Frequency Sensitive Competitive Learning for Balanced Clustering on High-dimensional Hyperspheres

Arindam Banerjee and Joydeep Ghosh, *Senior Member, IEEE,*

***Abstract*— Competitive learning mechanisms for clustering in general suffer from poor performance for very high dimensional (> 1000) data because of "curse of dimensionality" effects. In applications such as document clustering, it is customary to normalize the high dimensional input vectors to unit length, and it is sometimes also desirable to obtain balanced clusters, i.e., clusters of comparable sizes. The spherical kmeans (`spkmeans`) algorithm, which normalizes the cluster centers as well as the inputs, has been successfully used to cluster normalized text documents in 2000+ dimensional space. Unfortunately, like regular `kmeans` and its soft EM based version, `spkmeans` tends to generate extremely imbalanced clusters in high dimensional spaces when the desired number of clusters is large (tens or more). In this paper, we first show that the `spkmeans` algorithm can be derived from a certain maximum likelihood formulation using a mixture of von Mises-Fisher distributions as the generative model and in fact it can be considered as a batch mode version of (normalized) competitive learning. The proposed generative model is then adapted in a principled way to yield three frequency sensitive competitive learning variants that are applicable to static data and produced high quality and well balanced clusters for high-dimensional data. Like `kmeans`, each iteration is linear in the number of data points and in the number of clusters for all the three algorithms. We also propose a frequency sensitive algorithm to cluster *streaming*[1] data. Experimental results on clustering of high-dimensional text data sets are provided to show the effectiveness and applicability of the proposed techniques.**

***Index Terms*— high-dimensional clustering, normalized data, balanced clustering, frequency sensitive competitive learning (FSCL), expectation maximization (EM), kmeans, streaming data, text clustering, scalable clustering.**

I. Introduction

Clustering or segmentation of data is a fundamental data analysis step that has been widely studied across multiple disciplines for over 40 years [1]. But several large datasets that are being acquired recently from scientific domains as well as the world wide web, have a variety of complex characteristics that severely challenge traditional methods for clustering, and also pose new requirements for evaluation, scalability, visualization and actionability of results [2]. This article is concerned with the clustering of objects represented by very high (hundreds or more) dimensional feature vectors of unit length into a fairly large (tens or more) number of groups of comparable sizes. Such situations are relevant for applications such as clustering of text documents, wherein it is advantageous to normalize the feature vectors (i.e., scale them to unit length) before applying a clustering algorithm.

One can broadly categorize clustering approaches into *generative* and *discriminative* ones. In a generative approach [3], [4], [5], [6], the data is modeled as being generated by an underlying parametric, probabilistic generation process. Reasonable values for the parameters are obtained from the input data, and the properties of the clusters are then inferred from these parameters. Discriminative approaches [7], [8], [9], on the other hand, make no assumptions whatsoever about the data points were generated. Instead, they assume that the data points belong to a metric space and the metric is known, or, to a Hilbert space where the inner-product is known. In other words, it is assumed that a well defined distance or similarity measure exists between any pair of objects, and the clustering process is essentially an attempt to partition the objects such that objects within the same cluster tend to have less dissimilarities (distances) as compared to objects in different clusters. The performance of an approach (and of a specific method within that approach) is quite data dependent; there is no clustering method that works the best across all types of data distributions. Generative models, however often provide better insight into the nature of the clusters. From an application point of view, a lot of domain knowledge can be incorporated into the generative models so that clustering of data brings out specific desirable patterns that one is looking for. It is for this reason that the generative (parametric) approach is referred to as the method of particular inference in statistical learning theory [7].

For the specific scenario of clustering high-dimensional, L_2 normalized data, we first propose that a generative model consisting of a mixture of von Mises-Fisher distributions, is a suitable approach. The model is then modified in a principled manner by techniques inspired by "conscience" mechanisms developed in the competitive learning literature, to yield clustering algorithms that produce high quality, balanced solutions for both static and streaming data. In this section, we briefly motivate the problem setting outlined above and also provide a brief background on competitive learning.

Manuscript recieved XXX xx, xxxx; revised XXX xx, xxxx. This research was supported in part by NSF under grant ECS-9900353, and research grant #8032 from Intel Corporation.

The authors are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, Texas, USA (emails: abanerje@ece.utexas.edu, ghosh@ece.utexas.edu.

[1]We use the term "streaming" rather than "on-line" since not only have the parameters to be updated online, but also each data point is seen only once.

A. Motivation

In some applications, the data points to be clustered reside in a high-dimensional feature space even after suitable preprocessing steps, including feature selection, have been carried out. For example, in the popular vector space representation of text documents, the dimensionality of the feature space is the size of the vocabulary [10]. Even after stemming is carried out and both very rare as well as very common terms have been discarded, the residual vocabulary often contains thousands of terms. In market basket analysis of large retail data, the number of significant products, each represented by one feature, may run in the thousands [11]. Both text and market basket data also have other properties in common: the data matrix is typically very sparse and only contains non-negative entries, and the underlying clusters are highly non-Gaussian in nature.

In general, sparsity and other issues due to the "curse of dimensionality" [12] make the clustering of highly non-Gaussian, high-dimensional data a very difficult problem to solve using density based approaches. Partitional methods such as k-means and its variants also fail miserably since the underlying assumption that the data can be well modeled by a mixture of k Gaussians (value of k being pre-specified) with identical covariance matrices, is violated. The way out is to exploit domain knowledge about the properties of the data and of the desired clusters. For example, if it is known that the data actually resides in a manifold that is of much lower dimensions than the embedding space, solving for a mixture of principal surfaces can help [13], [14].

This paper is aimed at applications for which the domain knowledge indicates that data is directional [15]. For such scenarios, the curse of dimensionality is somewhat alleviated by normalizing the data to unit L_2 norm, since only the direction of a data vector is relevant. For example, many works on document clustering normalize the document vectors to unit length after all other preprocessing and normalizations such as TF-IDF have been carried out. The cosine of the angle between two such normalized vectors then serves as the default similarity measure between the two documents that they represent. Normalization prevents larger documents from dominating the clustering results, and can be viewed as an application of domain-specific characteristics and requirements to alleviate the "curse of dimensionality" problems. A noteworthy algorithm for clustering normalized document vectors is spherical kmeans (`spkmeans`) [16], [17], in which the cluster representatives are also constrained to be of unit length, allocation of data points to their nearest representatives is based on cosine similarity, and, after a full pass through the data, the updated locations of the representatives are based on minimizing the average cosine between the cluster "center" and all the points assigned to that cluster. Note that, like `kmeans`, `spkmeans` is also a batch-iterative procedure. This algorithm has been successfully used to cluster text documents in 2000+ dimensional space, providing superior cluster definitions in the process [16], [17]. We shall show in Section 2 that `spkmeans` is really a batch version of a competitive learning algorithm.

Normalization of high-dimensional vectors before clustering is also fruitful for some other applications. In fact, it is meaningful for market basket data analysis if one is interested in, say, grouping customers based on the similarities between the percentages of their money spent on the various products.

Having shown the need for clustering high dimensional data residing on hyperspheres by drawing examples from document clustering and market basket analysis, we use the same two domains to motivate the desirability of obtaining *balanced* clusters, i.e., clusters of comparable sizes [18], [19]. In general, the natural clusters in the data may be of widely varying sizes, this variation may not be known beforehand and balanced solutions may not be important. However, several real life applications demand comparably sized segmentations of the data. For example, a direct marketing campaign often starts with segmenting customers into groups of roughly equal size or equal estimated revenue generation, (say, based on market basket analysis, or purchasing behavior at a web site), so that the same number of sales teams, marketing dollars etc., can be allocated to each segment. In large retail chains, one often desires product categories/groupings of comparable importance, since subsequent decisions such as shelf/floor space allocation and product placement are influenced by the objective of allocating resources proportional to revenue or gross margins associated with the product groups [11]. Similarly, in clustering of a large corpus of documents to generate topic hierarchies, balancing greatly facilitates 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.

In addition to application requirements, balanced clustering is sometimes also helpful because it tends to decrease sensitivity to initialization and to avoid outlier clusters (highly under-utilized representatives) from forming, and thus has a beneficial regularizing effect even in situations where balancing is not a requirement. This will be evident from our experimental results in section VI.

