Scalable Clustering Algorithms with Balancing Constraints

Arindam Banerjee
Dept. of Computer Science & Engg
University of Minnesota, Twin Cities
banerjee@cs.umn.edu

Joydeep Ghosh
Dept. of Electrical and Computer Engg
University of Texas at Austin
ghosh@ece.utexas.edu

Abstract

Clustering methods for data-mining problems must be extremely scalable. In addition, several data mining applications demand that the clusters obtained be balanced, i.e., be of approximately the same size or importance. In this paper, we propose a general framework for scalable, balanced clustering. The data clustering process is broken down into three steps: sampling of a small representative subset of the points, clustering of the sampled data, and populating the initial clusters with the remaining data followed by refinements. First, we show that a simple uniform sampling from the original data is sufficient to get a representative subset with high probability. While the proposed framework allows a large class of algorithms to be used for clustering the sampled set, we focus on some popular parametric algorithms for ease of exposition. We then present algorithms to populate and refine the clusters. The algorithm for populating the clusters is based on a generalization of the stable marriage problem, whereas the refinement algorithm is a constrained iterative relocation scheme. The complexity of the overall method is $O(kN \log N)$ for obtaining k balanced clusters from N data-points, which compares favorably with other existing techniques for balanced clustering. In addition to providing balancing guarantees, the clustering performance obtained using the proposed framework is comparable to and often better than the corresponding unconstrained solution. Experimental results on several datasets, including high-dimensional ($> 20,000$) ones, are provided to demonstrate the efficacy of the proposed framework.

1 Introduction

The past few years have witnessed a growing interest in clustering algorithms that are suitable for data-mining applications [24, 23, 14, 17]. Clustering algorithms for data-mining problems must be extremely scalable. In addition, several data mining applications demand that the clusters obtained be (roughly) balanced, i.e., be of approximately the same size or importance. As elaborated later on, balanced groupings are sought in a variety of real-life scenarios as they make the results more useful or actionable towards fulfilling a business need.

There are several notable approaches that address the scalability issue. Some approaches try to build the clusters dynamically by maintaining sufficient statistics and other summarized information in main memory while minimizing the number of database scans involved. For example, Bradley *et al.* [7, 8] propose out-of-core methods that scan the database once to form a summarized model (for instance, the size, sum and sum-squared values of potential clusters, as well as a small number of unallocated data-points) in main memory. Subsequent refinements based on this summarized information is then restricted to main memory operations without resorting to further disk scans.

Another method with a similar flavor [38] compresses the data objects into many small subclusters using modified index trees and performs clustering with these subclusters. A different approach is to subsample the original data before applying the actual clustering algorithms [10, 19]. Ways of effectively sampling large datasets have also been proposed [32]. [12] suggest using less number of points in each step of an iterative relocation optimization algorithm like `kmeans` as long as the model produced does not differ significantly from the one that would be obtained with full data.

The computational complexity of several of these methods are linear per iteration in the number of data-points N as well as the number of clusters k and hence scale very well. However their "sequential cluster building" nature prevents a global view of the emergent clustering that is needed for obtaining well-formed as well as reasonably balanced clusters. Even the venerable `KMeans` does not have any explicit way to guarantee that there is at least a certain minimum number of points per cluster, though it has been shown that `KMeans` has an implicit way of preventing highly skewed clusters [26]. It has been empirically observed by several researchers [6, 18, 11] that `KMeans` and related variants quite often generate some clusters that are empty or extremely small, when both the input dimensionality and the number of clusters is in the tens or more. In fact, several commercial clustering packages, including Mathwork's implementation of `KMeans`, have a check for zero sized clusters, typically terminating with an error message when this happens.

Having null or very small clusters is undesirable in general. In fact, for certain practical applications one requires quite the opposite, namely that the clusters be all of comparable sizes. Here "size" of a cluster refers either to the number of data points in that cluster, or to the net value of all points in cluster in situations where different points may have different values or weights associated with them. This balancing requirement comes from the associated application/business needs rather than from the inherent properties of the data, and helps in making the clusters actually useful and actionable. Some specific examples are:

- Direct Marketing [35, 37]: A direct marketing campaign often starts with segmenting customers into groups of roughly equal size or equal estimated revenue generation, (based on, say, market basket analysis, demographics, or purchasing behavior at a web site), so that the same number of sales teams, marketing dollars, etc., can be allocated to each segment.

- Category Management [31]: Category management is a process that involves managing product categories as business units and customizing them on a store-by-store basis to satisfy customer needs. A core operation in category management is to group products into categories of specified sizes such that they match units of shelf space or floor space. This is an important balanced clustering application for large retailers such as Walmart [34]. Another operation key to large consumer product companies such as Procter & Gamble, is to group related stock keeping units (SKUs) in bundles of comparable revenues or profits. In both operations the clusters need to be refined on an on-going basis because of seasonal difference in sales of different products, consumer trends etc. [21].

- Clustering of Documents [27, 2]: In clustering of a large corpus of documents to generate topic hierarchies, balancing greatly facilitates *browsing/navigation* by avoiding the generation of hierarchies that are highly skewed, with uneven depth in different parts of the hierarchy "tree" or having widely varying number of documents at the leaf nodes. Similar principles apply when grouping articles in a website [27], portal design, and creation of domain specific ontologies by hierarchically grouping concepts.

- Balanced Clustering in Energy Aware Sensor Networks [16, 20]: In distributed sensor networks, sensors are clustered into groups, each represented by a sensor "head", based on attributes such as spatial location, protocol characteristics, etc. An additional desirable property, often imposed as an external soft constraint on clustering, is that each group consume comparable amounts of power, as this makes the overall network more reliable and scalable.

From the examples above, it is clear that a balancing constraint is typically imposed because of application needs rather than from observing the actual distribution of the data. Clusters that are too small may not be useful, e.g., a group of otherwise very similar customers that is too small to provide customized solutions for. Similarly very large cluster may not be differentiated enough to be readily actionable. Sometimes, even the desired range of the number of clusters sought, say 5 to 10 in a direct marketing application, comes from high level requirements rather than from data properties. Thus it may happen that balancing is sought even though the "natural" clusters in the data are quite imbalanced. Similarly the most appropriate number of clusters determined from a purely data-driven perspective may not match the number obtained from a need-driven one. In such cases, constrained clustering may yield solutions that are of poorer quality when measured by a data-centric criterion such as "average dispersion from cluster representative" (`KMeans` objective function), even though these same solutions are more preferable from the application viewpoint. In this paper, we report experimental results on several datasets where the associated class priors range from equal valued to highly varying ones, to study this aspect. A positive, seemingly surprising result from the empirical results is that even for fairly imbalanced data our approach to balanced clustering provides comparable, and sometimes better results as judged by the unconstrained clustering objective function. Thus balanced clusters are clearly superior if the benefit of meeting the constraints is also factored in. The advantage is largely because balancing provides a form of regularization that seems to avoid low-quality local minima stemming from poor initialization.

There are a few existing approaches for obtaining balanced clusters. First, an agglomerative clustering method can be readily adapted so that once a cluster reaches a certain size in the bottom-up agglomeration process, it can be removed from further consideration. However, this may significantly impact cluster quality. Moreover, agglomerative clustering methods have a complexity of $\Omega(N^2)$ and hence does not scale well. A recent approach to obtain balanced clusters is to convert the clustering problem into a graph partitioning problem [25, 35]. A weighted graph is constructed whose vertices are the data-points. An edge connecting any two vertices has a weight equal to a suitable similarity measure between the corresponding data-points. The choice of the similarity measure quite often depends on the problem domain, e.g., Jaccard coefficient for market-baskets, normalized dot products for text, etc. The resultant graph can be partitioned by efficient "min-cut" algorithms such as METIS [25] that also incorporate a soft balancing constraint. Though this approach gives very good results, $\Omega(N^2)$ computation is required just to compute the similarity matrix. Another approach is to iterate over `kmeans` but do the cluster assignment by solving a minimum cost flow problem satisfying constraints [6] on the cluster sizes. This approach is $O(N^3)$ and has even poorer scaling properties. The cost-flow approach was also used in [16] to obtained balanced groupings in energy aware sensor networks.

An alternative approach to obtain balanced clustering is via frequency sensitive competitive learning methods for clustering [1], where clusters of larger sizes are penalized so that points are less likely to get assigned to them. Such a scheme can be applied both in the batch as well as in the online settings [4]. Although frequency sensitive assignments can give fairly balanced clusters

in practice, there is no obvious way to guarantee that every cluster will have at least a pre-specified number of points.

A pioneering study of constrained clustering in large databases was presented by Tung et. al. [36]. They describe a variety of constraints that may be imposed on a clustering solution, but subsequently focus solely on a balancing constraint on certain key objects called pivot objects. They start with any clustering (involving *all* the objects) that satisfies the given constraints. This solution is then refined so as to reduce the clustering cost, measured as net dispersion from nearest representatives, while maintaining the constriant satisfaction. The refinement proceeds in two steps: pivot movement and deadlock resolution, both of which are shown to be NP-hard. They propose to scale their approach by compressing the objects into several tight "micro-clusters"[7] where possible, in a pre-clustering stage, and subsequently doing clustering at the micro-cluster level. However, since this is a coarse grain solution, an option of finer grain resolution needs to be provided by allowing pivot points to be shared among multiple micro-clusters. This last facility helps to improve solution quality, but negates some of the computational savings in the process.

In this paper, we address the issue of developing scalable clustering algorithms that satisfy balancing constraints on the cluster sizes, i.e., the number of objects in each cluster. [1] We present a method for clustering N data-points into k clusters so that each cluster has at least m points for some given $m \leq \frac{N}{k}$. The overall complexity of the method is $O(kN \log N)$. The proposed method can be broken down into three steps: (1) sampling, (2) clustering of the sampled set and (3) populating and refining the clusters while satisfying the balancing constraints. We address each of the steps separately and show that the three-step method gives a very general framework for scaling up balanced clustering algorithms. The post-processing refinement step is similar in spirit to the pivot movement step in [36], though it differs in the actual procedure. However, the overall framework is fundamentally different. Its efficiency really stems from the first two steps, and there is no need to resort to summarized micro-clusters for computational savings.

The rest of the paper is organized as follows. A brief overview of the three steps is presented in section 2. The three steps are discussed and analyzed in detail in sections 3, 4 and 5. In section 6 we present experimental results on high-dimensional text clustering problems. The paper concludes in section 7 with a discussion on the proposed framework.

2 Overview

Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \mathbb{R}^d, \forall i$, be a set of data-points that needs to be clustered. Let $d : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}_+$ be a given distance function between any two points in $\mathbb{R}^d$. The *clustering problem* this article addresses is that of finding a disjoint k-partitioning $\{S_h\}_{h=1}^k$ of $\mathcal{X}$ and a corresponding set of k cluster representatives $M = \{\mu_h\}_{h=1}^k$ in $\mathbb{R}^d$ for a given k such that the clustering objective function

$$\mathbb{L}(\{\mu_h, S_h\}_{h=1}^k) = \sum_{h=1}^{k} \sum_{\mathbf{x} \in S_h} d(\mathbf{x}, \mu_h) \tag{1}$$

[1] Note that both `KMeans` type partitional algorithms and graph-partitioning approaches can be readily generalized to cater to weighted objects [5, 35]. Therefore, our framework can be easily extended to apply to situations where balancing is desired based on a derived quantity such as net revenue per cluster, since such situations are dealt with by assigning a corresponding weight to each object.

is minimized under the constraint that $|S_h| \geq m, \forall h$, for a given m with $mk \leq N$. Further, we assume that for the *optimal* partitioning $\{S_h^*\}_{h=1}^k$, we have

$$\min_h \frac{|S_h^*|}{N} \geq \frac{1}{l} \tag{2}$$

for some integer $l \geq k$. In other words, if samples are drawn uniformly at random from the set $\mathcal{X}$, the probability of picking a sample from any particular optimal partition is at least $\frac{1}{l}$ for some given $l \geq k$. Note that $l = k$ if and only if $|S_h^*| = N/k, \forall h$, so that the optimal partitions are exactly of equal size and the clusters are all of equal size.

