) stores the summations of the weights and weighted instances in each cluster. That is, for \( k=1, \ldots, K \), \( \text{clusSum}(k, d+1) = S_k \), and \( \text{clusSum}(k, d+1) = \pi_{c_k} \), where \( d \) and \( K \) are the numbers of features and clusters, respectively. Two initialization modes are employed in our implementation, i.e., “random label” and “random center” (the default one).

The LocalSearch subroutine in Line 7 performs clustering for each instance. As shown in Fig. 3, it traverses all instances at random as a round. In each round, it assigns each instance to the cluster with the highest density of the objective-function value, and then updates the values of the relevant variables. The computational details have been given in Section 3.2.
[objVal*, label*, π] = SAIL(D, K, reps, maxIter)

Input:
- D: Data set
- K: the number of clusters
- reps: the number of repeated clusterings
- maxIter: the max number of iterations

Output:
- objVal*: the optimal objective-function value
- label*: the assigned cluster labels of instances
- π: the vector of instance weights

Variable:
- cluSum: storing πxa and Sb, ∀ k

Procedure
1. Load D, and compute π = {πx|x ∈ D};
2. Preprocess D;
3. ∀ x ∈ D, normalize x to a discrete distribution;
4. for i = 1 : reps
5. Initialize label[i], then objVal[i] and cluSum[i];
6. for j = 1 : maxIter
7. LocalSearch(D, π, objVal[i], label[i], cluSum[i]);
8. if label[i] is unchanged
9. break;
10. end if
11. end for
12 end for
13.t = arg min, objVal[i];
14.return objVal* = objVal[t], label* = label[t], π;

Fig. 2. The pseudocodes of SAIL.

LocalSearch(D, π, objVal, label, cluSum)
1. for l = 1 : n
2. Randomly select without replacement a x from D;
3. for s = 1 : K
4. ∆objVal(k) = TestAssign(para);
5. if para = (x, πx, cluSum, label(x), k, objVal(l))
6. end for
7. k* = arg min, (∆objVal(k), k = 1, ⋯ , K);
8. Update objVal, label and cluSum accordingly;
9. end for

Fig. 3. The LocalSearch subroutine.

Lines 8–10 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 13 and 14 choose and return the best clustering result among the reps clusterings.

Next, we briefly discuss the convergence and complexity 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 is guaranteed to converge after finite iterations. The time complexity of SAIL is O(kdKnD), where l is the number of iterations for the convergence, K and n are the numbers of clusters and instances, respectively, and d is the average number of dimensions. As K is often small, and l is typically not beyond 20, the complexity of SAIL is often very low—just as the classic K-means algorithm. However, due to the complexity of the feasible region, SAIL may converge 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.

4. The V-SAIL algorithm

As SAIL is apt to converge to some local minima or saddle points, we here incorporate the Variable Neighborhood Search (VNS) scheme [9] to SAIL, 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 [9]. Fig. 4 schematically illustrates the search process of VNS. That is, VNS first finds an initial solution x, and then shakes in the k-th neighborhood Nk 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 Nk+1 of x. The set of neighborhoods are often defined by metric function in the solution space, e.g., Hamming distance, Euler distance, k-OPT operator, etc. The stopping condition for VNS can be the size of the neighborhood set, the maximum CPU time, and/or the maximum number of iterations.

Fig. 5 shows the pseudocodes of V-SAIL. Generally speaking, V-SAIL is a two-stage algorithm employing a clustering step and a refinement step. In c-step, the SAIL algorithm is called to generate a clustering result. This result serves as the departure 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 “LocalSearch” subroutine of SAIL is also called in VNS as a heuristic method to search for local minima. Some important details are as follows.

Lines 3–18 in Fig. 5 describe the r-step of V-SAIL. In Line 4, for the i-th neighborhood Ni, the “Shaking” function is called to generate a solution label in Ni that has a Hamming distance Hk = i × [D]/knmax to the current solution label. More specifically, “Shaking” first randomly selects Hk 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 Lines 6–11, the “LocalSearch” subroutine of SAIL is called to search for a local minimum initialized on label.