Unfortunately, `kmeans` type algorithms (including the EM approach) as well as the basic on-line competitive learning mechanisms for clustering are increasingly prone to yielding imbalanced solutions as the input dimensionality increases. This problem is exacerbated when a large (tens or more) number of clusters are needed, and it is well known that both hard and soft `kmeans` invariably result in some near-empty clusters in such scenarios.

B. Competitive Learning

Competitive learning techniques employ winner-take-all mechanisms to determine the most responsive cell to a given input [20], [21], [22]. If this cell or exemplar then adjusts its afferent weights to respond even more strongly to the given input, the resultant system can be shown to perform unsupervised clustering. For example, the non-normalized competitive learning version of Rumelhart and Zipser [21], essentially yields an on-line analogue of the popular `k-means` clustering algorithm. There are also soft competitive learning methods with multiple winners per input [23], that can be viewed as on-line analogues of soft batch-iterative clustering algorithms such as fuzzy c-means [24] as well as the expectation-maximization (EM)-based approach to clustering data modeled as a mixture of Gaussians [4].

To address the problem of obtaining clusters of widely varying sizes, a "conscience" mechanism was proposed for competitive learning in 1988 [25], that made frequently winning representatives less likely to win in the future because of their heavier conscience. This work was followed by the notable frequency sensitive competitive learning(FSCL) method [26]. FSCL was originally formulated to remedy the problem of under-utilization of parts of a codebook in Vector Quantization. Motivated by earlier work of Grossberg [20], the conscience mechanism used in FSCL multiplicatively scaled the distortion (distance of the exemplar or codebook vector from the input) by the number of times that exemplar was the winner in the past. Thus highly winning exemplars were discouraged from attracting new inputs. However, this mechanism was not derived from first principles or applied to high-dimensional, normalized clustering.

C. Outline

In this paper, we first show that the `spkmeans` algorithm can be derived from a certain maximum likelihood formulation using a mixture of von Mises-Fisher distributions as the generative model. This generative model is particularly suitable for describing directional data. Then, in section III, we propose a variant in which the dispersion of a mixture component decreases if more data points are attributed to it, thus introducing a conscience mechanism. This results in three batch-mode frequency sensitive competitive algorithms for normalized data. A fully on-line version of this algorithm applicable to streaming data is then suggested in section V. Experimental results on high-dimensional text clustering problems are presented in section VI.

A word about the notation: bold faced variables, e.g., $\mathbf{x}, \boldsymbol{\mu}$, etc., represent vectors, $\|\cdot\|$ denotes the L_2 norm, and sets are represented by calligraphic upper-case alphabets, e.g., $\mathcal{X}, \mathcal{Z}$,etc.. Probability density functions are denoted by lower case alphabets, e.g., $f(\cdot)$.

II. Clustering on a Hypersphere

The classical `kmeans` clustering algorithm gives the (local) maximum likelihood estimates of the means of k Gaussians [27] under certain assumptions [28], [29] by using the Expectation Maximization (EM) algorithm [30]. The EM algorithm is guaranteed to give a local optimum for these maximum likelihood estimates. We use a similar approach for deriving the `spkmeans` algorithm for clustering points on the surface of a hypersphere. Recall that the von Mises-Fisher (vMF) distribution is an analogue of the Gaussian distribution on a hypersphere [15], [31], [32] in that it is the maximum entropy distribution on the hypersphere when the first moment is fixed [33] under the constraint that the points are on a unit hypersphere. The density of a d-dimensional vMF distribution is given by

$$f(\mathbf{x}; \boldsymbol{\mu}, \kappa) = \frac{1}{Z_d(\kappa)} \exp\left(\kappa \mathbf{x}^T \boldsymbol{\mu}\right), \quad (1)$$

where $\boldsymbol{\mu}$, with L_2 norm $\|\boldsymbol{\mu}\| = 1$, represents the mean direction vector and κ is the dispersion around the mean, analogous to the mean and covariance for the multivariate Gaussian distribution. The normalizing coefficient is

$$Z_d(\kappa) = (2\pi)^{d/2} I_{d/2-1}(\kappa)/\kappa^{d/2-1}, \quad (2)$$

where $I_r(y)$ is the modified Bessel function of the first kind and order r [34]. Assume that there are n data points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ on the surface of a unit hypersphere and there are k vMF distributions $f_h, h = 1, \cdots, k$, such that each point has been generated following exactly one of these distributions. We want to estimate the parameters of the k vMF distributions so that the likelihood of the observed data is maximized. Like the Euclidean `kmeans` case, we initially assume κ to be a constant for all the k distributions. Let $\mathcal{Z} = \{\mathrm{z}_1, \cdots, \mathrm{z}_n\}$ be the so-called hidden random variable over $\{1, \cdots, k\}$ corresponding to the generation of the point $\mathbf{x}_i$ so that $\mathrm{z}_i = h$ if $\mathbf{x}_i$ was generated following f_h and 0 otherwise. Assuming the data points have been drawn independently, the log-likelihood of the observed data is given by

$$\begin{aligned} \mathcal{L}(\Theta|\mathcal{X}) &= \sum_{i=1}^{n} \ln f(\mathbf{x}_i; \boldsymbol{\mu}_{\mathrm{z}_i}) \\ &= \sum_{i=1}^{n} (\kappa \mathbf{x}_i^T \boldsymbol{\mu}_{\mathrm{z}_i} - \ln(Z_d(\kappa))), \end{aligned} \quad (3)$$

where $\boldsymbol{\mu}_{\mathrm{z}_i}$ is the mean of the vMF distribution that generated $\mathbf{x}_i$. Since κ is a constant, maximizing this log-likelihood with respect to the $\boldsymbol{\mu}_h, h = 1, \cdots, k$, is same as maximizing

$$\mathcal{J} = \sum_{i=1}^{n} \mathbf{x}_i^T \boldsymbol{\mu}_{\mathrm{z}_i} \quad (4)$$

under the constraint that $\|\mu_h\| = 1, \forall h$. Since the parameters μ_h as well as the distributions of the random variables z_i are not known, we can use the EM algorithm to do maximum likelihood estimation under incomplete information. If the parameters are set to random values in a primal M-step, the E-step of the algorithm involves *assigning* the data points to the most likely vMF distribution to have generated it. More precisely, we compute h^* so that[2]

$$h^* = \underset{h}{\operatorname{argmax}} \ \ln f(\mathbf{x}_i; \boldsymbol{\mu}_h) = \underset{h}{\operatorname{argmax}} \ \mathbf{x}_i^T \boldsymbol{\mu}_h. \quad (5)$$

Then, the M-step involves computing the $\mu_h, h = 1, \cdots, k$, using the current assignments of the data. The parameters are computed by maximizing the expected log-likelihood, the expectation being over the distribution of the z_is. Note that unlike many other applications of the M-step, the maximization in this case is a constrained maximization with the constraints $\|\mu_h\| = 1, \forall h$ and hence is done by using the Lagrange multiplier method. Let λ_h be the Lagrange multiplier corresponding to the constraint $\boldsymbol{\mu}_h^T \boldsymbol{\mu}_h = 1$.[3] Then

[2]One can also formulate a soft assignment, but soft clustering is outside the scope of this paper. See [32] for details

[3]There is a subtle difference between the constraints $\|\boldsymbol{\mu}_h\| = 1$ and $\boldsymbol{\mu}_h^T \boldsymbol{\mu}_h = 1$. Since this difference is not important in the present analysis, we choose to ignore it for simplicity.