For the clustering problem to be tractable, the distance function d has to be well-behaved in the following sense: Given a set of points $\mathbf{x}_i, \cdots, \mathbf{x}_n$, there should be an efficient way of finding the representative

$$\boldsymbol{\mu} = \operatorname*{argmin}_c \sum_{i=1}^{n} d(\mathbf{x}_i, c) \ .$$

While a large class of distance functions are well-behaved in the above sense, in this article, we focus on the squared Euclidean distance and the cosine distance, for both of which the representative can be computed in $O(n)$ time. In fact, it can be shown that the representative can be computed in $O(n)$ for all Bregman divergences [5]. For the general framework discussed in this article, any meaningful distance function can potentially be used as long as the computation of the representative can be done efficiently. Further, the scalability analysis can be extended to non-parametric clustering algorithms, where explicit representative computation is absent.

To make the solution of this constrained clustering problem scalable, we break up the solution into three steps making use of the information regarding the nature of the optimal clustering. The three steps in the proposed scheme are as follows:

Step 1 *Sampling of the given data:* The given data is first sampled in order to get a small representative subset of the data. One potential problem with sampling is that one may end up sampling all the points from only a few of the optimal partitions so that the sampled data is not a representative subset for clustering. We show that this kind of a scenario is unlikely under the assumptions of (2) and compute the number of samples we must must draw from the original data in order to get a good representation from each of the optimal partitions in the sampled set with high probability.

Step 2 *Clustering of the sampled data:* Any clustering algorithm that fits the problem domain can be used in this stage. The algorithm is run on the sampled set. Since the size of the sampled set is much less than that of the original data, one can use slightly involved algorithms as well, without much blow-up in complexity. It is important to note that the general framework works for a wide class of distance (or similarity) based clustering algorithms as long as there is a concept of a representative of a cluster and the clustering objective is to minimize the average distance (or maximize the similarity) to the corresponding representatives.

Step 3 *Populating and Refining the clusters:* The third step has two parts: Populate and Refine. In the first part, the small clusters generated in the second step are populated with the datapoints that were not sampled in the first step. For populating, we propose an algorithm `poly-stable`, motivated by the stable marriage problem [22], that satisfies the requirement

on the number of points per cluster, thereby generating a *feasible* and reasonably good clustering.

In the second part, an iterative refinement algorithm `refine` is applied on this stable feasible solution to monotonically decrease the clustering objective function while satisfying the constraints all along. On convergence, the final partitioning of the data is obtained.

Note that populating and refining of the clusters can be applied irrespective of what clustering algorithm was used in the second step, as long as there is a way to represent the clusters. In fact, the third step is the most critical step and the first two steps can be considered a good way to initialize the populate-refine step.

The three steps outlined above provide a very general framework for scaling up clustering algorithms under mild conditions, irrespective of the domain from which the data has been drawn and the clustering algorithm that is used as long as they satisfy the assumptions.

3 Sampling

First, we examine the question: given the set $\mathcal{X}$ having k underlying (but unknown) optimal partitions $\{S_h^*\}_{h=1}^k$, and that the size of the smallest partition is at least $\frac{|\mathcal{X}|}{l}$, $l \geq k$, what is the number n of samples that need to be drawn from $\mathcal{X}$ so that there are at least $s \gg 1$ points from each of the k partitions with high probability? Let X be a random variable for the number of samples that need to be drawn from $\mathcal{X}$ to get at least s points from each cluster. $\mathbf{E}[X]$ can be computed by an analysis using the theory of recurrent events and renewals [15]. A simpler version of this analysis can be found in the so-called Coupon Collector's problem [30], or, equivalently in the computation of the cover time of a random walk on a complete graph. With a similar analysis in this case, we get the following result.

Lemma 1 *If X is the random variable for the number of samples be drawn from $\mathcal{X}$ to get at least s points from each partition, $\mathbf{E}[X] \leq sl \ln k + O(sl)$.*

Proof: Let $C_1, C_2, \cdots, C_X$ be the sequence of samples drawn where $C_h \in \{1, \cdots, k\}$ denotes the partition number from which the ith sample was drawn. We call the h-th sample C_h *good* if less than s samples have been drawn from partition C_h in the previous $(h-1)$ draws. Note that the first s samples are always good. The sequence of draws is divided into epochs where epoch i begins with the draw following the draw of the ith good sample and ends with the draw on which the $(i+1)$-th good sample is drawn. Since a total of sk good samples have to be drawn, the number of epochs is sk. Let X_i, $0 \leq i \leq (sk-1)$, be a random variable defined to be the number of trials in the ith epoch, so that

$$X = \sum_{i=0}^{sk-1} X_i \ .$$

Let p_i be the probability of success on any trial of the ith epoch. A sample drawn in the ith epoch is not good if the draw is made from a partition from which s vertices have already been drawn. In the ith epoch there can be at most $\lfloor \frac{i}{s} \rfloor$ partitions from which s samples have been already drawn. Hence, there are still at least $(k - \lfloor \frac{i}{s} \rfloor)$ partitions from which good samples can be drawn. Then,

since the probability of getting a sample from any particular partition is at least $\frac{1}{l}$, the probability of the sample being good is

$$p_i \geq \sum_{j=1}^{k-\lfloor \frac{i}{s} \rfloor} \frac{1}{l} = \frac{k - \lfloor \frac{i}{s} \rfloor}{l} \ .$$

The random variable X_i is geometrically distributed with parameter p_i and hence

$$\mathbf{E}[X_i] = \frac{1}{p_i} \leq \frac{l}{k - \lfloor \frac{i}{s} \rfloor} \ .$$

Therefore, by linearity of expectation,

$$\begin{aligned}
\mathbf{E}[X] &= \mathbf{E}[\sum_{i=0}^{sk-1} X_i] = \sum_{i=0}^{sk-1} \mathbf{E}[X_i] \\
&\leq \sum_{i=0}^{sk-1} \frac{l}{k - \lfloor \frac{i}{s} \rfloor} = l \sum_{j=1}^{k} \sum_{h=0}^{s-1} \frac{1}{k - \lfloor \frac{(j-1)s+h}{s} \rfloor} \\
&= sl \sum_{j=1}^{k} \frac{1}{k-(j-1)} = sl \sum_{i=1}^{k} \frac{1}{i} \\
&\approx sl \ln k + O(sl) \ .
\end{aligned}$$

That completes the proof. ∎

Using this lemma, we present the following result that shows that if we draw $n = csl \ln k \approx c\mathbf{E}[X]$ samples from $\mathcal{X}$, where c is an appropriately chosen constant, then we we will get at least s samples from each optimal partition with high probability.

Lemma 2 *If $csl \ln k$ samples are drawn uniformly at random from $\mathcal{X}$ the probability of getting at least $s \gg 1$ samples from each of the optimal partitions is more than $(1 - \frac{1}{k^d})$, where c and d are constants such that $c \geq \frac{1}{\ln k}$ and $d \leq \frac{s}{\ln k}\{c \ln k - \ln(4c \ln k)\} - 1$.*

Proof: Let E_h^n denote the event that less than s points have been sampled from the partition S_h^* in the first n draws. Let q_h be the probability that a uniformly random sample is chosen from S_h^*. Then, $q_h \geq \frac{1}{l}$. We shall consider the number of draws $n = csl \ln k$ for a constant $c \geq \frac{1}{\ln k}$. Then,

$$\mathbf{Pr}[E_h^n] = \sum_{j=0}^{s-1} \binom{n}{j} q_h^j (1-q_h)^{n-j} \leq \sum_{j=0}^{s-1} \binom{n}{j} \left(\frac{1}{l}\right)^j \left(1 - \frac{1}{l}\right)^{n-j} \ .$$

Now, since the expectation for the binomial distribution $B(csl \ln k, \frac{1}{l})$ is $cs \ln k \geq s > (s-1)$, the largest term in the summation is the term corresponding to $i = s-1$. Hence,

$$\mathbf{Pr}[E_i^n] \leq s \binom{n}{s-1} \left(\frac{1}{l}\right)^{s-1} \left(1 - \frac{1}{l}\right)^{n-s+1} \leq s \binom{n}{s} \left(\frac{1}{l}\right)^{s} \left(1 - \frac{1}{l}\right)^{n-s} \ .$$

Since $n \left(\frac{1}{l}\right) \left(\frac{l-1}{l}\right) \approx \frac{n}{l} = cs \ln k \geq s \gg 1$, applying the Poisson approximation [33],

$$\binom{n}{s} \left(\frac{1}{l}\right)^{s} \left(1 - \frac{1}{l}\right)^{n-s} \approx e^{-\frac{n}{l}} \frac{\left(\frac{n}{l}\right)^s}{s!} \ .$$

Hence,

$$\mathbf{Pr}[E_i^n] \leq se^{-\frac{n}{l}} \frac{\left(\frac{n}{l}\right)^s}{s!} \ .$$

Using the fact that the probability of a union of events is always less than the sum of the probabilities of these events, we obtain

$$\mathbf{Pr}[X > n] = \mathbf{Pr}[\cup_{i=1}^k E_i^n] \leq \sum_{i=1}^k \mathbf{Pr}[E_i^n] \leq kse^{-\frac{n}{l}} \frac{\left(\frac{n}{l}\right)^s}{s!} \ .$$

Putting $n = csl \ln k$, we get

$$\begin{aligned}
\mathbf{Pr}[X > n] &\leq kse^{-\frac{n}{l}} \frac{\left(\frac{n}{l}\right)^s}{s!} \\
&= kse^{-cs\ln k} \frac{(cs\ln k)^s}{s!} \\
&\leq (ce)^s sk^{1-cs} (\ln k)^s \qquad (\text{since } s! \geq \left(\frac{s}{e}\right)^s) \\
&= \left(cek^{\frac{1}{s}-c} \ln k s^{\frac{1}{s}}\right)^s \ .
\end{aligned}$$

Then $\mathbf{Pr}[X \leq n] \geq (1 - \frac{1}{k^d})$, or, $\mathbf{Pr}[X > n] \leq \frac{1}{k^d}$ if $\left(cek^{\frac{1}{s}-c} \ln k s^{\frac{1}{s}}\right)^s \leq \frac{1}{k^d}$, which is true if $k^{c-\frac{d+1}{s}} \geq ces^{\frac{1}{s}} \ln k$. Now, $s^{\frac{1}{s}} \leq e^{\frac{1}{e}}$, $\forall s$, and hence $es^{\frac{1}{s}} \leq ee^{\frac{1}{e}} < 4$. So, the probability bound is satisfied if $k^{c-\frac{d+1}{s}} \geq 4c \ln k$. A simple algebraic manipulation of the last inequality gives our desired upper bound on d. ∎

To help understand the result, consider the case where $l = k = 10$ and we want at least $s = 50$ points from each of the partitions. Table 1 shows the total number of points that need to be sampled for different levels of confidence. Note that if the k optimal partitions are of equal size and

d	1	2	3	4	5
Confidence, $100(1 - \frac{1}{k^d})\%$	90.000	99.000	99.900	99.990	99.999
Number of Samples, n	1160	1200	1239	1277	1315

Table 1: Number of samples required to achieve a given confidence level for k=10 and s=50

$csk \ln k$ points are sampled uniformly at random, the expected number of points from each partition is $cs \ln k$. Thus the underlying structure is expected to be preserved in this smaller sampled set and the chances of wide variations from this behavior is very small. For example, for the 99.99% confidence level in Table 1, the average number of samples per partition is 127, which is only about 2.5 times the minimum sample size that is desired, irrespective of the total number of points in $\mathcal{X}$, which could be in the millions for example.