Fig. 4. Illustration of VNS.
[\text{objVal}^*, \text{label}^*, \pi] = \text{V-SAIL}(\text{para}, k_{\text{max}})

\textbf{Input:}
para = (D, K, reps, maxIter), as for SAIL
k_{\text{max}}: the number of neighborhood structures

\textbf{Output:}
\text{objVal}^*, \text{label}^*, \pi, \text{as for SAIL}

\textbf{Variable:}
\text{cluSum}, as for SAIL
\text{N}_i; the ith neighborhood with a Hamming distance \(i \times \|D\| / k_{\text{max}}\) from the current label

\textbf{Procedure}
1. \(\text{[\text{objVal}, \text{label}, \pi]} = \text{SAIL}(D, K, \text{reps}, \text{maxIter})\);
2. \(i = 1;\)
3. \text{while} \(i + + \leq k_{\text{max}}\)
4. \(\text{label}' = \text{Shaking}(\text{N}_i, \text{label});\)
5. \text{Update objVal' and cluSum' accordingly;}
6. \text{for } j = 1: \text{maxIter} - 1
7. \(\text{LocalSearch}(D, \pi, \text{objVal}', \text{label}', \text{cluSum}');\)
8. \text{if label}' is unchanged
9. \(\text{break;}\)
10. \text{end if}
11. \text{end for}
12. \(\text{if } \text{objVal}' < \text{objVal}\)
13. \(\text{objVal} = \text{objVal}', \text{label} = \text{label}', i = 1;\)
14. \text{end if}
15. \text{if the stopping condition is met}
16. \(\text{break;}\)
17. \text{end if}
18. \text{end while}
19. \text{return objVal}^* = \text{objVal}, \text{label}^* = \text{label}, \pi;

\text{Fig. 5.} The pseudocodes of V-SAIL.

This implies that SAIL is not only a clustering algorithm, but also a combinatorial optimization method. Then in Lines 12–14, 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. Line 15 stops VNS either if calling “Shaking” too many times or when a given CPU time is due.

In summary, V-SAIL well combines the complementary 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. However, it is worth noting that VNS is often very time-consuming, since it does not have a convergence strategy. As a result, V-SAIL is a better choice than SAIL only if the quality rather than runtime is the major concern of the clustering.

5. Experimental validation

In this section, we demonstrate the effectiveness of SAIL and its variants on high-dimensional sparse data clustering.

5.1. Experimental setup

\textbf{Data sets}: Ten real-world text data sets are used in our experiments, as listed in Table 2. “CV” is the coefficient of variation [11] used to characterize the class imbalance of 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 sparsity.

\textbf{Clustering tools}: In the experiments, we employ three types of clustering tools. The first one is SAIL and its variant V-SAIL, coded by ourselves in C++. The other two are well-known software packages for K-means clustering, i.e., MATLAB v7.1 [1] and CLUTO v2.1.1. [2]

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

CLUTO is a software package for clustering high-dimensional data sets. Its K-means implementation with cosine similarity as the proximity function shows superior performances in text clustering [27].

The parameters of the three 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 classes of each data set.

\textbf{Validation measures}: Many recent studies on clustering use the Normalized Mutual Information (NMI) to evaluate the clustering performance [28]. For consistency, we also use NMI in our experiments, which can be 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. Note that \(\sqrt{H(X)H(Y)}\) here serves as the normalization factor for \(I\). The value of NMI is in the interval: [0,1], and a larger value indicates a better clustering result. More computational details of NMI can be found in [21] from a contingency-table perspective.

5.2. The impact of zero-value 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), we select tr23 and tr45 as the test data sets and employ MATLAB Info-Kmeans on them. The clustering results are shown in Table 3, where CV_0 and CV_1 represent the distributions of the class and cluster sizes, respectively.