the Lagrangian is given by

$$\begin{aligned}&L(\boldsymbol{\mu}_1,\cdots,\boldsymbol{\mu}_k,\lambda_1,\cdots,\lambda_k;\mathcal{X})\\ &\quad=\sum_{i=1}^{n}\mathbf{x}_i^T\boldsymbol{\mu}_h+\sum_{h=1}^{k}\lambda_h(\boldsymbol{\mu}_h^T\boldsymbol{\mu}_h-1)\\ &\quad=\sum_{h=1}^{k}\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i^T\boldsymbol{\mu}_h+\sum_{h=1}^{k}\lambda_h(\boldsymbol{\mu}_h^T\boldsymbol{\mu}_h-1),\end{aligned}$$

where $\mathcal{X}_h$ is a set such that if the current $z_i = h$, then $\mathbf{x}_i \in \mathcal{X}_h$. In other words, $\mathcal{X}_h$ the set of points assigned to the current h-th cluster. Now, taking derivatives of the Lagrangian with respect to $\boldsymbol{\mu}_h$ and λ_h and setting it to zero, for $h = 1, \cdots, k$, we get the following equations:

$$\boldsymbol{\mu}_h = \frac{1}{2\lambda_h}\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i, \tag{6}$$

$$\boldsymbol{\mu}_h^T\boldsymbol{\mu}_h = 1. \tag{7}$$

Substituting Eqn. 6 into Eqn. 7, we get

$$\lambda_h = \frac{1}{2}\parallel\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i\parallel. \tag{8}$$

Now, replacing λ_h in Eqn. 6 with Eqn. 8, we get the final M-step as

$$\boldsymbol{\mu}_h = \frac{\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i}{\parallel\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i\parallel}. \tag{9}$$

Repeating the steps given in Eqns.5 and 9 results in a gradient ascent scheme that, being an EM algorithm, is guaranteed to give a local maxima of the likelihood function in Eqn. 3 and hence also Eqn. 4 after convergence. This scheme was introduced as the spherical kmeans (spkmeans) algorithm by Dhillon *et. al.* [16] since the data points lie on the surface of the unit hypersphere. We have now provided a derivation of the same from maximum likelihood principles. In fact, we have obtained a batch mode version of normalized competitive learning [21]. Note that while the performance of this algorithm can be evaluated using Eqn. 3, the following objective function obtained by adding constant additive and multiplicative factors, is simpler and more interpretable:

$$\bar{\mathcal{J}} = \frac{1}{n}\sum_{h=1}^{k}\sum_{\mathbf{x}_i\in\mathcal{X}_h}\mathbf{x}_i^T\boldsymbol{\mu}_h. \tag{10}$$

$\bar{\mathcal{J}}$ can be interpreted as the average *cosine similarity* (cosine of the angle) between any vector $\mathbf{x}_i$ and its cluster representative $\boldsymbol{\mu}_{z_i}$. It serves as an intrinsic measure of cluster quality and will be called the spkmeans objective function.

III. Frequency Sensitive Assignments

From empirical studies [16], [35], [32] in clustering, spkmeans has been shown to be clearly superior to regular kmeans for directional data [32]. However, like its Euclidean space counterpart, it quite often gets stuck in poor local solutions resulting in empty clusters or clusters having very few points, for moderately large values of k. Both the formulations do not have any explicit way to guard against such a scenario. A similar problem had been reported in the signal processing community for the problem of vector quantization where some parts of the codebook were under-utilized as a result of poor local solutions to the optimization problem for codebook generation [21]. The problem was empirically addressed by using frequency sensitive competitive learning(FSCL) [26], [36]. FSCL is a conscience type competitive learning approach that overcomes the problems associated with simple competitive learning [26] and Kohonen's self-organizing feature maps in vector quantization applications. In the FSCL, the competitive computing units are penalized in proportion to the frequency of their winning, so that eventually all units participate in the quantization of the data space. Convergence properties of the FSCL algorithm to a local minima have been studied by approximating the final phase of the FSCL by a diffusion process described by a Fokker-Plank equation [37].

As mentioned previously, the kmeans algorithm can be viewed as an EM algorithm on a mixture of identity variance Gaussians assuming the cluster assignment hidden variables are distributed such that they take one value in $\{1, \cdots, k\}$ (for hard assignments) [29]. A frequency sensitive learning mechanism can be derived from this mixture of Gaussians framework by making each of the Gaussians *shrink* in proportion to the number of points that have been assigned to it. More precisely, if n_h points have been assigned to the h-th cluster, the covariance of its representative Gaussian is set to $\Sigma_h = \frac{1}{n_h}\mathbb{I}$ where $\mathbb{I}$ is the identity matrix. Note that if n_h is large for a particular h, then that Gaussian shrinks more in the sense that the density gets more peaked around the mean. The log-likelihood of a particular point $\mathbf{x}$ with respect to this Gaussian is given by

$$\begin{aligned}&\mathcal{L}(\mu_h,n_h|\mathbf{x})\\ &\quad=\ln\frac{1}{\sqrt{(2\pi)^d|\Sigma_h|}}\exp(-\frac{1}{2}(\mathbf{x}-\mu_h)^T\Sigma_h^{-1}(\mathbf{x}-\mu_h))\\ &\quad=-\frac{n_h}{2}\|\mathbf{x}-\mu_h\|^2-\frac{d}{2}\ln n_h-\frac{d}{2}\ln 2\pi,\end{aligned}$$

where d is the dimensionality of the data. Since data points are assigned to the most likely Gaussian to have generated it, the point $\mathbf{x}$ will be assigned to the cluster h^* where

$$\begin{aligned}h^* &= \underset{h}{\operatorname{argmax}}\ \mathcal{L}(\mu_h,n_h|\mathbf{x})\\ &= \underset{h}{\operatorname{argmin}}\ \left\{n_h\|\mathbf{x}-\mu_h\|^2+d\ln n_h\right\}.\end{aligned} \tag{11}$$

Thus, the higher n_h is, the lower is the chances of a point getting assigned to that cluster. This is exactly what any FSCL tries to achieve. Interestingly, the empirically proposed FSCL method [26] only considers $n_h\|x-\mu_h\|^2$. Our formal treatment of the idea results in an extra second term, namely $d\ln n_h$.

Using the same basic idea, we propose a change in the formulation of spherical kmeans in order to prevent poor local solutions. Rather than keeping κ constant, we propose to make it inversely proportional to the number of points assigned to the corresponding distribution. Thus, if n_h is the number of points assigned to f_h, then we set $\kappa_h \propto 1/n_h$. Intuitively, this is akin to using shrinking Gaussians in the Euclidean space in the sense that as more points are assigned to a particular cluster, the "width" of its representative Gaussian reduces. As a result, effective distance of points from this cluster increases, or, in the spherical case, the similarity of points from this cluster decreases. Thus, if a point $\mathbf{x}$ is such that $\mathbf{x}^T\boldsymbol{\mu}_1 = \mathbf{x}^T\boldsymbol{\mu}_2$ but $n_{h_1} < n_{h_2}$, then $\mathbf{x}$ has a higher likelihood of having been generated from f_{h_1} than f_{h_2} in the frequency sensitive setting. Hence, the likelihood of points going to clusters having less number of points is higher and this implicitly discourages poor local solutions having empty clusters or clusters having very small number of points.

Formally, let $\kappa_h \propto 1/n_h \Rightarrow \kappa_h = c/n_h$, where c is a suitable proportionality constant. Then, the log-likelihood of data-point $\mathbf{x}_i$ having been generated from f_h is given by

$$\begin{aligned}
&\log f(\mathbf{x}_i;\boldsymbol{\mu}_h,\kappa_h)\\
&\quad= \frac{c}{n_h}\,\mathbf{x}_i^T\boldsymbol{\mu}_h - \log Z_d\left(\frac{c}{n_h}\right)\\
&\quad\equiv \frac{c}{n_h}\,\mathbf{x}_i^T\boldsymbol{\mu}_h - \log(I_{d/2-1}(c/n_h)) - (\frac{d}{2}-1)\log n_h,
\end{aligned}$$

where $\mathcal{F}(h) \equiv \mathcal{G}(h)$ means that $\text{argmax}_h\ \mathcal{F}(h) = \text{argmax}_h\ \mathcal{G}(h)$. For simplifying the expression further, we choose $c = nd^2/2k$. Then, noting that $n_h = O(n/k)$ and d is a large number so that $nd^2/2kn_h \gg d$, using the fact that $I_n(x) \approx e^x/\sqrt{2\pi x}$ for fixed n and $x \gg n$ [34], we get

$$\begin{aligned}
\log\, I_{d/2-1}(c/n_h) &= \log I_{d/2-1}\left(\frac{n/k}{n_h}\cdot\frac{d^2}{2}\right)\\
&\approx \frac{n/k}{n_h}\cdot\frac{d^2}{2} - \frac{1}{2}\log(2\pi\frac{n/k}{n_h}\cdot\frac{d^2}{2})\\
&\equiv \frac{n/k}{n_h}\cdot\frac{d^2}{2} + \frac{1}{2}\log n_h.
\end{aligned}$$