4 Clustering of the Sampled Set

The second step involves clustering the set of n sampled points, $\mathcal{X}_s$. The only requirement from this stage is to obtain a k-partitioning of $\mathcal{X}_s$ and have a representative $\boldsymbol{\mu}_h, h = 1, \cdots, k$ corresponding

to each partition. There are several clustering formulations that satisfy this requirement, e.g., clustering using Bregman divergences [5] for which the optimal representatives are given by the centroids, clustering using cosine-type similarities [11, 3] for which the optimal representatives are given by the ℓ_2-normalized centroids, convex clustering [29] for which the optimal representatives are given by generalized centroids, etc. Since there are already a large number of clustering formulations that satisfy our requirement and the main issue we are trying to address is how to make the existing clustering algorithms scalable, we do not propose any new algorithm for clustering the sampled set. Instead, we just review two widely used existing algorithms, viz Euclidean kmeans clustering [28] and spherical kmeans clustering [11], for which experimental results will be presented.

4.1 Euclidean kmeans

Euclidean kmeans takes the set of n points to be clustered, $\mathcal{X}_s$, and the number of clusters, k, as an input. The Euclidean kmeans problem [28, 13] is to get a k-partitioning $\{S_h\}_{h=1}^k$ of $\mathcal{X}_s$ such that the following objective function,

$$\mathcal{J}_{\text{kmeans}} = \frac{1}{n} \sum_{h=1}^{k} \sum_{\mathbf{x} \in S_h} (\mathbf{x} - \mu_h)^2,$$

is minimized. The `kmeans` algorithm gives a simple greedy solution to this problem that guarantees a local minimum of the objective function. The algorithm starts with a random guess for $\{\mu_h\}_{h=1}^k$. At every iteration, each point x is assigned to the partition S_{h^*} corresponding to its nearest centroid μ_{h^*}. After assigning all the points, the centroids of each partition is computed. These two steps are repeated till convergence. It can be shown that the iterations converge in a finite number of iterations to a local minimum of the objective function. The reason we chose the `kmeans` algorithm is its wide usage. Further, since `kmeans` is a special case of Bregman clustering algorithms [5] as well as convex clustering algorithms [29], the experiments of section 6 gives some intuition as to how the proposed framework will perform in more general settings.

4.2 Spherical kmeans

Spherical kmeans [11] is applicable to data points on the surface of the unit hypersphere and has been successfully applied and extended to clustering of natural high-dimensional datasets [3]. The spherical kmeans problem is to get a k-partitioning $\{S_h\}_{h=1}^k$ of a set of n data points $\mathcal{X}_s$ on the unit hypersphere such that the following objective function,

$$\mathcal{J}_{\text{spkmeans}} = \frac{1}{n} \sum_{h=1}^{k} \sum_{\mathbf{x} \in S_h} \mathbf{x}^T \mu_h,$$

is maximized. The `spkmeans` algorithm [11] gives a simple greedy solution to this problem that guarantees a local maximum of the objective function. Like `kmeans`, `spkmeans` starts with a random guess for $\{\mu_h\}_{h=1}^k$. At every iteration, each point x is assigned to the partition S_{h^*} corresponding to its most similar centroid μ_{h^*}, and finally, the centroids of each partition are computed. It can be shown that the algorithms converge in a finite number of iterations to a local maximum of the objective function. Successful application of this algorithm to natural datasets, e.g., text, gene-expression data, etc., gives us the motivation to study the algorithm in greater detail.

5 Populating and Refining the Clusters

After clustering the n point sample from the original data $\mathcal{X}$, the remaining $(N-n)$ points can be assigned to the clusters, satisfying the balancing constraint. Subsequently, these clusters need to be refined to get the final partitioning. This is the final and perhaps most critical step since it has to satisfy the balancing requirements while maintaining the scalability of the overall approach. In fact, the previous two steps can essentially be considered as a good way of finding an initialization for an iterative refinement algorithm that works with $\mathcal{X}$ and minimizes the objective function $\mathbb{L}$. In this section, we discuss a novel scheme for populating the clusters and then iteratively refining the clustering.

5.1 Overview

There two parts in the proposed scheme:

1. Populate: First, the points that were not sampled, and hence do not currently belong to any cluster, are assigned to the existing clusters in a manner that satisfies the balancing constraints while ensuring good quality clusters.

2. Refine: Iterative refinements are done to improve on the clustering objective function while satisfying the balancing constraints all along.

Hence, part 1 gives a reasonably good feasible solution, i.e., a clustering in which the balancing constraints are satisfied. Part 2 iteratively refines the solution while always remaining in the feasible space.

Let n_h be the number of points in cluster h, so that $\sum_{h=1}^{k} n_h = n$. Let $\mathcal{X}_u = \{\mathbf{x}_1, \cdots, \mathbf{x}_{N-n}\}$ be the set of $(N-n)$ non-sampled points. The final clustering needs to have at least m points per cluster to be feasible. Let $b = \frac{mk}{N}$, where $0 \leq b \leq 1$ since $m \leq N/k$, be the *balancing fraction*. For any assignment of the members of $\mathcal{X}$ to the clusters, let $\ell_i \in \{1, \ldots, k\}$ denote the cluster assignment of $\mathbf{x}_i$. Further, let $S_h = \{\mathbf{x}_i \in \mathcal{X} | \ell_i = h\}$.

5.1.1 Part 1: Populate

In the first part, we just want a reasonably good feasible solution so that $|S_h| \geq m, \forall h$. Hence, since there are already n_h points in S_h, we need to assign $[m - n_h]_+$ more points to S_h, where $[x]_+ = \max(x, 0)$. Ideally, each point in $\mathcal{X}_u$ should be assigned to the nearest cluster so that $\forall \mathbf{x}_i$, $d(\mathbf{x}_i, \mu_{\ell_i}) \leq d(\mathbf{x}_i, \mu_h), \forall h$. Such assignments will be called *greedy* assignments. However, this need not satisfy the balancing constraint. So, we do the assignment of all the points as follows:

(1) First, exactly $[m - n_h]_+$ points are assigned to cluster $h, \forall h$, such that for each $\mathbf{x}_i$ that has been assigned to a cluster,

either it has been assigned to its nearest cluster, i.e.,

$$d(\mathbf{x}_i, \mu_{\ell_i}) \leq d(\mathbf{x}_i, \mu_h), \quad \forall h \ ,$$

or all clusters h' whose representatives $\mu_{h'}$ are nearer to $\mathbf{x}_i$ than its own representative μ_{ℓ_i} already have the required number of points $[m - n_{h'}]_+$, all of which are nearer to $\mu_{h'}$ than $\mathbf{x}_i$, i.e.,

$$\forall h' \text{ such that } d(\mathbf{x}_i, \mu_{\ell_i}) > d(\mathbf{x}_i, \mu_{h'}), \qquad d(\mathbf{x}_i, \mu_{h'}) \geq d(\mathbf{x}', \mu_{h'}), \quad \forall \mathbf{x}' \in S_{h'} \ .$$

(2) The remaining points are greedily assigned to their nearest clusters.

Condition (1) is motivated by the stable marriage problem that tries to get a stable match of n men and n women, each with his/her preference list for marriage over the other set [22]. The populate step can be viewed as a generalization of the standard stable marriage setting in that there are k clusters that want to "get married", and cluster h wants to "marry" at least $[m - n_h]_+$ points. Hence, an assignment of points to clusters that satisfies condition (1) will be called a *stable* assignment and the resulting clustering is called *stable*.

5.1.2 Part 2: Refine

In the second part, feasible iterative refinements are done starting from the clustering obtained from the first part until convergence. Note that at this stage, each point $\mathbf{x}_i \in \mathcal{X}$ is in one of the clusters and the balancing constraints are satisfied, i.e., $|S_h| \geq m, \forall h$. There are two ways in which a refinement can be done, and we iterate between these two steps and the updation of the cluster representative:

(1) If a point $\mathbf{x}_i$ can be moved, without violating the balancing constraint, to a cluster whose representative is nearer than its current representative, then the point can be safely re-assigned. First, all such possible individual re-assignments are done.

(2) Once all possible individual re-assignments are done, there may still be groups of points in different clusters that can be simultaneously re-assigned to reduce the cost without violating the constraints. In this stage, all such possible group re-assignments are done.

After all the individual and group reassignments are made, the cluster representatives are re-estimated. Using the re-estimated means, a new set of re-assignments are possible and the above two steps are performed again. The process is repeated until no further updates can be done, and the refinement algorithm terminates.

5.2 Details of Part 1: Populate

In this subsection, we discuss `poly-stable`, an algorithm to make stable assignment of $[m - n_h]_+$ points per cluster. The algorithm, which is a generalization of the celebrated stable marriage problem, is presented in detail in Algorithm 1. First the distance $d(\mathbf{x}, \mu_h)$ between every unassigned point $\mathbf{x} \in \mathcal{X}_u$ and every cluster representative is computed. For each cluster S_h, all the non-sampled $(N-n)$ points are sorted in increasing order of their distance with $\mu_h, h = 1, \ldots, k$. Let the ordered list of points for cluster S_h be denoted by Π_h. Let m_h denote the number of points yet to be assigned to cluster S_h at any stage of the algorithm to satisfy the constraints. Initially $m_h = [m - n_h]_+$. The algorithm terminates when $m_h = 0, \forall h$.

The basic idea behind the algorithm is "cluster proposes, point disposes". In every iteration, each cluster S_h proposes to its nearest m_h points. Every point that has been proposed, gets temporarily assigned to the nearest among its proposing clusters and rejects any other proposals. Note that if a point has received only one proposal, it gets temporarily assigned to the only cluster which proposed to it and there is no rejection involved. For each cluster, m_h is recomputed. If $m_h \neq 0$, cluster S_h proposes the next m_h points from its sorted list Π_h that have not already rejected its proposal. Each of the proposed points accepts the proposal either if it is currently unassigned or if the proposing cluster is nearer than the cluster $S_{h'}$ to which it is currently assigned. In the

later case, the point rejects its old cluster $S_{h'}$ and the cluster $S_{h'}$ loses a point so that $m_{h'}$ goes up by 1. This process is repeated until $m_h = 0, \forall h$ and the algorithm terminates.

Algorithm 1 `poly-stable`

Input: The existing clusters $\{S_h\}_{h=1}^k$, the cluster representatives $M = \{\mu_h\}_{h=1}^k$, the required minimum cluster size m, and the set of unassigned data points $\mathcal{X}_u$.
Output: A disjoint stable k-partitioning $\{S_h\}_{h=1}^k$ of $\mathcal{X}$ such that $|S_h| \geq m, \forall h$

Method:

```
Mark all x ∈ X_u as free
for h = 1 to k do
    Π_h ← Sort X_u in increasing order according to d(x, μ_h), x ∈ X_u, μ_h ∈ M
    m_h ← [m − |S_h|]_+        {number of points to be assigned}
    ι_h ← 1                    {index in the sorted list Π_h}
end for
λ ← Σ_{h=1}^k m_h
while λ > 0 do
    for h = 1 to k do
        if m_h > 0 then
            m*_h ← m_h
            { Cluster C_h proposes to the next m_h points Π_h(i), i = ι_h, ··· , (ι_h + m_h − 1) }
            for i = ι_h to (ι_h + m_h − 1) do
                if Π_h(i) is free then
                    S_h ← S_h ∪ {Π_h(i)}
                    Mark Π_h(i) ∈ X^(u) as engaged to h
                    m_h ← m_h − 1
                else if Π_h(i) is engaged to h' but d(v, μ_h) < d(v, μ_h') then
                    S_h ← S_h ∪ {Π_h(i)}, S_h' ← S_h' \ {Π_h(i)}
                    Mark Π_h(i) ∈ X_u as engaged to h
                    m_h ← m_h − 1, m_h' ← m_h' + 1
                end if
            end for
            ι_h ← ι_h + m*_h
        end if
    end for
    λ ← Σ_{h=1}^k m_h
end while
```

Now we show that this algorithm indeed satisfies all the required conditions and hence ends up in a stable assignment. Before giving the actual proof, we take a closer look at what exactly happens when an assignment is unstable. Let $(\mathbf{x}_i \rightarrow S_h)$ denote the fact that the point $\mathbf{x}_i$ has been assigned to the cluster S_h. An assignment is unstable if there exist at least two assignments $(\mathbf{x}_{i_1} \rightarrow S_{h_1})$ and $(\mathbf{x}_{i_2} \rightarrow S_{h_2})$, $h_1 \neq h_2$, such that $\mathbf{x}_{i_1}$ is nearer to μ_{h_2} than its own cluster representative μ_{h_1} and μ_{h_2} is nearer to $\mathbf{x}_{i_1}$ than $\mathbf{x}_{i_2}$. The point-cluster pair $(\mathbf{x}_{i_1}, S_{h_2})$ is said to be dissatisfied under such an assignment. An assignment in which there are no dissatisfied point-cluster pairs is a stable assignment and satisfies the constraints. Next, we show that there will no dissatisfied point-cluster

pair after the algorithm terminates.