\begin{table}[h]
\centering
\caption{Experimental text data.}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
ID & Data set & #Instance & #Feature & #Class & CV & Density \\
\hline
1 & kib & 2340 & 21839 & 6 & 1.316 & 0.0068 \\
2 & lai & 3204 & 21604 & 6 & 0.493 & 0.0048 \\
3 & sports & 8580 & 126373 & 7 & 1.022 & 0.0010 \\
4 & tr11 & 414 & 6429 & 9 & 0.882 & 0.0438 \\
5 & tr12 & 313 & 5804 & 8 & 0.638 & 0.0471 \\
6 & tr23 & 204 & 5832 & 6 & 0.935 & 0.0661 \\
7 & tr31 & 927 & 10128 & 7 & 0.936 & 0.0265 \\
8 & tr41 & 878 & 7454 & 10 & 0.913 & 0.0262 \\
9 & tr45 & 690 & 8261 & 10 & 0.669 & 0.0340 \\
10 & wap & 1560 & 8460 & 20 & 1.040 & 0.0167 \\
\hline
\end{tabular}
\end{table}
As indicated by the close-to-zero NMI values, the clustering performance of MATLAB Info-Kmeans without smoothing is extremely poor. Also, by comparing the CV0 and CV1 values, we find 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 text vectors have been assigned to ONE cluster!

This experimental result well confirms our analysis in Section 2; that is, Info-Kmeans will face the serious zero-feature dilemma when clustering highly sparse data sets.

5.3. The comparison of SAIL and the smoothing technique

Here we illustrate the effect of the smoothing technique on sparse data sets. We use MATLAB Info-Kmeans and take seven text collections for illustration. 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 the comparison.

We also test SAIL on these data sets for the comparison purpose. The parameters are set as follows: (1) $\pi_x = 1$, $\forall x \in \Omega$; (2) no row or column modeling; (3) the initialization mode is "random center"; (4) reps = 1, but we repeat SAIL 10 times for each data set and return the average NMI value. Unless otherwise stated, these are the default settings of SAIL in our experiments.

Fig. 6 shows the comparison results. As can be seen, the clustering results of SAIL are consistently superior to the ones using the smoothing technique. For some data sets such as tr11, tr12, tr41 and tr45, SAIL takes the lead with a very wide margin.

5.4. The comparison of SAIL and spherical K-means

In this subsection, we compare the clustering performances between SAIL and the spherical K-means [27]. In the literature, people have shown that spherical K-means usually produces better clustering results than Info-Kmeans [28]. However, we would like to show in this experiment that the performance of SAIL is comparable to or even slightly better than the benchmark spherical K-means algorithm: CLUTO.

In this experiment, the parameter setting of CLUTO is: clmethod=direct, crfun=i2, sim=cosine, colmodel=none, ntrials=10. For SAIL, we use the default setting as for the previous experiment. Fig. 7 shows the clustering results,
where "CLUTO (no IDF)" indicates the spherical K-means clustering without using the IDF (Inverse-Document-Frequency) weighting [20]. As can be seen, SAIL beats CLUTO (no IDF) at all 10 data sets. For some data sets, e.g., la1, sports and tr31, SAIL even shows dominant advantages. This result demonstrates that SAIL is particularly suitable for high-dimensional data clustering, even compared with the state-of-the-art methods.

As it is reported that feature weighting often makes great impact to the spherical K-means, we also compare the clustering performances of SAIL and CLUTO with IDF. Fig. 7 shows the comparison result. Two observations are notable as follows. First, although CLUTO with IDF improves the clustering quality of CLUTO without IDF in five out of ten data sets, it still shows poorer performance than SAIL in eight out of ten data sets. Second, for some data sets, such as tr11, tr12, tr41, and tr45, the IDF scheme actually seriously degrades the clustering performance of CLUTO. These observations imply that feature weighting is an x-factor for spherical K-means without the guidance of external information. In contrast, SAIL with default setting shows stable clustering quality and therefore is more robust in practice.

5.5. The comparison of SAIL and V-SAIL

In this subsection, we take a further step to explore the properties of SAIL, and compare the performance of SAIL and V-SAIL.

We firstly investigate some factors that may impact the performance of SAIL. Specifically, we introduce the feature weighting, instance weighting and bisecting schemes for SAIL, and observe the variation of clustering performance. Table 4 shows the results. Note that "Default" represents SAIL with defaulting settings, "IDF" represents SAIL using the IDF scheme, and "Prob." represents SAIL with texts weighted by $x^k/\sum x^k$. 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 two cluster-selection criteria for bisecting SAIL, where "B-Size" chooses the largest cluster, and "B-Obj" chooses the one with the largest objective-function value.