Hence,

$$\begin{aligned}
&\log\ f(\mathbf{x}_i;\boldsymbol{\mu}_h,\kappa_h)\\
&\equiv \frac{n/k}{n_h}\cdot\frac{d^2}{2}\mathbf{x}_i^T\boldsymbol{\mu}_h - \frac{n/k}{n_h}\cdot\frac{d^2}{2} - \frac{1}{2}\log n_h - (\frac{d}{2}-1)\log n_h\\
&= \frac{n/k}{n_h}\cdot\frac{d^2}{2}\left(\mathbf{x}_i^T\boldsymbol{\mu}_h + 1\right) - \frac{d-1}{2}\log n_h\\
&\approx \frac{d}{2}\left[\frac{(n/k)d}{n_h}\left(\mathbf{x}_i^T\boldsymbol{\mu}_h + \boldsymbol{\mu}_h^T\boldsymbol{\mu}_h\right) - \log n_h\right]\\
&\equiv \frac{(n/k)d}{n_h}(\mathbf{x}_i + \boldsymbol{\mu}_h)^T\boldsymbol{\mu}_h - \log n_h.
\end{aligned}$$

Hence, the most likely distribution to have generated the point $\mathbf{x}_i$ is given by

$$h^* = \underset{h}{\text{argmax}}\left\{\frac{(n/k)d}{n_h}(\mathbf{x}_i + \boldsymbol{\mu}_h)^T\boldsymbol{\mu}_h - \log n_h\right\} \quad (12)$$

$$= \underset{h}{\text{argmax}}\,\frac{1}{n_h}\left\{\mathbf{x}_i^T\boldsymbol{\mu}_h + 1 - \frac{n_h}{(n/k)d}\log n_h\right\}. \quad (13)$$

Note, from Eqn. 13, that the spherical kmeans assignment function (Eqn. 5) now gets a multiplicative and an additive term, both of which penalize larger clusters. Further note that this particular form of the likelihood function is due to our choice of the proportionality constant and other values of the constant will give slightly different forms for this likelihood function. However, this particular choice helps us use an asymptotic behavior of the modified Bessel function of the first kind thereby making the formulation computationally tractable.

In the next few sections, we shall present algorithms for clustering data based on the frequency sensitive assignment rules as derived above (Eqn. 13). We focus on two types of problems – (a) in which the data points is static and the algorithm can read the data as many times as required, and (b) in which the data points is streaming so that algorithm can read every data point exactly once. In section IV, we present three algorithms for clustering static data – `fs-spkmeans` is a direct extension of `spkmeans` using the frequency sensitive assignments from Eqn. 13; `pifs-spkmeans` is a partially incremental version of `fs-spkmeans` where the effective number of points per cluster are updated incrementally after processing every point and the mean of every cluster is updated in batch once in every iteration (after processing all the points); and `fifs-spkmeans` is a fully incremental version of `fs-spkmeans` where both the effective number of points per cluster and the cluster means are updated after processing every point. Note that all these algorithms need to know the number of points to be processed up-front and hence are applicable to static data only. In section V, we present `sfs-spkmeans`, an algorithm for frequency sensitive clustering of streaming data.

IV. Algorithms for Static Data

Based on the analysis presented in section III and the frequency sensitive assignments according to Eqn. 13, we first present the algorithm `fs-spkmeans` (**frequency sensitive** `spkmeans`) that is an extension of `spkmeans` and is applicable to static data that can be read as many times as necessary.

Algorithm `fs-spkmeans`

```
1.  Set iteration count $t \leftarrow 0$. Choose $k$ points (unit vectors)
    as the cluster means $\boldsymbol{\mu}_h^{(0)}$, set $n_h^{(0)} \leftarrow \frac{n}{k}$, $h = 1,\cdots,k$.
2.  Repeat until convergence
2a.   Assign each data-point $\mathbf{x}$ to the cluster $f_{h^*}^{(t)}$ where
          $h^* = \text{argmax}_h \frac{1}{n_h^{(t)}}\left\{\mathbf{x}^T\boldsymbol{\mu}_h + 1 - \frac{n_h}{(n/k)d}\log n_h^{(t)}\right\}$
2b.   $n_h^{(t+1)} \leftarrow |\{\mathbf{x} : \mathbf{x} \in f_h^{(t)}\}|$
2c.   $\boldsymbol{\mu}_h^{(t+1)} \leftarrow (\sum_{\mathbf{x}\in f_h^{(t)}}\mathbf{x})/\|\sum_{\mathbf{x}\in f_h^{(t)}}\mathbf{x}\|$
2d.   $t \leftarrow (t+1)$
```

Though the algorithm `fs-spkmeans` is motivated by the FSCL, it does not have the incremental flavor of FSCL. To study the effect of the incremental behavior of `fs-spkmeans`, we present and empirically evaluate two variants of this algorithm. The first, called `pifs-spkmeans` (**partially incremental** `fs-spkmeans`), basically incorporates step 2b of `fs-spkmeans` into step 2a. In other words, in each iteration t, as soon as a point gets assigned to the h-th cluster, the value of $n_h^{(t)}$ is updated. The algorithm is presented below.

Algorithm `pifs-spkmeans`

```
1.  Set iteration count $t \leftarrow 0$. Choose $k$ points (unit vectors)
    as the cluster means $\boldsymbol{\mu}_h^{(0)}$, set $n_h^{(0)} \leftarrow \frac{n}{k}$, $h = 1,\cdots,k$.
```

```
2.  Repeat until *convergence*
2a.   For $i = 1$ to $n$,
        (i)  Assign the data-point $\mathbf{x}_i$ to the cluster $f_{h^*}^{(t)}$ where
               $h^* = \mathrm{argmax}_h \frac{1}{n_h^{(t)}} \left\{ \mathbf{x}^T \boldsymbol{\mu}_h + 1 - \frac{n_h}{(n/k)d} \log n_h^{(t)} \right\}$
        (ii) $n_{h^*}^{(t)} \leftarrow n_{h^*}^{(t)} + 1$ ; $n_h^{(t)} \leftarrow n_h^{(t)} - \frac{1}{k}$, $\forall h$
2b.   $\boldsymbol{\mu}_h^{(t+1)} \leftarrow (\sum_{x \in f_h^{(t)}} x)/\| \sum_{x \in f_h^{(t)}} x \|$
2c.   $n_h^{(t+1)} \leftarrow n_h^{(t)}, h = 1, \cdots, k, \quad t \leftarrow (t+1)$
```

The second variant is called `fifs-spkmeans` (**fully incremental** `fs-spkmeans`), and has the full flavor of competitive learning. In this scheme, in a particular *epoch*, i.e., a run through all the data points, as soon as a point gets assigned to the h-th cluster, both n_h and $\boldsymbol{\mu}_h$ are updated. Thus, in this scheme, we have shrinking as well as moving vMF distributions trying to model the data. The basic algorithm is presented below.

```
**Algorithm** fifs-spkmeans
1.   Choose $k$ points (unit vectors) as the cluster means $\boldsymbol{\mu}_h$,
     set $n_h \leftarrow \frac{n}{k}, h = 1, \cdots, k$.
2.   Repeat until *convergence*
2a.    For $i = 1$ to $n$
         (i)   Assign the data-point $\mathbf{x}_i$ to the cluster $f_{h^*}$ where
                 $h^* = \arg\max_h \frac{1}{n_h} \left\{ \mathbf{x}^T \boldsymbol{\mu}_h + 1 - \frac{n_h}{(n/k)d} \log n_h \right\}$
         (ii)  $n_{h^*} \leftarrow n_{h^*} + 1$ ; $n_h \leftarrow n_h - \frac{1}{k}$, $\forall h$
         (iii) $\boldsymbol{\mu}_h \leftarrow (\boldsymbol{\mu}_h + \frac{1}{n_h}(x_i - \boldsymbol{\mu}_h))/\|\boldsymbol{\mu}_h + \frac{1}{n_h}(x_i - \boldsymbol{\mu}_h)\|$
```

Note that in both the incremental algorithms, after each point is assigned to a cluster and its count incremented, a constant $1/k$ is subtracted from each n_h. This ensures that at any point of time, the total number of points in all the clusters add up to n.