Lemma 3 `poly-stable` *gives a stable assignment.*

Proof: If possible, let `poly-stable` give an unstable assignment. Then there must exist two assignments $(\mathbf{x}_{i_1} \rightarrow S_{h_1})$ and $(\mathbf{x}_{i_2} \rightarrow S_{h_2})$ such that $(\mathbf{x}_{i_1}, S_{h_2})$ is a dissatisfied point-cluster pair. Then, since $\mathbf{x}_{i_1}$ is nearer to S_{h_2} than $\mathbf{x}_{i_2}$, S_{h_2} must have proposed to $\mathbf{x}_{i_1}$ before $\mathbf{x}_{i_2}$. Since $\mathbf{x}_{i_1}$ is not assigned to S_{h_2}, $\mathbf{x}_{i_1}$ must have either rejected the proposal of S_{h_2} meaning it was currently assigned to a cluster which was nearer than S_{h_2}, or accepted the proposal initially only to reject it later meaning it got a proposal from a cluster nearer than S_{h_2}. Since a point only improves its assignments over time, the cluster S_{h_1} to which $\mathbf{x}_{i_1}$ is finally assigned must be nearer to it than the cluster S_{h_2} which it rejected. Hence $\mathbf{x}_{i_1}$ cannot be dissatisfied which contradicts the initial assumption. So, the final assignment is indeed stable. ∎

Next we look at the total number of proposals that are made before the algorithm terminates. After a cluster proposes to a point for the first time, there are three possible ways this particular point-cluster pair can behave during the rest of the algorithm: (i) the point immediately rejects the proposal in which case the cluster never proposes to it again; (ii) the point accepts it temporarily and rejects it later — the cluster obviously does not propose to it during the acceptance period and never proposes to it after the rejection; (iii) the point accepts the proposal and stays in that cluster till the algorithm terminates — obviously the cluster does not propose to it again during this time. Hence, each cluster proposes each point at most once and so the maximum number of proposals possible is $k \times (bN - n)$.

We take a look at the complexity of the proposed scheme. Following the above discussion, the complexity from the proposal part is $O(k(bN - n))$. Before starting the proposals, the sorted lists of distances of all the points from each of the clusters have to be computed, which has a complexity of $O(k(N - n)\log(N - n))$. And after `poly-stable` terminates, the remaining $N(1 - b)$ points are greedily assigned to their nearest clusters. Note that the greedy assignments at this stage does not hamper the feasibility or the stability of the resulting clustering. So, the total complexity of the first part of the proposed scheme is $O(k(N-n)(\log(N-n)+k(bN-n)+N(1-b)) = O(kN \log N)$.

After all the points are assigned to clusters using `poly-stable`, the cluster representatives $\{\mu_h\}_{h=1}^k$ are updated such that for $h = 1, \ldots, k$,

$$\mu_h = \operatorname*{argmin}_{\mu} \sum_{\mathbf{x} \in S_h} d(\mathbf{x}, \mu) \ .$$

We assume that there is an efficient way to get the representatives. Note that each new representative will be at least as good as the old representatives in terms of the cost function. Hence, optimal updation of the cluster representatives results in a decrease in the cost function value.

5.3 Details of Part 2: Refine

In this subsection, we discuss `refine`, an algorithm to do iterative refinements in order to get a better solution to the clustering problem while satisfying the constraints all along. Note that this part of the scheme applies to all points in $\mathcal{X}$, and does not differentiate between a sampled or non-sampled point. Further, the input to this part is an existing disjoint partitioning $\{S_h\}_{h=1}^k$ of $\mathcal{X}$ such that $|S_h| = n_h \geq m, \forall h$. As outlined earlier, `refine` has two parts in every iteration: the first part involves performing *individual re-assignments*, where individual points are re-assigned

with a guaranteed decrease in the objective while satisfying constraints; the second part involves performing *group reassignments* with the same guarantees.

5.3.1 Individual Re-assignments

The individual re-assignment (IR) runs over all the points $\mathbf{x} \in \mathcal{X}$. If, for a point $\mathbf{x}_i$ in cluster S_{ℓ_i},

(a) there is a cluster h whose representative $\boldsymbol{\mu}_h$ is nearer to $\mathbf{x}_i$ than its current representative $\boldsymbol{\mu}_{\ell_i}$, i.e.,

$$\exists h \neq \ell_i, \qquad d(\mathbf{x}_i, \boldsymbol{\mu}_{\ell_i}) > d(\mathbf{x}_i, \boldsymbol{\mu}_h) \ ,$$

(b) cluster S_{ℓ_i} can afford to let go a point without violating the constraints, i.e., $n_{\ell_i} > m$,

then $\mathbf{x}_i$ can be safely reassigned to (any such) cluster h. Clearly, the overall objective function decreases without violating any constraints. To obtain the maximum decrease, $\mathbf{x}_i$ should be assigned to its nearest cluster.

After a pass through the entire data, for any point $\mathbf{x}_i$

either it is in its nearest cluster, and hence there is no reason to re-assign it,

or it is not in its nearest cluster, but it cannot be re-assigned since that will violate the balancing constraints for the cluster it is currently in.

For the first set of points, no updates are necessary. For the second set, we investigate if group re-assignments can improve the objective while satisfying constraints.

5.3.2 Group Re-assignments

For every point $\mathbf{x} \in \mathcal{X}$, let $H^{(\mathbf{x})}$ be the set of clusters h whose cluster representatives μ_h are nearer to $\mathbf{x}$ than its current cluster representative $\boldsymbol{\mu}_{\ell_i}$, i.e.,

$$H^{(\mathbf{x})} = \{h | d(\mathbf{x}, \mu_{\ell_i}) > d(\mathbf{x}, \mu_h)\} \ ,$$

Clearly, for points already in their nearest cluster, $H^{(\mathbf{x})}$ is the null set. Using the sets $H^{(\mathbf{x})}$, $\forall \mathbf{x} \in \mathcal{X}$, we construct a k-vertex potential assignment graph $G = (V, E)$ with $V = \{h\}_{h=1}^{k}$, one vertex corresponding to every cluster, such that for every point $\mathbf{x}_i \in \mathcal{X}$, there is a directed edge of unit capacity from vertex ℓ_i to all vertices $h \in H^{(\mathbf{x}_i)}$. The total capacity on every directed edge is consolidated to get a single integer value, and edges with zero capacity are considered non-existent. Note that a cycle in this integer weighted directed graph, suggests a set of group re-assignments of points that is guaranteed to decrease the clustering objective function. In order to do group re-assignments, the algorithm finds the strongly connected components of G and keeps removing weighted cycles from each component till there are no cycles in the graph.

The final step is to re-estimate the representatives of the individual clusters so that the clustering objective function is further decreased. Note that the re-estimation of the representatives should be followed by individual and group re-assignments, and the process is to be repeated till convergence. The details of the algorithm `refine` are presented in Algorithm 2.

Algorithm 2 `refine`

Input: A disjoint stable k-partitioning $\{S_h\}_{h=1}^k$ of $\mathcal{X}$ such that $|S_h| \geq m$; a set of cluster representatives μ_h

Output: A disjoint k-partitioning $\{S_h^*\}_{h=1}^k$ of $\mathcal{X}$ and a set of representatives $\{\boldsymbol{\mu}_h^*\}_{h=1}^k$ such that $|S_h^*| \geq m$, $L(\{\mu_h^*, S_h^*\}_{h=1}^k) \leq L(\{\mu_h, S_h\}_{h=1}^k)$ and $\{\mu_h^*, S_h^*\}_{h=1}^k$ is a local minimum of $\mathbb{L}(\cdot)$.

Method:

repeat
  $\theta \leftarrow 0$
  {Update the cluster means}
  for $h = 1$ to k **do**
    $\mu_h \leftarrow \frac{1}{|S_h|} \sum_{x \in S_h} x$
  end for
  {Reassign points}
  for all $\mathbf{x}_i \in \mathcal{X}$ **do**
    $H^{(\mathbf{x}_i)} \leftarrow \emptyset$
    {Individual Reassignments}
    for $h = 1$ to k **do**
      if $d(x, \mu_h) < d(x, \mu_{h^*})$ **then**
        if $|S_{\ell_i}| > m$ **then**
          $S_h \leftarrow S_h \cup \{x\},\ S_{\ell_i} \leftarrow S_{\ell_i} \setminus \{x\},\ \ell_i \leftarrow h,\ \theta \leftarrow \theta + 1$
        else
          $H^{(\mathbf{x}_i)} \leftarrow H^{(\mathbf{x}_i)} \cup \{h\}$
        end if
      end if
    end for
  end for
  {Group Reassignments}
  Construct potential assignment graph $G = (V, E)$ from $\{H^{(\mathbf{x})}\}_{\mathbf{x} \in \mathcal{X}}$
  $C \leftarrow$ `strongly-connected-components`(G)
  for all $\gamma \in C$ **do**
    while there exists *back-edge* in `DFS`(γ) **do**
      $\lambda \leftarrow$ the cycle corresponding to this back-edge
      $\theta \leftarrow \theta +$ minimum edge-weight in λ
      $\lambda_\theta \leftarrow \lambda$ with all edge-weights set to θ
      reassign points according to directed cycle λ_θ
      $\gamma \leftarrow \gamma \setminus \lambda_\theta$
    end while
  end for
until $\theta = 0$
$\{S_h^*\}_{h=1}^k \leftarrow \{S_h\}_{h=1}^k$

6 Experimental Results

In this section, we first briefly discuss the complexity of the proposed framework. Then, we present empirical results on several high-dimensional text datasets using different performance evaluation criterion for comparison.

First, we show that the complexity of the proposed framework is quite reasonable. The sampling of the n points out of N uniformly at random has a complexity of $O(N)$. If the clustering algorithm on the sampled data is a variant of `KMeans`, the computational complexity is $O(t_0 kn)$ where t_0 is the number of iterations. As shown earlier, the `poly-stable` algorithm has a complexity of $O(kN \log N)$. The individual re-assignments of the refinement step is $O(N)$, whereas the group re-assignments take $O(kN + t_1 k^2)$ where t_1 is the number of strongly connected components in that iteration. If the refinement is performed for t_2 iterations, then the complexity of the step is $O(t_2 N + t_2 kN + t_1 t_2 k^2) = O(kN)$. So, the total complexity of the scheme is $O(kN \log N)$, assuming t_0, t_1, t_2 are constants. Note that this is better than the $\Omega(N^2)$ complexity of graph-based algorithms that can provide balancing. It is worse than the $O(kN)$ complexity (assuming constant number of iterations) of the iterative greedy relocation algorithms such as `KMeans`, but such algorithms are not guaranteed to satisfy the balancing constraints.

6.1 Datasets

The datasets that we used for empirical validation and comparison of our algorithms were carefully selected to represent some typical clustering problems. We also created various subsets of some of the datasets for gaining greater insight into the nature of clusters discovered or to model some particular clustering scenario (e.g. balanced clusters, skewed clusters, overlapping clusters etc.).