As can be seen in Table 4, one observation is that "Prob." and "B-Obj" may produce very poor clustering results (scores underlined), and therefore cannot be used for SAIL. In contrast, "IDF" and "B-Size" produce relatively robust results, and in some cases even improve SAIL slightly (scores in bold), thus can be considered as valuable supplements to SAIL. Nonetheless, SAIL with default settings produces competitive clustering results in most cases, and therefore is the most robust one among different settings.

For the experiments of V-SAIL, we set $k_{\text{max}} = 12$, and return the average NMI values of 10 repetitions. The stopping criterion for V-SAIL is the maximum times for calling "LocalSearch", which we set to 500. Fig. 8 shows the comparison results of V-SAIL and SAIL. As can be seen from the figure, V-SAIL generally achieves higher NMI scores than SAIL. For some data sets, such as tr31, tr41 and tr45, the improvements are quite considerable. Nonetheless, it is worthy of noting that V-SAIL often consumes much more time than SAIL for the iterations of VNS.

6. Application to image data clustering

In this section, we exploit SAIL and V-SAIL algorithms for high-dimensional image clustering. We begin by briefly introducing the data set and the clustering procedure. Then we present the results and evaluations.

6.1. The image data

Our target is the oxford_5K data set, which is an image collection retrieved from Flickr, using 17 queries including several specific queries such as “All Souls Oxford”, “Balliol Oxford” and “Jesus Oxford”, and some general queries such as “Oxford” and “New Oxford”. In total, it contains 5060 images of 11 different Oxford landmarks—a landmark here means a particular part of a building.

Each image is then scanned for “salient” regions, and a high-dimensional descriptor is computed for each region. These descriptors are then quantized into a vocabulary of

Table 4
Impact of parameter settings of SAIL.

<table>
<thead>
<tr>
<th>ID</th>
<th>Data set</th>
<th>Default</th>
<th>IDF</th>
<th>Prob.</th>
<th>B-Size</th>
<th>B-Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>k1b</td>
<td>0.648</td>
<td>0.568</td>
<td>0.607</td>
<td>0.538</td>
<td>0.554</td>
</tr>
<tr>
<td>2</td>
<td>la1</td>
<td>0.595</td>
<td>0.571</td>
<td>0.559</td>
<td>0.578</td>
<td>0.574</td>
</tr>
<tr>
<td>3</td>
<td>sports</td>
<td>0.685</td>
<td>0.650</td>
<td>0.533</td>
<td>0.641</td>
<td>0.602</td>
</tr>
<tr>
<td>4</td>
<td>tr11</td>
<td>0.640</td>
<td>0.584</td>
<td>0.578</td>
<td>0.642</td>
<td>0.658</td>
</tr>
<tr>
<td>5</td>
<td>tr12</td>
<td>0.645</td>
<td>0.556</td>
<td>0.447</td>
<td>0.553</td>
<td>0.545</td>
</tr>
<tr>
<td>6</td>
<td>tr23</td>
<td>0.385</td>
<td>0.403</td>
<td>0.174</td>
<td>0.364</td>
<td>0.327</td>
</tr>
<tr>
<td>7</td>
<td>tr31</td>
<td>0.545</td>
<td>0.525</td>
<td>0.550</td>
<td>0.616</td>
<td>0.550</td>
</tr>
<tr>
<td>8</td>
<td>tr41</td>
<td>0.645</td>
<td>0.623</td>
<td>0.684</td>
<td>0.585</td>
<td>0.589</td>
</tr>
<tr>
<td>9</td>
<td>tr45</td>
<td>0.618</td>
<td>0.685</td>
<td>0.527</td>
<td>0.630</td>
<td>0.602</td>
</tr>
<tr>
<td>10</td>
<td>wap</td>
<td>0.584</td>
<td>0.543</td>
<td>0.594</td>
<td>0.558</td>
<td>0.548</td>
</tr>
</tbody>
</table>

Note: (1) Measured by average 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.
visual words. As a result, an image is represented by a bag of 1M visual words, with feature values being the numbers of word occurrences. Table 5 shows some characteristics of this image collection. Note that the features occurring less than 3 times across the entire data set are deleted.