V. Algorithm for Streaming Data

The algorithms presented in section IV necessarily need to know the number of data points to be processed from beforehand. They also need to make multiple read accesses over all the data points. Note that neither of these conditions is satisfied when the application demands clustering of streaming data. Streaming data is often typical of non-stationary environments requiring continuous on-line adaptation [38]. The need for clustering streaming, normalized data is encountered, for example, for real-time incremental grouping of news stories or message alerts that are received on-line. In classical pattern recognition, streaming data are often encountered in the form of non-linear time series [39]. It is not surprising that much work on analyzing streaming data has been done in the neural network, signal processing and applied physics communities, starting from the early days of ADALINE [40], [41], [42], [43]. More recently, some machine learning researchers have also got interested in this problem [44], [45]. But to date, there has been little work on clustering of such data [46], [47], and there is a lack of benchmark data sets for the same.

An algorithm working on streaming data gets to read the data only once. Thus none of the algorithms presented in section IV can be applied to streaming data. In this section, we present `sfs-spkmeans` (**streaming** `fs-spkmeans`, a variant of `fs-spkmeans` that can be applied to streaming data since it does not need to know the number of points to be processed and reads every data-point exactly once. Note that the online version may actually be more applicable in certain real life scenarios, e.g., when data is being collected incrementally over time, or, when the clustering has to be done by making a single pass over the data kept in a database.

For constructing the online variant, we first note that a non-normalized mean $\mu^{(t+1)}$ of $(t+1)$ data points can be written as a recursion in terms of $\mu^{(t)}$ [48] as follows:

$$\mu^{(t+1)} = \mu^{(t)} + \frac{1}{t+1}(\mathbf{x}_{t+1} - \mu^{(t)}). \quad (14)$$

If the data is obtained from a stationary process, i.e., the parameters of the underlying generative model does not change with time, then $\mu^{(t)}$, as computed by the above recursion will converge, and do not need updating after sufficiently large t. However, typical streaming data is non-stationary. There are two popular approaches taken in such cases: (i) if the data characteristics change abruptly, then such breakpoints can be detected, and a model is fitted for each segment (regime) between two successive breakpoints, assuming stationarity within such segments. Piecewise autoregressive modeling is an example of such an approach. (ii) If the data characteristics vary slowly over time, the problem may be addressed by discounting the past – an approximation recursion can be used that keeps an exponentially decaying window over the history of observations and maintains the effective count c_{t+1} of the history rather than the exact $(t+1)$. More precisely, the approximate recursion for the mean [48] is given by:

$$\tilde{\mu}^{(t+1)} = \tilde{\mu}^{(t)} + \frac{1}{c_{t+1}}(\mathbf{x}_{t+1} - \tilde{\mu}^{(t)}),$$

where $c_{t+1} = (1 - 1/L)c_t + 1$ and L is a large number [48], [49], [50]. Note that this exponential decay factor of $(1 - 1/L)$ ensures that c_{t+1} converges from below to L. Thus, after the "cold start" period is over, the history maintained in the computation has an effective length L. The choice of L depends on the degree of non-stationarity, and a fundamental tradeoff between resolution and memory depth is encountered [51]. We take a similar approach for approximating the normalized mean. The normalized mean of $(t+1)$ data points can be written as a normalized recursion as follows:

$$\boldsymbol{\mu}^{(t+1)} = \frac{\boldsymbol{\mu}^{(t)} + \frac{1}{L_{t+1}} \left\{ \mathbf{x}_{t+1} - (L_{t+1} - L_t)\ \boldsymbol{\mu}^{(t)} \right\}}{\|\boldsymbol{\mu}^{(t)} + \frac{1}{L_{t+1}} \left\{ \mathbf{x}_{t+1} - (L_{t+1} - L_t)\ \boldsymbol{\mu}^{(t)} \right\}\|}, \quad (15)$$

where $L_t = \| \sum_{i=1}^t \mathbf{x}_i \|$. Again, using L_t in this recursion results in similar issues as outlined above for the non-normalized case. Using the same exponential decay approximation, we have

$$\tilde{L}_{t+1} = (1 - 1/L)\tilde{L}_t + \|x_{t+1}\| = (1 - 1/L)\tilde{L}_t + 1, \quad (16)$$

since $\|x_{t+1}\| = 1$, and $\tilde{L}_0 = 0$. It can be easily seen that $\lim_{t \to \infty} L_t = L$. A direct calculation using the recursion above shows that

$$\tilde{L}_{t+1} - \tilde{L}_t = (1 - 1/L)^t. \quad (17)$$

Now, for a large L and $t \ll L$, from Eqn. 17 we have $\tilde{L}_{t+1} - \tilde{L}_t \approx 1$. Using this and the approximations of Eqn. 16, following Eqn. 15, the approximate normalized recursion for $\boldsymbol{\mu}^{(t)}$ is given by

$$\tilde{\boldsymbol{\mu}}^{(t+1)} = \frac{\tilde{\boldsymbol{\mu}}^{(t)} + \frac{1}{\tilde{L}_{t+1}}(\mathbf{x}_{t+1} - \tilde{\boldsymbol{\mu}}^{(t)})}{\|\tilde{\boldsymbol{\mu}}^{(t)} + \frac{1}{\tilde{L}_{t+1}}(\mathbf{x}_{t+1} - \tilde{\boldsymbol{\mu}}^{(t)})\|}, \tag{18}$$

where $\tilde{L}_{t+1}$ is given by Eqn. 16.

To make the frequency sensitive version of spherical kmeans applicable to streaming data, as before, we want to make $\kappa_h \propto 1/n_h$. However, the number of points to be processed, n_h is unknown and may be unbounded. Therefore, we use the same exponential decay recursion for n_h so that $\tilde{n}_h^{(t+1)} = (1 - 1/L)\tilde{n}_h^{(t)} + 1$ and $\tilde{n}_h^{(0)} = 0$. Note that the recursion and the base case for $\tilde{n}_h^{(t)}$ and $\tilde{L}_t$ are exactly the same so that, for notation we can use only one of them. We choose to use $\tilde{n}_h^{(t)}$. Then, if x_{t+1} is assigned to cluster h, from Eqn. 18, we have

$$\tilde{\boldsymbol{\mu}}_h^{(t+1)} = \frac{\tilde{\boldsymbol{\mu}}_h^{(t)} + \frac{1}{\tilde{n}_h^{(t+1)}}(x_{t+1} - \tilde{\boldsymbol{\mu}}_h^{(t)})}{\|\tilde{\boldsymbol{\mu}}_h^{(t)} + \frac{1}{\tilde{n}_h^{(t+1)}}(x_{t+1} - \tilde{\boldsymbol{\mu}}_h^{(t)})\|}, \tag{19}$$

Now, in the limit, all the clusters will have a perfect balancing with effectively L points per cluster and hence L plays the role of n/k in the static case. Thus, following the static case, the proportionality constant in $\tilde{\kappa}_h^{(t)} \propto 1/\tilde{n}_h^{(t)}$ is set to $Ld^2/2$. Since $Ld^2/2n_h \gg d$, following the previous argument and replacing n/k with L, the most likely distribution to have generated a particular point $\mathbf{x}_i$ is given by

$$h^* = \arg\max_h \frac{1}{\tilde{n}_h^{(t)}} \left\{ \mathbf{x}_i^T \tilde{\boldsymbol{\mu}}_h^{(t)} + 1 - \frac{\tilde{n}_h^{(t)}}{Ld} \log \tilde{n}_h^{(t)} \right\}. \tag{20}$$

Using the above equation, we present `sfs-spkmeans`, the variant of frequency sensitive spherical kmeans applicable to streaming data.