- classic3: Classic3 is a well known collection of documents used for text analysis. It is an easy dataset to cluster since it contains documents from three well-separated sources. Moreover, the intrinsic clusters are largely balanced. The corpus contains 3893 documents, among which 1400 CRANFIELD documents are from aeronautical system papers, 1033 MEDLINE documents are from medical journals, and 1460 CISI documents are from information retrieval papers. The particular vector space model used had a total of 4666 features (words). Thus, each document is represented as a vector in a 4666-dimensional space. The dataset serves as a sanity check for the algorithms.

- news20: The CMU Newsgroup dataset, hereafter called news20, is a widely used compilation of documents. We tested our algorithms on not only the original dataset, but on a variety of subsets with differing characteristics to explore and understand the behavior of our algorithms. The standard dataset is a collection of 19,997 messages, gathered from 20 different USENET newsgroups. One thousand messages are drawn from the first 19 newsgroups, and 997 from the twentieth. The headers for each of the messages are then removed to avoid biasing the results. The particular vector space model used had 25,924 words. The dataset embodies the features characteristic of a typical text dataset—high-dimensionality, sparsity and some significantly overlapping clusters.

- small-news20: We formed this subset of news20 by sampling 2000 messages from original dataset. We sampled 100 messages from each category in the original dataset. Hence the dataset has balanced clusters (though there may be overlap). The dimensionality of the data

was 13,406. Since the dimensionality of the data is much smaller than the number of data points, this is a harder dataset to cluster.

- similar-1000: We formed another subset of news20 by choosing 3 somewhat similar newsgroups: talk.politics.guns, talk.politics.mideast, talk.politics.misc. The dataset has a total of 3000 documents, with 1000 documents per newsgroup, and a dimensionality of 10,083. The dataset has a fair amount of overlaps in high dimensions and is hence a difficult dataset to cluster.

- yahoo: The Yahoo20 dataset (K-series) is a collection of 2340 Yahoo news articles belonging one of 20 different Yahoo categories, and has a dimensionality of 24273. The salient feature of this dataset is that the natural clusters are not at all balanced, with cluster sizes ranging from 9 to 494. This is where it significantly differs from the previous datasets. This dataset helps in studying the effect of forcing balanced clustering in a naturally unbalanced data.

- nsf-top30: The NSF dataset contains abstracts of all NSF proposals that were granted funding over a period of 14 years from 1990-2003. The nsf-top30 is a mildly pruned version of the data that contains the top 30 categories, in terms of number of grants, that were funded. The data consists of 128,111 abstracts from 30 categories with sizes ranging from 9911 to 1447. In fact, all categories with more than 1000 grants were selected. The data has a dimensionality of 45,998. This is a rather large and quite sparse dataset since abstracts are typically much smaller than normal documents.

6.2 Algorithms

We compared 6 different algorithms on every dataset.

- `KMeans` is the widely used and popular iterative relocation clustering algorithm. There is one important modification we make in the way the algorithm was run — we ran the data on the L_2 normalized version of the data. There are two reasons for this: (i) all algorithms run on the same representation of the data, and (ii) it has been well established that applying kmeans on the original sparse high-dimensional data gives poor results [11].

- `SPKMeans` is the spherical kmeans algorithm [11] outlined in section 4. Motivated by research in information retrieval, the algorithm uses cosine similarity between data points and cluster representatives and has been shown to give good results on several benchmark text datasets [11, 3].

  Note that both the above algorithms do not have any way to ensure balanced clustering. The next algorithm uses all the components of the proposed framework, and hence guarantees balanced clusterings with a scalable algorithm.

- `SPKpr` uses `SPKMeans` as the base clustering algorithm and uses both the populate (`p`) and refine (`r`) steps as outlined in section 5. This algorithm is guaranteed to satisfy any balancing constraints given as input, although the quality of clusters may deteriorate under stringent conditions.

  We also perform lesion studies using three other algorithms, in which one or more components of the proposed framework are missing.

- `SPKpnr` uses `SPKMeans` as the base clustering algorithm. It uses the populate (`p`) step, but does not refine (`nr`) the resulting clusters. The algorithm satisfies any given balancing constraints but need not give good results since the feasible solution is not refined.

- `SPKgpnr` also uses `SPKMeans` for clustering. It uses a greedy populate (`gp`) scheme where every point is assigned to the nearest cluster. Further, no refinements (`nr`) are done after the greedy populate step. Clearly, this algorithm is not guaranteed to satisfy balancing constraints.

- `SPKgpr` uses `SPKmeans` as the base clustering algorithm. It uses greedy populate (`gp`) to put points into clusters, but performs a full refinement (`r`) after that. The algorithm is not guaranteed to satisfy the balancing constraints since the populate step is greedy and the refinements do not start from a feasible clustering.

In a tabular form, the four algorithms can be presented as follows:

	No Refine	Refine	Balancing
Greedy Populate	`SPKgpnr`	`SPKgpr`	No Guarantee
Populate	`SPKpnr`	`SPKpr`	Guaranteed

6.3 Methodology

Performance of the algorithms are evaluated using one measure for the quality of clustering and two measures for the balancing of cluster sizes. The measure for cluster quality evaluates how the clustering agrees with the hidden true labels of the data. The measures for cluster sizes evaluate how well balanced the cluster sizes are in the average as well as worst case.

Quality of clustering is evaluated using the following measure:

- Normalized Mutual Information (NMI) is used to measure the agreement of the assigned cluster labels and the true class labels from the confusion matrix of the assignments. Mutual information (MI) gives the amount of statistical similarity between the cluster and class labels [9]. If X is a random variable for the cluster assignments and Y is a random variable for the true labels on the same data, then their MI is given by $I(X;Y) = E[\ln \frac{p(X,Y)}{p(X)p(Y)}]$ where the expectation is computed over the joint distribution of (X, Y) estimated from the particular clustering of the dataset under consideration. In order to get a number in $[0, 1]$, we normalize the MI to get $NMI(X;Y) = I(X;Y)/\sqrt{H(X)H(Y)}$, where $H(X), H(Y)$ are the entropies of the marginal distributions of X, Y respectively.

Quality of balancing is evaluated using the following measures:

- Standard Deviation in cluster sizes is the first and perhaps most obvious way to evaluate balancing. For algorithms that are guaranteed to give balanced clusters, the standard deviation becomes smaller as the balancing fraction is increased.

- The ratio between the minimum to average cluster sizes is our second measure of evaluation. For N data points put into k clusters, the average cluster size is just N/k. For a given clustering, we compare the size of the smallest cluster to this average.

All results reported herein have been averaged over 10 runs. All algorithms were started with the same random initialization to ensure fairness of comparison. Each run was started with a

different random initialization. However, no algorithm was restarted within a given run and all of them were allowed to run to completion. Since the standard deviations over 10 runs were reasonably small, to reduce clutter, we have chosen to omit a display of error bars in our plots.

We report experimental results for particular choices of datasets, algorithms, number of samples and balancing fraction in the next section. Note that there may be a mismatch in the choice of number of samples and the balancing fraction. For example, consider a 3-clustering task with 10,000 points. Let the number of samples be 6,000 and the balancing fraction be 0.8. If the preliminary 3-clustering of 6,000 points puts all points in one cluster with two other empty clusters, then `populate` cannot ensure balancing fraction of 0.8 using the remaining 4,000 points. As a result, `refine` has to start working on a clustering that does not already satisfy the balancing constraints. In practice, there are several ways to address such a situation. First, one can reallocate sets of points before or after `populate`, so that `refine` starts from a clustering that satisfies the balancing constraint. If formal guarantees of balancing are not critical, then one can simply run `refine` on the output of `populate` and not worry about this issue. Since smaller (than required by the balancing constraints) clusters can only get new points in `refine`, it is reasonable to expect that the final solution will almost, if not exactly, satisfy the balancing constraints. Since we want to guarantee balancing, we use yet another approach—that of restricting the number of samples you can draw. For the above example, if we sample at most 2000 points, then irrespective of what the preliminary cluster sizes were, it is always possible to satisfy the balancing constraints. In general, sampling a maximum of $\lfloor N(1 - b(1 - \frac{1}{k}))\rfloor$ points always leaves enough points to generate a balanced cluster with balancing fraction b. Since $\lfloor N(1-b)\rfloor \leq \lfloor N(1-b(1-\frac{1}{k}))\rfloor$, we draw at most $\lfloor N(1-b)\rfloor$ samples in the the reported experiments. Hence, for all the results reported, if s is the attempted number of samples and s_{true} is the actual number of samples, then $s_{\text{true}} = \lfloor \min(s, N(1-b))\rfloor$. Unless otherwise mentioned, all reported results are for $s = N/2$ so that $s_{\text{true}} = \lfloor \min(N/2, (1-b)N)\rfloor$. For the example above, since $b = 0.8$, we draw $s_{\text{true}} = \lfloor \min(N/2, N/5)\rfloor = \lfloor N/5\rfloor = 2000$ samples.

6.4 Results

The results on classic3 are shown in figure 1. From the results in 1(a), we see that most algorithms perform very similarly under very weak balancing requirements. `KMeans` performs poorly compared to all the other algorithms that are well suited for high-dimensional data. Also, `SPKgpnr` achieves lower NMI values compared to the other `SPKMeans` based algorithms since it uses greedy populate and does not refine the resulting clusters. In 1(b), since the balancing fraction is 0.9, the algorithms that satisfy the balancing constraints, viz `SPKpr` and `SPKpnr`, achieve lower NMI than those that do not satisfy the constraints. In particular, `SPKgpr` performs better than its corresponding balanced algorithm `SPKpr`, and `SPKgpnr` performs better than its corresponding `SPKpnr`. This decrease in performance is due to the fact that the balancing algorithms are satisfying the balancing constraints. In fact, the value of the refinement stage becomes clear as `SPKpr`, that guarantees balancing, performs better than `SPKgpnr`, that is unconstrained but does use the refinement phase. Figure 1(c) shows the variation in NMI across a range of balancing fractions, with the results of `KMeans` and `SPKMeans` shown as points, since their performance does not depend on the balancing fraction. Again, the algorithms respecting balancing constraints perform better over the entire range. Fig 1(d) shows the change in standard deviation in cluster sizes as the number of clusters changes. The standard deviation goes down for all the algorithms, although it is marginally more pronounced for the balancing algorithms. Figures 1(e) and (f) show the minimum-to-average ratio of cluster sizes with varying number of clusters. While there is not much difference in figure 1(e)

due to the weak balancing requirement, figure 1(f) clearly shows how SPKpr and SPKpnr respect the balancing requirement while the fraction gets small for the other algorithms.

Figure 2 shows the results on news-20. This is a typical high-dimensional text clustering problem and the true clusters are balanced. As shown in figure 2(a), the balanced algorithms SPKpr and SPKpnr perform as good as SPKMeans, whereas the unconstrained algorithms SPKgpr and SPKgpnr do not perform as well. Clearly, the balancing constraints resulted in better results. As before, KMeans does not perform as well as the other algorithms. Under a stricter balancing requirement in figure 2(b), as before, SPKgpr performs marginally better than SPKpr, but the latter satisfies the balancing constraints. The same behavior is observed for SPKgpnr and its corresponding SPKpnr. Note that among the two balancing algorithms, SPKpr performs much better than SPKpnr, thereby showing the value of the refinement step. The same is observed for the unbalanced algorithms as well. Figure 2(c) shows the variation in NMI across balancing constraints for the right number of clusters. We note that the refined algorithms perform much better, although the constraints do decrease the performance by a little amount. Interestingly, both KMeans and SPKmeans achieve very low minimum balancing fraction. Figure 2(d) shows the standard deviation in cluster sizes. The balancing algorithms achieve the lowest standard deviations, as expected. Figures 2(e) and 2(f) show the minimum-to-average ratio of cluster sizes. Clearly, the balancing algorithms respect the constraints whereas the ratio gets really low for the other algorithms. For a large number of clusters, almost all the unconstrained algorithms start giving zero-sized clusters.