As can be seen from Table 5, Oxford_5K has two notable features. First, it is a very sparse high-dimensional data set, whose density (the ratio of non-zero entities) is only slightly higher than 0.02%. Second, since the images are retrieved by simple queries, Oxford_5K contains a large amount of vague or even irrelevant images. Indeed, as pointed out in [15], if we take the images with more than 25% of the object clearly visible as normal instances, there are only 568 normal instances in Oxford_5K, or equivalently, roughly 88.77% instances are noise!

### 6.2. The clustering procedure

Given the Oxford_5K data set in Table 5, our task is to do image clustering, and then identify the 11 Oxford landmarks from the clusters. To this end, we must deal with two intractable challenges, i.e., the high sparsity and the vast noise. Fig. 9 shows some normal and particularly noise images for the two landmarks: All Souls and Ashmolean, which may seriously degrade the clustering quality. To address these challenges, we employ a three-step procedure as follows (the whole procedure is summarized in Fig. 10):

- **Step 1: Noise removal.** Since there are huge volume of noise in the data, we first launch a noise-filtering step based on the CosMiner algorithm proposed in [22]. That is, we first transform Oxford_5K into a transaction data (or equivalently, the market-basket data), and then employ CosMiner on the new data for the so-called “cosine interesting patterns” (CIPs), i.e., itemsets that have support and cosine values larger than the user-specified support and cosine thresholds, respectively [22, 23]. In our case, a CIP is a set of words that occurs simultaneously in some images. Images that cover no CIP are treated as noise, and therefore removed from the data.

- **Step 2: Image clustering.** In this step, we employ SAIL and V-SAIL on the data after noise removal. The number of clusters $K$ is set to 11.

- **Step 3: Landmark recognition.** In this step, we try to recognize the 11 landmarks from the resultant 11 clusters. Since the noise may not be fully removed in the first step, we manually remove the remaining noise in each cluster, and have the landmark in majority as the landmark of that cluster.

### 6.3. The results

Table 6 shows the parameter setting and the results of noise removal using CosMiner, where $\min_{\text{supp}}$ and $\min_{\text{cos}}$.
cos are the minimum support and cosine thresholds respectively for cosine interesting pattern mining. We consider three cases for the purpose of comparison in the subsequent image clustering. In Case 1, we do not employ noise removal, and therefore have all the 5060 instances remained. In Case 2, we set a relatively lower min_supp and generate over 12,000 patterns, which lead to the remaining 1001 instances. Finally, in Case 3, we generate less patterns and therefore have only 559 instances remained. It is noteworthy that less instances can lower the manual labor for noise removal, but may increase the risk of missing too many normal images removed by CosMiner as noise.

We then employ SAIL and V-SAIL for the remaining images, remove the noise in each cluster, and recognize the landmarks from the resultant clusters. Results returned by V-SAIL in Case 2 indicate that nine out of 11 landmarks are successfully found. Fig. 11 shows 10 images for each of the four sample landmarks identified. Among them, All Souls and Radcliffe Cam are recognized easily. For instance, in the cluster for All Souls, 62 out of 63 instances are the correct images, which comes near to perfection. In contrast, the other two landmarks, i.e., Magdalen and Ashmolean, are much harder to recognize. For instance, the cluster for Magdalen contains 12 instances, but only half of them are the correct images (the images in the rectangle are the incorrect ones in the cluster). Finally, it is noteworthy that the two landmarks, i.e., Cornmarket and Keble, cannot be recognized from the clusters due to the rareness of the samples—there are only one and three images for these two landmarks in the data.

Table 7 lists the results of clustering evaluation for the three cases in terms of the NMI measure. For each case, we do clustering 5 times and compute the average score after manual noise-removal. One observation is that, without noise removal in the first step, the clustering quality of Case 1 is much poorer than the ones of the other cases. This is because vast noise will exert very bad influence on the centroid computation of SAIL, and eventually bias the clustering result. Moreover, although the clustering quality of Case 3 is the best, it indeed misses many normal images such that four landmarks totally disappear in the clusters. This implies that we should find a good balance between noise removal and image completeness. That is also why we regard Case 2 as the best one among the three cases.