Algorithm `sfs-spkmeans`

```
1.   Set data count $t \leftarrow 0$. Choose $k$ points (unit vectors) as
     the cluster means $\tilde{\boldsymbol{\mu}}_h^{(0)}$, set $\tilde{n}_h^{(0)} \leftarrow 0$, $h = 1, \cdots, k$.
2.   For the next data-point $\mathbf{x}_{t+1}$
2a.    Assign $\mathbf{x}_{t+1}$ to the cluster $f_{h^*}^{(t)}$ where
         $h^* = \arg\max_h \frac{1}{\tilde{n}_h^{(t)}} \left\{ \mathbf{x}_{t+1}^T \tilde{\boldsymbol{\mu}}_h^{(t)} + 1 - \frac{\tilde{n}_h^{(t)}}{Ld} \log \tilde{n}_h^{(t)} \right\}$
2b.    $\tilde{n}_{h^*}^{(t+1)} \leftarrow (1 - 1/L)\tilde{n}_{h^*}^{(t)} + 1$
2c.    $\tilde{\boldsymbol{\mu}}_{h^*}^{(t+1)} \leftarrow \frac{(\tilde{\boldsymbol{\mu}}_{h^*}^{(t)} + \frac{1}{\tilde{n}_{h^*}^{(t+1)}}(\mathbf{x}_{t+1} - \tilde{\boldsymbol{\mu}}_{h^*}^{(t)}))}{\|\tilde{\boldsymbol{\mu}}_{h^*}^{(t)} + \frac{1}{\tilde{n}_{h^*}^{(t+1)}}(\mathbf{x}_{t+1} - \tilde{\boldsymbol{\mu}}_{h^*}^{(t)})\|}$
2d.    For $h \neq h^*$,  $\tilde{n}_h^{(t+1)} \leftarrow \tilde{n}_h^{(t)}$, $\tilde{\boldsymbol{\mu}}_h^{(t+1)} \leftarrow \tilde{\boldsymbol{\mu}}_h^{(t)}$
2d.    $t \leftarrow (t + 1)$
```

Computational Requirements: Finally, a word on the complexity of the proposed approaches. Since all of the proposed approaches follow the basic infrastructure of `kmeans`, each iteration is linear in the number of data-points and the number of clusters. For the static algorithms, under realistic assumptions [32], the algorithms converge within a finite number of iterations. Note that, if need be, the approximations can be totally avoided with some extra computational effort [32]. For the streaming data, since the data points are processed one at a time, the algorithm is of course linear in the number of data points, but in a rather different sense. Further, for each data point, the streaming algorithm is linear in the number of clusters. Hence, all the proposed approaches are very scalable and can be employed in large scale clustering tasks.

VI. Experimental Results

In this section, experimental results on the proposed ideas are described. We present results on three high-dimensional text datasets - the Classic3 dataset[4], the News 20 dataset[5] and the Yahoo news dataset[6](K1) for the empirical performance analysis. In spite of being high-dimensional, sparse datasets, they have quite different properties that are quite useful in demonstrating the biases of the proposed algorithms.

The Classic3 dataset contains 3893 files, 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 toolkit MC [17] was used for creating the high-dimensional vector space model for the text documents and a total of 6061 words were used. Thus, each document, after normalization, is represented as a unit vector in a 6061 dimensional space. This is a relatively simple dataset in the sense that the documents in the 3 clusters are on completely different topics. Also, the *natural clusters* [7] are quite balanced.
The News20 dataset is a collection of 19,997 messages, collected from 20 different usenet newsgroups, with approximately 1000 messages per newsgroup. chosen at random and partitioned by newsgroup name. The headers for each of the messages were removed so that they do not bias the results. Using the toolkit MC, the high-dimensional model had a total of 26099 words. This is a typical dataset that one may encounter in real life — it is very high-dimensional, sparse and there is significant overlaps between the newsgroups. In fact, some cross-posted articles appear multiple times in the dataset – once under every group in which it was posted. Another feature of this dataset is that the natural clusters are perfectly balanced.
The Yahoo20 dataset (K-series) is a collection of 2340 Yahoo news articles belonging one of 20 different Yahoo categories. The K1 set actually gives the high-dimensional vector space model having 21839 words. After normalization, the data points reside on the surface of a 21839 dimensional hypersphere. 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 two previous datasets. This dataset helps in studying

[4]http://www.cs.utexas.edu/users/yguan/datamining/project/project.html
[5]http://www.ai.mit.edu/people/jrennie/20Newsgroups/
[6]ftp://ftp.cs.umn.edu/users/boley/PDDPdata/
[7]All three data sets used have class labels. We call the clustering indicated by the class labels as the "natural clustering" of the dataset. Extrinsically evaluating clustering quality by comparing cluster labels with class labels is quite common, but such results should be presented with a caveat, since some classes can be multi-modal, and classes may overlap as well, as is quite evident in the News20 dataset.

the effect of the balancing bias of the proposed algorithms on the cluster quality if the natural clusters are highly unbalanced.

A. Performance Measures

We study four performance measures to get a comparative understanding of the proposed algorithms – two of them measure cluster quality whereas the other two measure cluster balancing.

Cluster Quality is evaluated by an extrinsic and an intrinsic measure. Extrinsic measures such as purity and entropy of clusters [52] can be used if external information such as class labels for all data points can be obtained [2]. An increasingly popular extrinsic measure is the mutual information between the cluster assignments and a pre-existing labeling of the dataset. Formally, if X is a random variable for the cluster assignments and Y is a random variable for the pre-existing labels on the same data, then the mutual information $I(X;Y) = E_{X,Y}[\ln \frac{p(X,Y)}{p(X)p(Y)}]$ is the amount of statistical information shared by X and Y [53]. If m_{hl} is the number of documents in the h-th cluster, $h \in \{1, \cdots, k\}$, that has class label $l \in \{1, \cdots, c\}$, then the empirical estimate of the probability of the joint event $\{X = h, Y = l\}$ is computed as $p(X = h, Y = l) = m_{hl}/n$, where n is the total number of documents. The mutual information is computed using the empirical estimates for the joint events and the corresponding empirical marginals. We shall use a *normalized mutual information* (NMI) measure so that the numbers are in the range $[0, 1]$. The normalization is done using the arithmetic mean of the maximum possible entropies of the empirical marginals, i.e., NMI$(X,Y) = \frac{I(X,Y)}{(\log k + \log c)/2}$. The NMI [11] measures the amount of statistical similarity between the cluster assignments and the pre-existing labels under an appropriate (constant) scaling. This measure is better than certain other commonly used extrinsic measures such as entropy or purity [52], in the sense that NMI does not necessarily increase with increase in the number of clusters k, whereas both entropy and purity do.

As a second point of reference, an intrinsic measure of cluster quality, as indicated by the `spkmeans` *objective function* (SOF) value (Eqn. 10), is used. Note that using this measure favors `spkmeans` which only addresses this measure, while all the proposed methods attempt to optimize modified versions of this objective that also weave in balancing constraints.

Cluster Balancing is also evaluated by two measures. One measure is the *standard deviation in cluster sizes* (SDCS) for a given number of clusters requested from the algorithm. As mentioned earlier, obtaining balanced clusters, i.e., clusters with approximately equal sized clusters, is often an application requirement, or a desirable regularization property. SDCS is one measure that helps in understanding the balancing behavior of a clustering algorithm. Thus, if $\{n_1, \cdots, n_k\}$ are the sizes of the k clusters generated by an algorithm, then SDCS = $\{\frac{1}{k-1}\sum_{h=1}^{k}(n_h - \frac{n}{k})^2\}^{1/2}$.

In addition, it is also useful to know whether an algorithm is returning empty or extremely small clusters. To quantify this behavior, the second measure we use is the ratio of the minimum cluster size generated by an algorithm to the expected cluster size under perfect balancing. We shall refer to this measure as *ratio of minimum to expected* (RME). By definition, RME = $(\min\{n_1, \cdots, n_k\})/(n/k)$.

B. Experiments with Static Algorithms

In this section, we present results on the performance of the proposed static algorithms – `fs-spkmeans`, `pifs-spkmeans` and `fifs-spkmeans` – and compare them with that of the basic `spkmeans` algorithm[8]. We study the algorithms over a reasonable (depending on the dataset under consideration) range of values for the number of clusters to get a better understanding of their properties. All the results presented are averaged over 10 runs. The initial k means of the spherical kmeans were generated by computing the mean of the entire data and making k small random perturbations to this mean [17]. For stability and repeatability, the frequency sensitive algorithms were initialized at points of local minima of the `spkmeans` objective function.