The overall behavior of the algorithms on small-news-20 (figure 3) and similar-1000 (figure 4) are more or less the same as that described above for news-20 and classic3.

Figure 5 shows the results on yahoo. This is a very different dataset from the previous datasets since the natural clusters are highly unbalanced with cluster sizes ranging from 9 to 494. The comparison on most measures of performance look similar to that of other datasets. The major difference is in the minimum-to-average ratio shown in figures 5(e) and (f). As expected, the balanced algorithms SPKpr and SPKpnr respect the constraints. The other algorithms (except KMeans) start getting zero-sized clusters for quite low values of clusters. Also, as the balancing requirement becomes more strict (as in figure 5(f)), the disparity between the balanced and other algorithms become more pronounced. Surprisingly, even for such an unbalanced data, the balanced algorithms, particularly SPKpr, perform almost as good as the unconstrained algorithms (figures 5(c)).

Figure 6 shows the results on nsf-top30. Although this is a rather large dataset with 128,111 points and 30 natural clusters, the performance of the scaled algorithms are quite competitive. In fact, as shown in figure 6(a), at a balancing fraction of 0.3, the refined algorithms SPKpr and SPKgpr often perform better than the base algorithm SPKMeans. At a higher balancing fraction of 0.9, the NMI values for the algorithms SPKpr and SPKpnr guaranteed to give balancing decreases (figure 6(b)). Even under such a strict balancing requirement, which is violated by the natural clusters themselves, SPKpr still performs quite well. At the correct number of clusters, the performance of the algorithms, as shown in figure 6(c), are comparable over a range of balancing fractions. Interestingly, the balancing fractions achieved by KMeans and SPKMeans are really low as seen in figure 6(c). In fact, SPKMeans has a balancing fraction of 0, meaning it generated at least one empty cluster in all 10 runs. Figure 6(d) shows the variation in standard deviation in cluster sizes. As expected, the algorithms guaranteed to generate balanced clusters show a lower standard deviation. In figures 6(e) and (f), we present the minimum-to-average ratio of cluster sizes over a range of number of clusters. Other than SPKpr and SPKpnr which are guaranteed to meet the balancing requirements, all other algorithms generate empty clusters even for moderate number of

Number of Clusters	Number of Samples							
	1000	5000	10000	20000	32027	64055	96083	128111
10	92.09	92.49	94.77	97.21	95.81	98.21	100.36	104.65
20	107.98	116.31	117.39	118.51	118.71	119.19	121.43	125.39
30	123.12	138.94	143.75	138.15	141.89	148.69	149.06	153.41

Table 2: Running times (in seconds) for SPKpnr on nsf-top30, averaged over 10 runs.

Number of Clusters	Number of Samples					
	1000	5000	10000	50000	100000	1000000
10	130.80	136.80	143.53	164.76	172.63	253.55
20	252.35	261.89	272.21	306.80	322.02	453.98
30	373.19	386.20	400.02	446.89	468.02	647.79

Table 3: Running times (in seconds) for SPKpnr on Art1M, averaged over 10 runs.

clusters. Also, the low balancing fractions of KMeans and SPKMeans, as observed in figure 6(c), is better explained by these figures.

The above results show that in terms of cluster quality, the proposed methods achieve clustering results that are as good as applying the basic clustering method on the entire dataset. In addition, the methods ensure that the clustering has guaranteed balancing properties. We now take a look at how well the framework scales with increase in the number of samples used. In Table 2, we report actual running times (in seconds) for SPKpnr on nsf-top30 for varying number of samples. The run times were observed on a 2.4 GHz P4 linux machine with 512 MB of RAM.

A few things need to be noted regarding the running times in Table 2. First, although there are speed-ups due to sampling, the improvements are not as stark as one would expect, especially when less than 10% of the samples are used. There are two main reasons for this. First, the base clustering algorithm SPKMeans is quite fast and takes about 2-3 minutes to cluster 128,111 documents. For slower base clustering algorithms such as agglomerative hierarchical clustering, the improvement would be more marked. Second, most of the time is taken to read the data from disk into the data structures needed for the algorithm, and we have no ready way to separate this time from the computational time. Also, we have not attempted to optimize the code, which could bring down the run-time further. It is also important to note that the running time need not monotonically increase with increasing number of samples. The actual running time (in seconds) depends on various factors such as the speed, load and memory availability on the machine, as well as the number of iterations needed by the algorithm to converge. However, there is a clear trend towards lower running times with less samples.

In order to study the scalability aspect for even larger datasets, we generated an artificial dataset Art1M of 1 million points sampled from a sparse multivariate Gaussian distribution.[2] The objective of the experiment was to study scalability, and not quality, for which we have already presented extensive evidence. The running times (averaged over 10 runs) of SPKpnr on Art1M is presented in Table 3. Note that the number of samples denote the size of the subset selected in step 1 of

[2]We used the sprandn function in Matlab to generate the artificial data.

Minimum acceptable number, s	10	20	30	40	50	60	70	80	90	100
Min. obtained/100 runs (news20)	13	41	67	90	120	150	178	205	243	273
Min. obtained/100 runs (nsf-top30)	21	49	77	113	143	170	200	234	276	301

Table 4: Comparison of the minimum acceptable number of samples per cluster with the smallest numbers obtained over 100 runs.

the proposed methodology, and the running time is the total time for the three steps for clustering of all the points. As before, sampling less points clearly results in smaller running times, and the improvement is more pronounced for larger number of clusters. Thus, the relative advantages of the proposed methodology becomes more clear for larger datasets, which is very desirable.

Lemma 2 gives a high probability guarantee that $csl \ln k$ samples are sufficient to get at least s samples from each cluster, for an appropriately chosen constant c. To compare empirically observed numbers with the guidelines provided by Lemma 2, we ran additional experiments on the two largest datasets, nsf-top30 and news20, with $k = 30$ and 20 respectively. For nsf-top30, there are 128111 documents and the smallest true cluster size is 1447 documents, giving $l = 89$. Since $c = 1$ satisfies the condition of the Lemma, we chose this simple setting for both data sets. We now sampled the dataset randomly according to our formula to get at least s samples per cluster for a range of values of s. In Table 4, we report results by taking the worst numbers over 100 runs, i.e., we draw $csl \ln k$ samples from the dataset 100 times, note the smallest representative size for every run and then report the smallest among these 100 numbers. As is clear from the table, $csl \ln k$ samples indeed seem sufficient as we always exceeded the desired minimum number of samples per cluster. At the same time, the numbers observed are of the same order as s, hence the Lemma is not overtly conservative. Similar results were observed for news20, with $l = 21, c = 1, k = 20$.

7 Conclusion

In this paper, we used ideas from the theory of recurrent events and renewals, generalizing a known result in the coupon collector's problem in route to Lemma 2 which essentially showed that a small sample was sufficient to obtain a core clustering. We then efficiently allocated the rest of the data points to the core clusters while adhering to balancing constraints, by using a scheme that generalizes the stable marriage problem. These two steps underlie the efficiency and generality of the proposed framework. Indeed, this framework for scaling up balanced clustering algorithms is fairly broad and is applicable to a very wide range of clustering algorithms, since the sampling as well as the populate and refine phases of the framework are practically independent of the clustering algorithm used for clustering the sampled set of points. Moreover, the guarantees provided by sampling, as stated in Lemma 2 and exemplified in Table 1 implies that the computational demands for the second step of the framework is not affected by the size of the dataset. Hence, it may be worthwhile to apply more sophisticated graph-based balanced partitioning algorithms in step 2 [25, 35]. Further, the populate and refine stages are quite efficient and provide the guarantee that each cluster will eventually have the pre-specified minimum number of points. As seen from the experimental results, the sampled balanced algorithms perform competitively, and often better, compared to a given base clustering algorithm that is run on the entire data.

It is often hard to choose the right clustering algorithm for a given problem. However, given

a base algorithm, the proposed framework can make an efficient scalable algorithm out of it, as long as the chosen algorithm belongs to a large class of parametric clustering algorithms. One possible concern for applying the method is that if the natural clusters are highly imbalanced, the framework may give poor results. The competitive results on the highly skewed yahoo dataset shows that this is not necessarily true. Further, one can always set the minimum number of points per cluster to a small value, or, generate a large number of clusters and then agglomerate, so that the balancing constraints do not hamper the quality of final clusters formed. Problems of uniform sampling from extremely skewed true clusters may be alleviated by applying more involved density-biased sampling techniques [32] during step 1, so that the assumptions and requirements on step 2 remain unchanged, i.e., the base clustering algorithm gets enough points per cluster. The proposed framework, with appropriate modifications, is also applicable to such settings where instead of an explicit cluster representative, the goodness of an assignment is measured by the objective function of the corresponding clustering. Computational and qualitative gains by applying the proposed framework on such sophisticated algorithms can be studied as a future work.

Acknowledgements: The research was supported in part by NSF grants IIS 0325116, IIS 0307792, and an IBM PhD fellowship.

References

[1] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3(3):277–290, 1990.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.

[3] A. Banerjee, I. Dhillon, J. Ghosh, and S. Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. *Journal of Machine Learning Research*, 6:1345–1382, 2005.

[4] A. Banerjee and J. Ghosh. Frequency sensitive competitive learning for balanced clustering on high-dimensional hyperspheres. *IEEE Transactions on Neural Networks*, 15(3):702–719, May 2004.

[5] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 2005. To Appear.

[6] K. P. Bennett, P. S. Bradley, and A. Demiriz. Constrained k-means clustering. Technical report, Microsoft Research, TR-2000-65, May 2000.

[7] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 9–15, 1998.

[8] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling EM (Expectation-Maximization) clustering to large databases. Technical report, Microsoft Research, 1998.

[9] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[10] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. 15th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 318–329, 1992.

[11] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.

[12] P. Domingos and G. Hulton. A general method for scaling up machine learning algorithms and its application to clustering. In *Proc. 18th Intl. Conf. Machine Learning*, pages 106–113, 2001.

[13] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, 2001.

[14] D. Fasulo. An analysis of recent work on clustering. Technical report, University of Washington, Seattle, 1999.

[15] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, 1967.

[16] Soheil Ghiasi, Ankur Srivastava, Xiaojian Yang, and Majid Sarrafzadeh. Optimal energy aware clustering in sensor networks. *Sensors*, 2:258–269, 2002.

[17] J. Ghosh. Scalable clustering methods for data mining. In Nong Ye, editor, *Handbook of Data Mining*, pages 247–277. Lawrence Erlbaum, 2003.

[18] Y. Guan, A. Ghorbani, and N. Belacel. Y-means: A clustering method for intrusion detection. In *Proc. CCECE-2003*, pages 1083–1086, May 2003.

[19] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *In Proc. ACM SIGMOD Intl. Conf.on Management of Data*, pages 73–84, New York, 1998. ACM.

[20] G. Gupta and M. Younis. Load-balanced clustering of wireless networks. In *Proc. IEEE Int'l Conf. on Communications*, volume 3, pages 1848–1852, May 2003.

[21] G. K. Gupta and J. Ghosh. Detecting seasonal and divergent trends and visualization for very high dimensional transactional data. In *Proc. 1st SIAM Intl. Conf. on Data Mining*, April 2001.

[22] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, 1989.

[23] J. Han, M. Kamber, and A. K. H. Tung. Spatial clustering methods in data mining: A survey. In *Geographic Data Mining and Knowledge Discovery*. Taylor and Francis, 2001.

[24] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[25] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[26] M. Kearns, Y. Mansour, and A. Ng. An information-theoretic analysis of hard and soft assignment methods for clustering. In *Proc. of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 282–293, 1997.

[27] P. J. Lynch and S. Horton. *Web Style Guide:Basic Design Principles for Creating Web Sites*. Yale Univ. Press, 2002.

[28] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Statist. Prob.*, volume 1, pages 281–297, 1967.

[29] D. Modha and S. Spangler. Feature weighting in k-means clustering. *Machine Learning*, 52(3):217–237, 2003.