In summary, both SAIL and V-SAIL show excellent clustering performance in the application of landmark image clustering, even with the presence of vast noise.

7. Related work

In the literature, great research efforts have been taken to incorporate information theoretic measures into existing clustering algorithms, such as K-means [3, 4, 28, 10]. 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 [3]. 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 [4]. This algorithm is the two-dimensional 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.

Since many other proximity functions such as the squared Euclidean distance and the cosine similarity can also be used in K-means clustering [17], a natural idea is to compare their performances in practice [28] is one of such studies. The authors argued that the spherical K-means produces better clustering results than InfoKmeans [16, 27] also showed the merits of spherical K-means in text clustering. However, these studies did

---

**Table 6**

<table>
<thead>
<tr>
<th>Case</th>
<th>Parameter setting</th>
<th>#Patterns</th>
<th>#Images</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min_supp</td>
<td>min_cos</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>–</td>
<td>–</td>
<td>5060</td>
</tr>
<tr>
<td>2</td>
<td>0.09%</td>
<td>0.48</td>
<td>12,675</td>
</tr>
<tr>
<td>3</td>
<td>0.10%</td>
<td>0.48</td>
<td>1327</td>
</tr>
</tbody>
</table>

---

Please cite this article as: J. Cao, et al., Towards information-theoretic K-means clustering for image indexing, Signal Processing (2012), http://dx.doi.org/10.1016/j.sigpro.2012.07.030
Fig. 11. Four sample landmarks recognized using V-SAIL.
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 SAIL and V-SAIL to handle the zero-feature dilemma, and improving the clustering performance of Info-Kmeans to a level competitive to the spherical K-means.

8. Concluding remarks

This paper studied the problem of information-theoretic K-means clustering (Info-Kmeans) for high-dimensional sparse data, such as images and texts featured by the bag-of-words model. In particular, we revealed the zero-feature dilemma of Info-Kmeans in assigning objects to the centroids. To deal with it, we developed a Summation-based Incremental Learning (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 the V-SAIL algorithm to further enhance the clustering ability of SAIL. Finally, as demonstrated by extensive benchmark data sets in the experiment and a challenging image data set in the application, SAIL and V-SAIL can greatly improve the performance of Info-Kmeans on high-dimensional sparse data, even with a huge volume of noise inside.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (Nos. 71072172, 61103229), Jiangsu Provincial Colleges and Universities Outstanding S&T Innovation Team Fund (No. 2001013), and Key Project of Natural Science Research in Jiangsu Provincial Colleges and Universities (No. 12KJA520001). The third author was also supported in part by the National Natural Science Foundation of China (Nos. 70901002, 71171007, 71031001, 70890080, 90924020).

References


Table 7

Performance of image clustering (measured by NMI).

<table>
<thead>
<tr>
<th>Case</th>
<th>i</th>
<th>ii</th>
<th>iii</th>
<th>iv</th>
<th>v</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SAIL</td>
<td>0.105</td>
<td>0.095</td>
<td>0.083</td>
<td>0.090</td>
<td>0.079</td>
</tr>
<tr>
<td>V-SAIL</td>
<td>0.115</td>
<td>0.106</td>
<td>0.097</td>
<td>0.084</td>
<td>0.091</td>
<td>0.099</td>
</tr>
<tr>
<td>2</td>
<td>SAIL</td>
<td>0.601</td>
<td>0.633</td>
<td>0.576</td>
<td>0.541</td>
<td>0.523</td>
</tr>
<tr>
<td>V-SAIL</td>
<td>0.643</td>
<td>0.627</td>
<td>0.646</td>
<td>0.620</td>
<td>0.599</td>
<td>0.627</td>
</tr>
<tr>
<td>3</td>
<td>SAIL</td>
<td>0.652</td>
<td>0.632</td>
<td>0.550</td>
<td>0.651</td>
<td>0.595</td>
</tr>
<tr>
<td>V-SAIL</td>
<td>0.702</td>
<td>0.616</td>
<td>0.652</td>
<td>0.642</td>
<td>0.561</td>
<td>0.635</td>
</tr>
</tbody>
</table>