1) The Classic3 dataset: On the Classic3 dataset, all the algorithms perform quite similarly in terms of the two cluster quality measures. All the four algorithms achieve their individual highest values of NMI at $k = 3$, which is the number of natural clusters in the data (Fig. 1(a)). Even for other values of k, they perform quite similarly in terms of NMI, though `pifs-spkmeans` seems to be perform a little differently from the rest of the group. The SOF values for all the algorithms are almost the same (Fig. 1(b)).

A difference in their performance is observed while studying the cluster balancing measures. The `pifs-spkmeans` algorithm gives a much lower SDCS as compared to the other algorithms (Fig. 1(c)). Also, it gives a much higher RME as compared to the other algorithms (Fig. 1(d)). This points out to the fact that `pifs-spkmeans` has high bias towards balancing.

2) The News20 dataset: The News20 dataset demonstrates the basic nature of the four algorithms quite well. In all the experiments, and in terms of all the performance measures, `fs-spkmeans` and `fifs-spkmeans` show very similar behavior. As seen before in the Classic3 dataset, `pifs-spkmeans` has a high bias towards balancing, whereas `spkmeans` has no explicit mechanism for balancing.

All the algorithms achieve their individual highest values of the NMI at $k = 20$, which is the correct number of clusters (Fig. 2(a)). At $k = 20$, `fifs-spkmeans` and `fs-spkmeans` perform better than the other two in terms of the NMI, and also show good balancing. For lower values of k, `pifs-spkmeans` performs worse than the other three which have quite similar behavior. For much higher values of k, `spkmeans` has significantly higher values of NMI compared to the three proposed approaches, since it starts generating zero-sized clusters (Fig. 2(d)) in order to maintain the NMI and objective at a reasonable value. On the other hand, since none of the proposed algorithms generate zero sized clusters, their performance in terms of NMI suffers. As seen from

[8]We do not compare with `kmeans` or equivalent algorithms since they perform miserably on high dimensional text data [52]

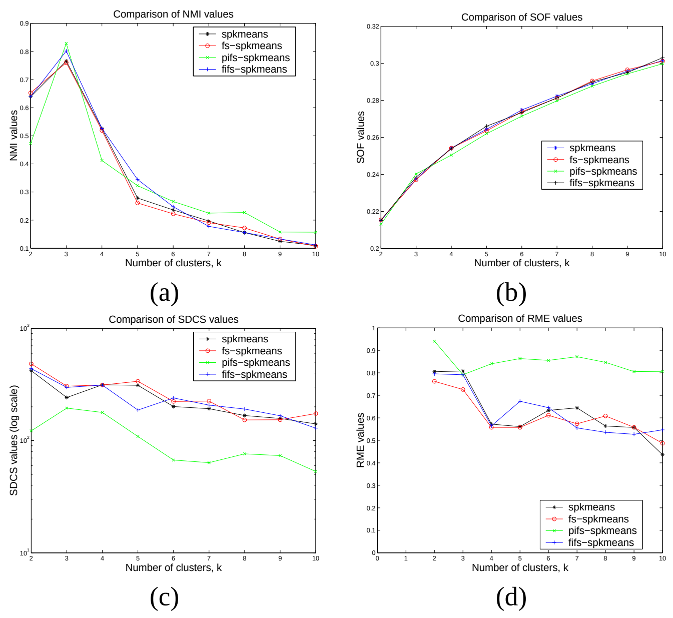


Fig. 1. Comparison between the static algorithms on the Classic3 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) the standard deviation in cluster sizes, and (e) the ratio of the minimum to expected cluster size values, averaged over 10 runs of each algorithm.

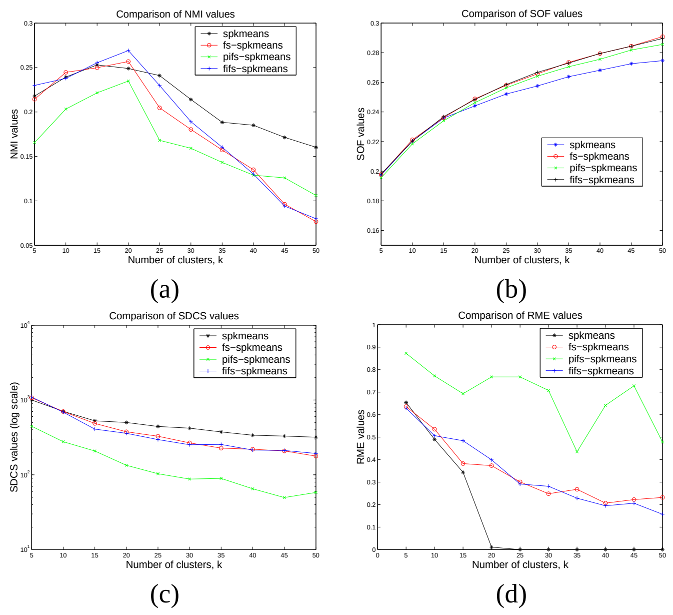


Fig. 2. Comparison between the static algorithms on the News20 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) the standard deviation in cluster sizes, and (e) the ratio of the minimum to expected cluster size values, averaged over 10 runs of each algorithm.

Figs. 2(c),(d), `pifs-spkmeans` has the most bias towards balancing thereby achieving the lowest SDCS and the highest RME values for the entire range of k over which experiments were performed. It is interesting to note that `fs-spkmeans` and `fifs-spkmeans` seems to follow a middle ground in terms of its cluster balancing and quality biases. It is also to be noted that the SOF values for the proposed algorithms are equal or greater than those achieved by `spkmeans`.

3) The Yahoo20 dataset: As mentioned earlier, the Yahoo20 dataset is highly unbalanced according to the labelling it has. Hence, results on this dataset show how the proposed algorithms handle the data when their balancing bias is not going to help.

It is interesting to see that the performance of the algorithms in terms of the NMI is quite similar to what was observed for the News20 dataset. As before `fs-spkmeans` and `fifs-spkmeans` perform very similarly and the NMI values they achieve deteriorate for values of k greater than 20, the correct number of clusters (Fig. 3(a)). `pifs-spkmeans` performs poorly in terms on the NMI because of its high bias towards balancing that does not help in this particular dataset. It also performs slightly worse than the other algorithms in terms of the SOF values (Fig. 3(b)). However, as before, it consistently gives the lowest SDCS (Fig. 3(c)) and highest RME values (Fig. 3(d)). `spkmeans` maintains a reasonable value of the NMI even for large values of k by generating empty clusters. It is interesting to note that due to fact that the natural clusters are not at all balanced, `fs-spkmeans` and `fifs-spkmeans` give quite low values of RME, but never actually give a zero-sized cluster in the range of k over which experiments were performed. Again, these two algorithms seem to have a good balance between the biases and can respond quite well to the underlying nature of the dataset.

Summary of Results

Both `fs-spkmeans` and `fifs-spkmeans` perform admirably when the value of k chosen is in the neighborhood of the number of classes in the data. They are comparable to or superior than `spkmeans` in terms of cluster quality, and superior in terms of balancing. This result is particularly remarkable for the Yahoo20 dataset where the underlying classes have widely varying priors. This is indicative of the beneficial effect of the regularization provided by the soft balancing constraint. However, if k is chosen to be much larger than the number of natural clusters, `spkmeans` has an advantage since it starts generating zero-sized clusters, while the others are now hampered by their proclivity to balance cluster sizes. On the other hand, if balancing is very critical, then `pifs-spkmeans` is the best choice, but it has to compromise to some extent on cluster quality in order to achieve its superior balancing. So the choice of algorithm clearly depends on the nature of the dataset and the clustering goals, but in general, both `fs-spkmeans` and `fifs-spkmeans` are attractive even when balancing is not an objective.