[30] R. Motwani and P. Raghavan. *Randmized Algorithms*. Cambridge University Press, 1995.

[31] Neilson Marketing Research. *Category Management: Positioning Your Organization to Win*. McGraw-Hill, 1993.

[32] C. R. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. Technical report, Carnegie Mellon University, May 1999.

[33] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw Hill, 1991.

[34] Pramod Singh. *Personal communication at KDD05*. Aug, 2005.

[35] Alexander Strehl and Joydeep Ghosh. Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing*, 15(2):208–230, 2003.

[36] A. K. H. Tung, R. T. Ng, L. V. S. Laksmanan, and J. Han. Constraint-based clustering in large databses. In *Proc. Intl. Conf. on Database Theory (ICDT'01)*, Jan 2001.

[37] Y. Yang and B. Padmanabhan. Segmenting customer transactions using a pattern-based clustering approach. In *Proceedings of ICDM*, pages 411–419, Nov 2003.

[38] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *In Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 103–114, Montreal, 1996. ACM.

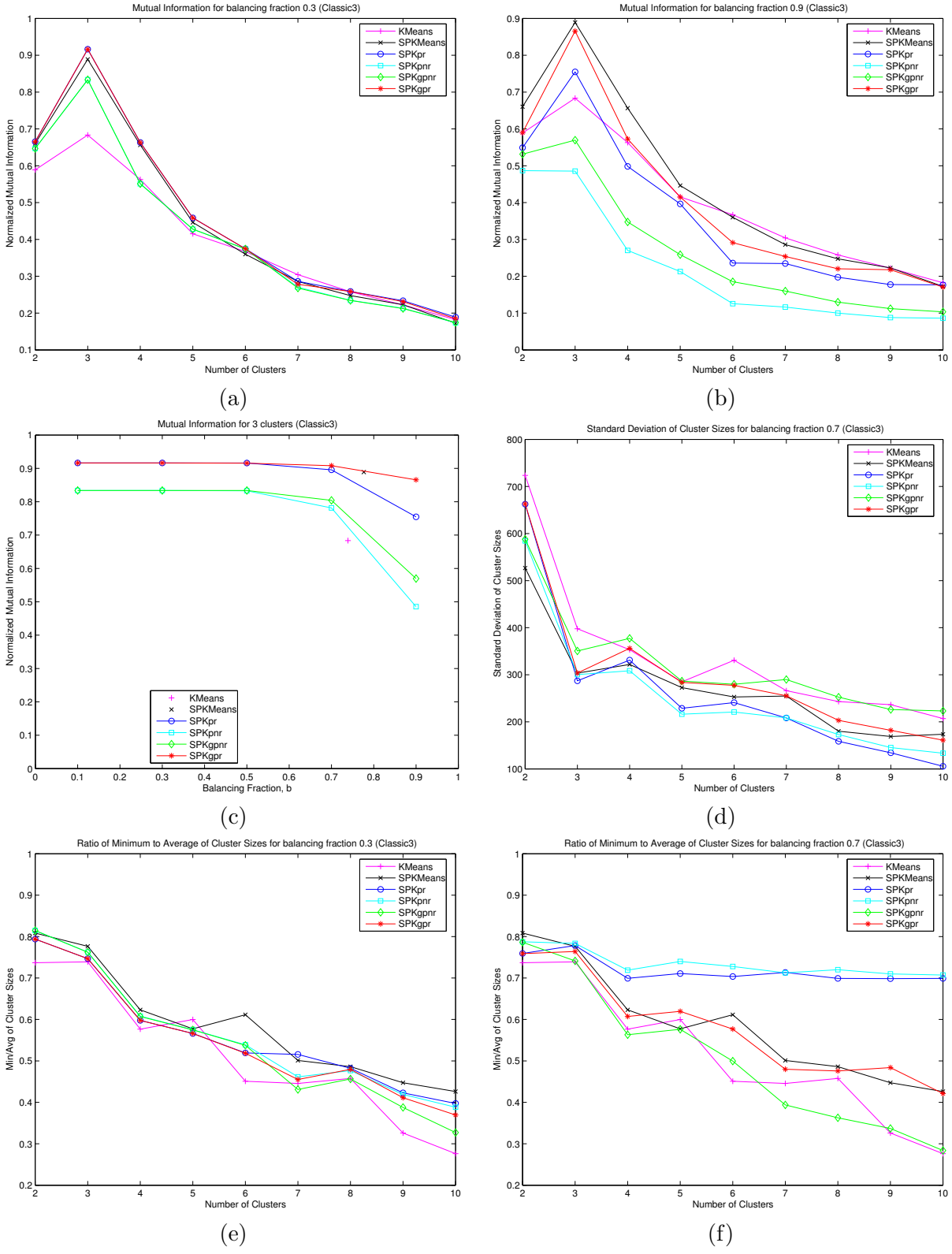

Figure 1: Results on classic3

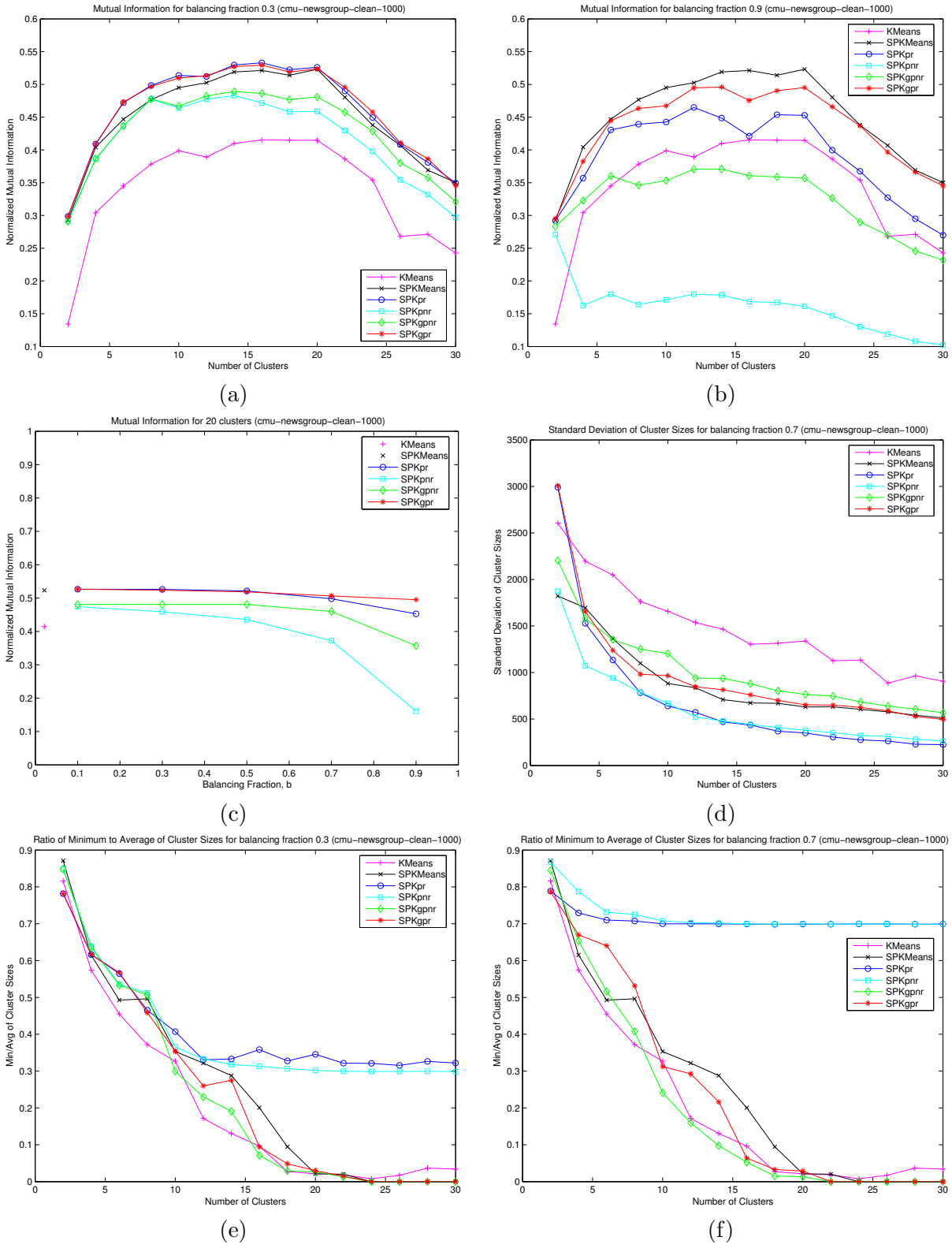

Figure 2: Results on news20

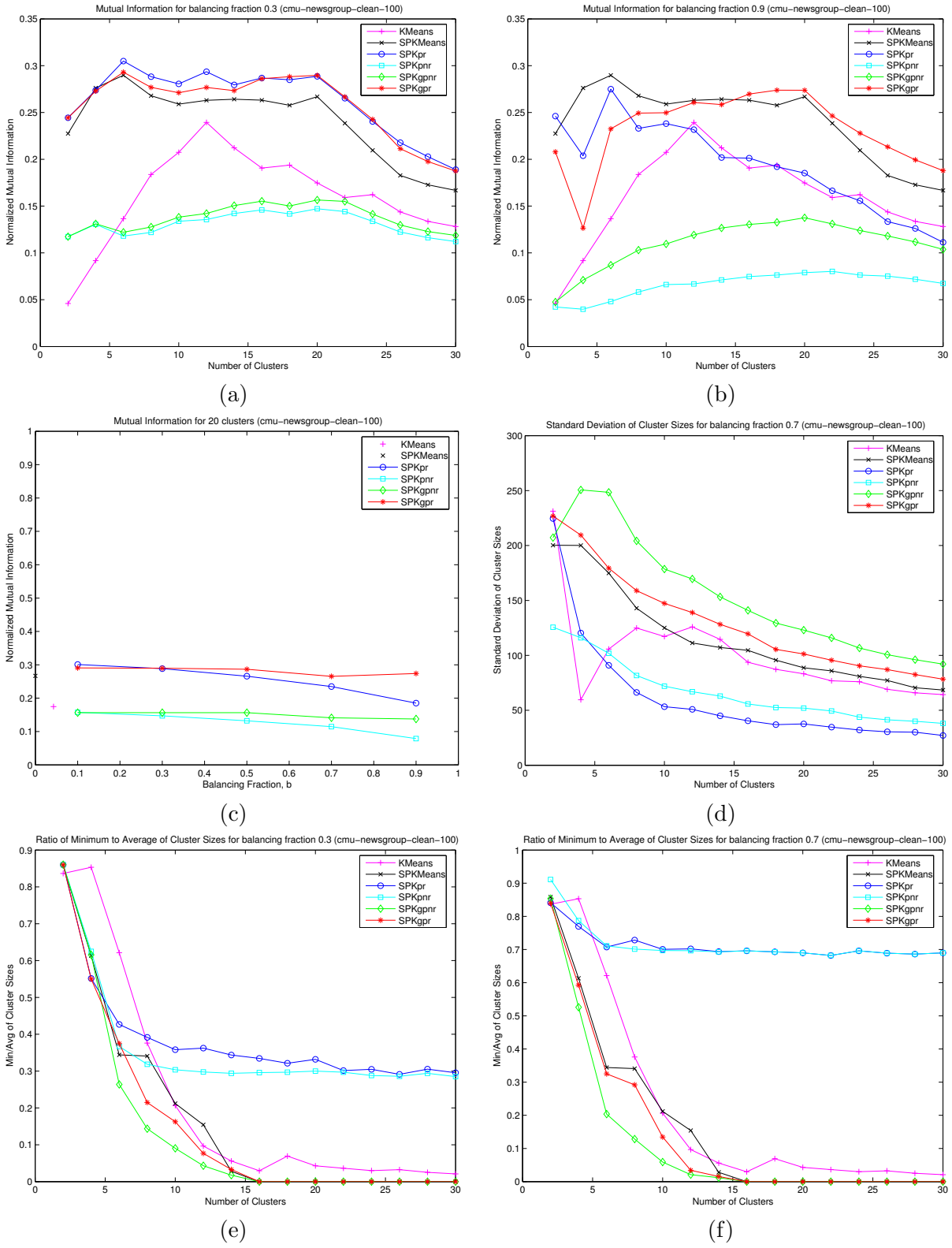

Figure 3: Results on small-news20

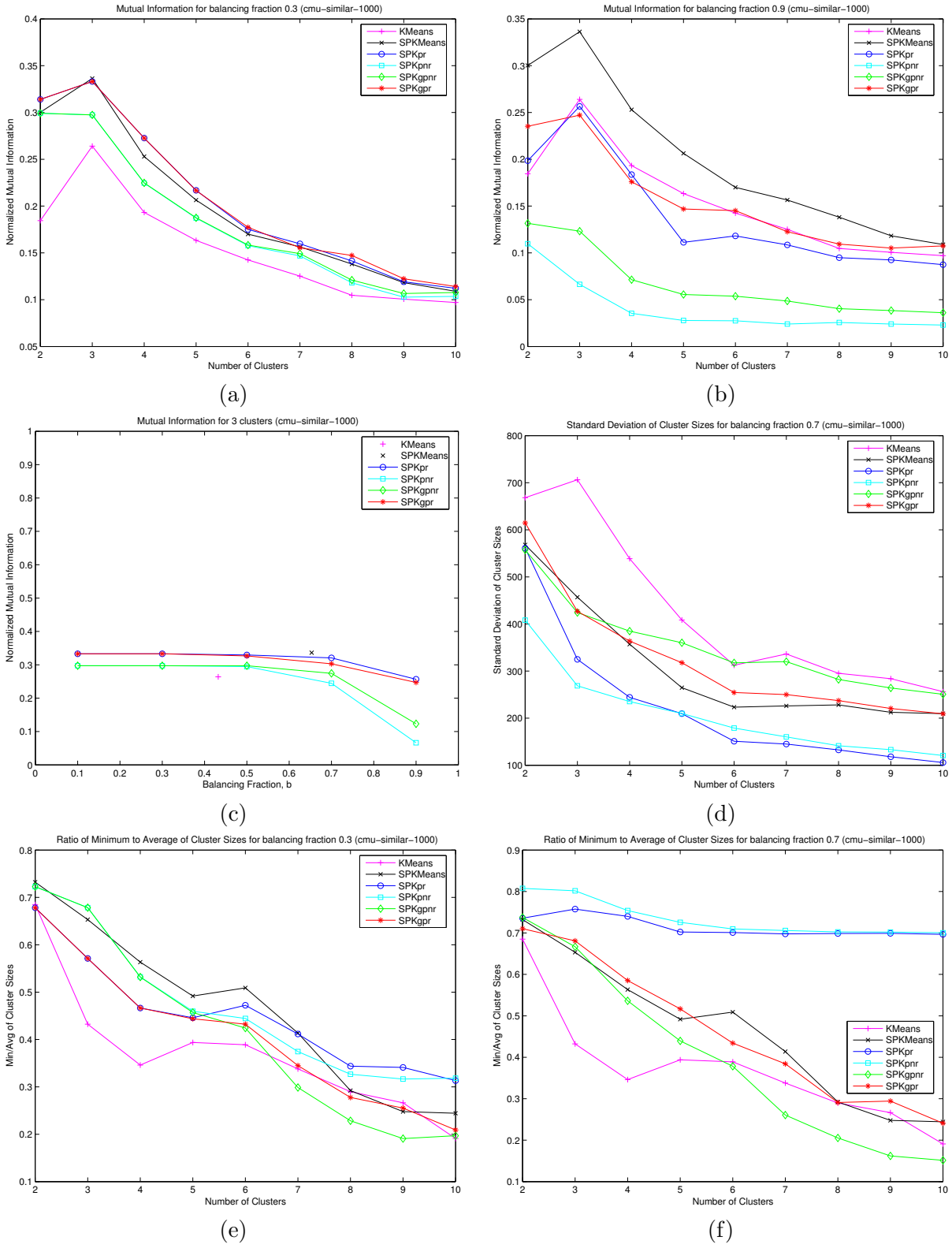

Figure 4: Results on similar-1000

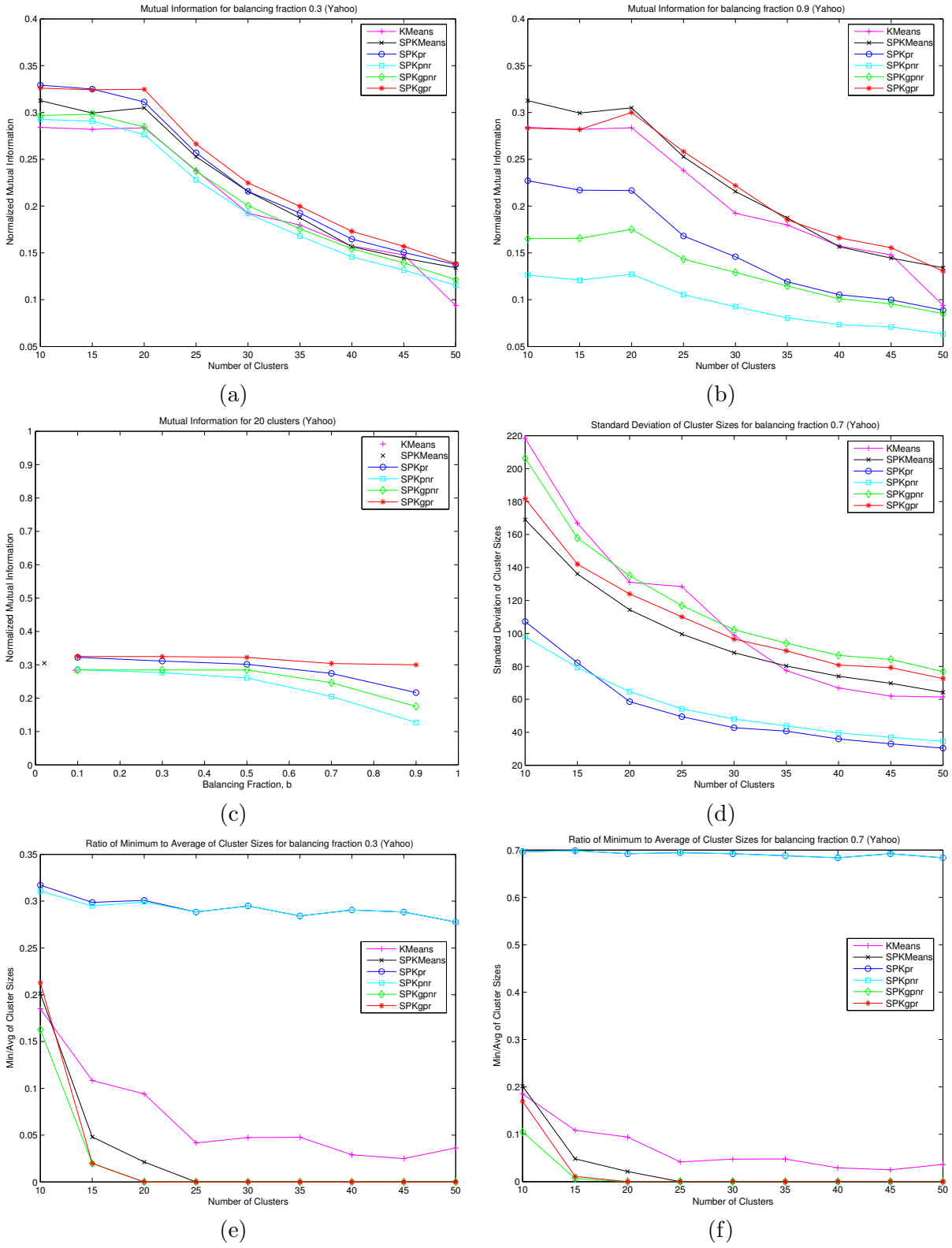

Figure 5: Results on yahoo

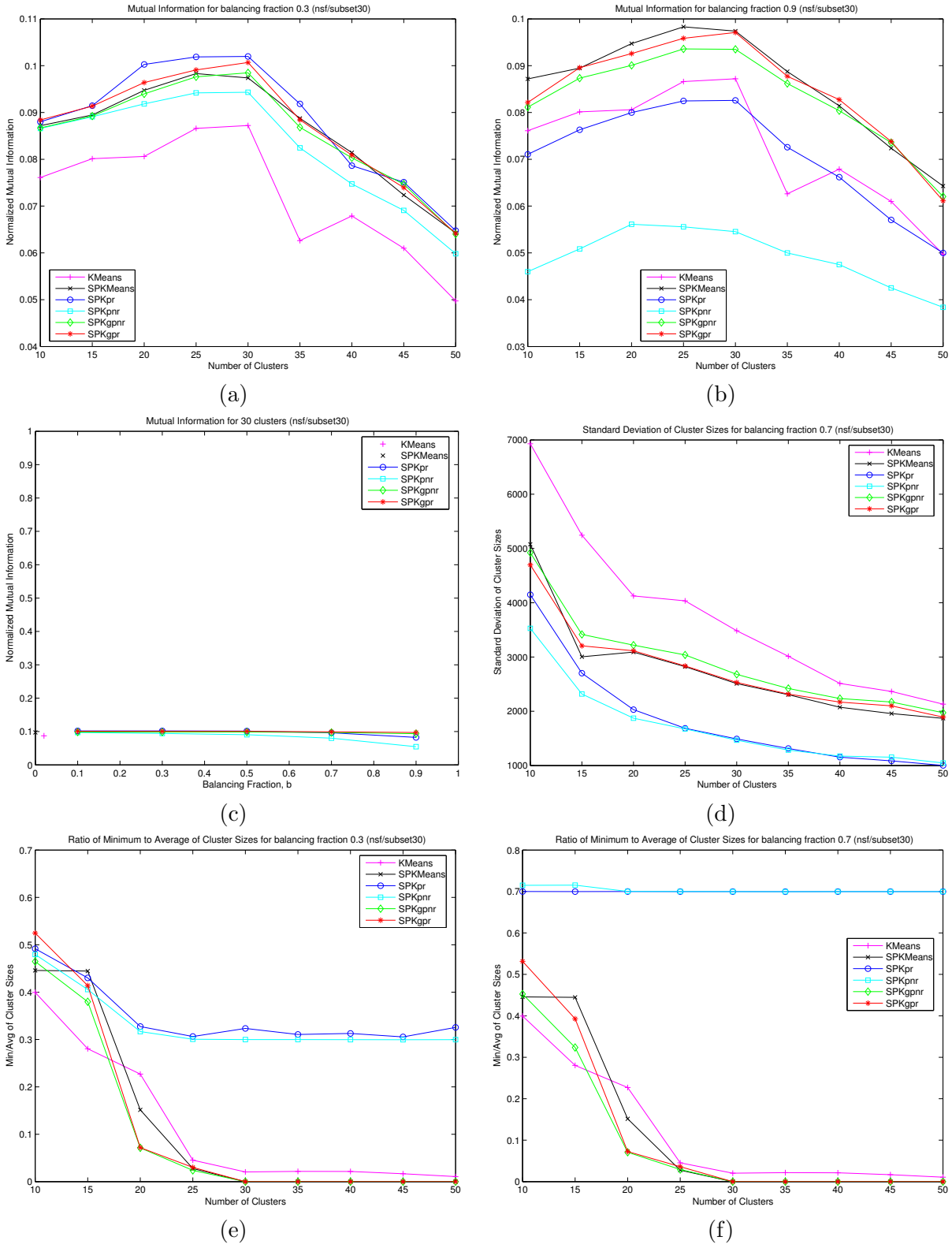

Figure 6: Results on nsf-top30