C. *Experiments with the Streaming Algorithm*

The primary problem in experimenting with the streaming algorithm is that there is no well-known benchmark for clustering of high-dimensional, normalized streaming data. So, the experiments with the streaming algorithms were done by artificially “streaming” the public domain static datasets. The data points are presented sequentially to the `sfs-spkmeans` algorithm, repeating the process as many times as necessary

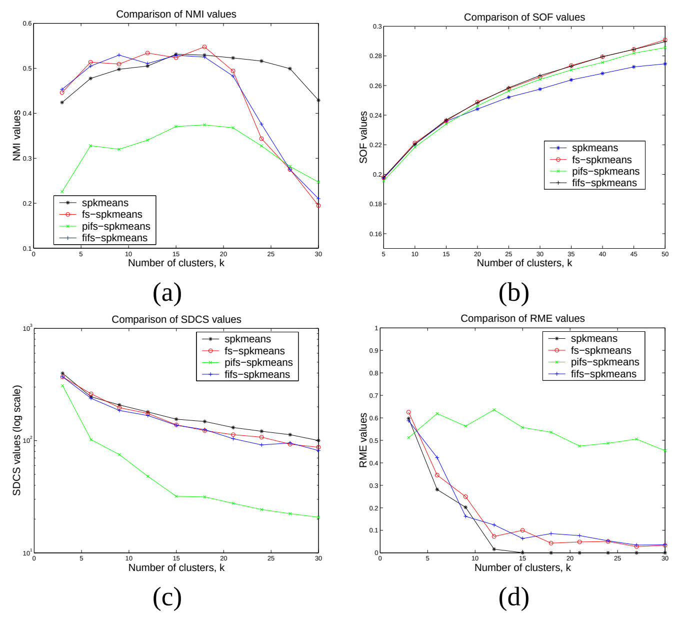


Fig. 3. Comparison between the static algorithms on the Yahoo20 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) standard deviation in cluster sizes, and (e) the ratio of the minimum to expected cluster size values, averaged over 10 runs of each algorithm.

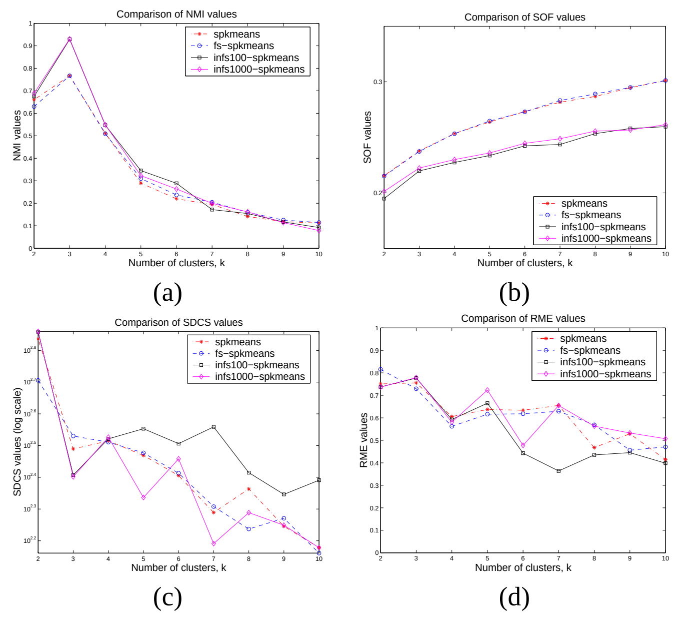


Fig. 4. Comparison between streaming and static algorithms on the Classic3 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) the standard deviation in cluster sizes, and (d) the ratio of minimum to the expected cluster size values, averaged over 10 runs of each algorithm.

in order to simulate streaming data. We call the sequence of showing every document in the selected dataset once as an *epoch* and the algorithm is run over multiple epochs until it converges or some preset maxEpoch value is reached. Note that the resulting scheme is very similar to `fifs-spkmeans` but there are subtle differences. In order to understand the effect of the choice of L, the number to which the norm and the effective cluster sizes of `sfs-spkmeans` converges, we present results corresponding to two choices of L: 100 and 1000, and the corresponding algorithms will be referred to as `sfs100-spkmeans` and `sfs1000-spkmeans` respectively. Note that both these values of L are less than the data set sizes. This means that `sfs-spkmeans` has less effective memory than the static algorithms. In fact, such a low effective memory handicaps the streaming algorithm as compared to the static ones which use all the data to update their parameters. As we shall see, the streaming algorithm actually performs reasonably well even with this handicap.

There are three aspects to be considered when evaluating a streaming algorithm [54]: (i) how quickly does it ramp up to a solution, (ii) the quality of the solution, and (iii) if the data characteristics change, how quickly does the system respond to such changes. Since in this paper, the streaming data sets are obtained by reproducing a fixed data set end-to-end, all the solutions show an asymptotic behavior. So we shall first address aspect (ii) and compare these asymptotic solutions with those obtained by the static algorithms. Then, in the next subsection, we address aspect (i) and look at the learning curves.

1) Asymptotic Results: The streaming algorithm ramps up to a “steady state” solution in a few epochs and after that there are only minor perturbations to this solution. In this section, such steady state solutions are averaged over 10 runs and then compared with spkmeans and fs-spkmeans.

The Classic3 dataset: For the Classic3 dataset, all the algorithms achieve their highest individual values of NMI at $k = 3$, the actual number of clusters in the dataset. The streaming algorithms achieve higher NMI at $k = 3$ compared to the batch algorithms (Fig. 4(a)). All the four algorithms perform very similarly in terms of the NMI for all other values of k. The SOF values achieved by the static algorithms are consistently higher that by the streaming algorithms (Fig. 4(b)). There is no significant difference between the behavior of the algorithms in terms of the two cluster balancing measures (Figs. 4(c),(d)).

The News20 dataset: In the 20 News dataset, the streaming algorithms perform significantly better than the static ones in terms of the NMI (Fig. 5(a)). The primary reason for this is that since the natural clusters in the data are perfectly balanced and the streaming algorithms are biased towards balanced clustering, they get the correct structure in the data due to their bias. Among the streaming algorithms, `infs100-spkmeans` performs marginally better than `infs1000-spkmeans` though the differences are not always significant. The SOF values for the static algorithms are significantly better than those achieved by the streaming algorithms (Fig. 5(b)). There is no significant difference in the SDCS for the various algorithms (Fig. 5(c)). The frequency sensitive algorithms perform better than `spkmeans` in terms of the RME values, and the streaming algorithms give higher values of RME than `fs-spkmeans`(Fig. 5(d)).

The Yahoo20 dataset: In the Yahoo dataset, the static algorithms seem to achieve higher values of NMI than the streaming ones (Fig. 6(a)). In the trade-off between balancing and cluster quality, the streaming algorithms seem to give more importance to the balancing aspect whereas the static ones seems to give higher priority to the cluster quality.

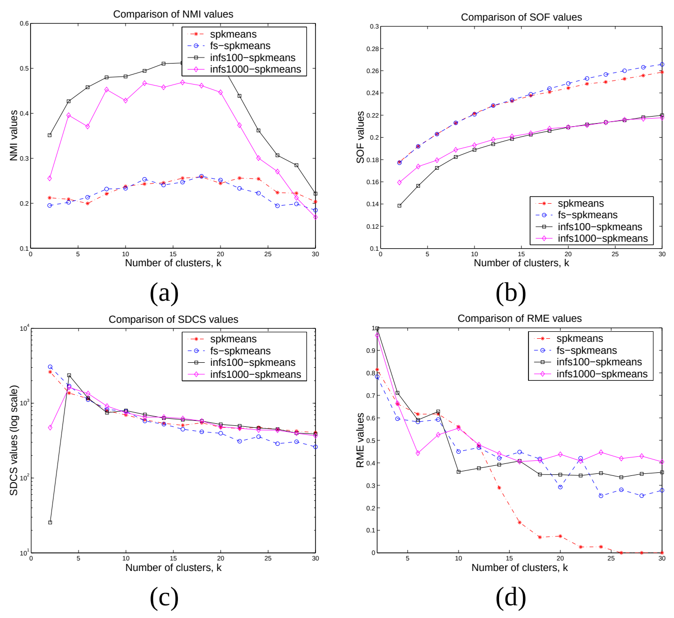


Fig. 5. Comparison between streaming and static algorithms on the News20 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) the standard deviation in cluster sizes, and (d) the ratio of the minimum to the expected cluster size values, averaged over 10 runs of each algorithm.

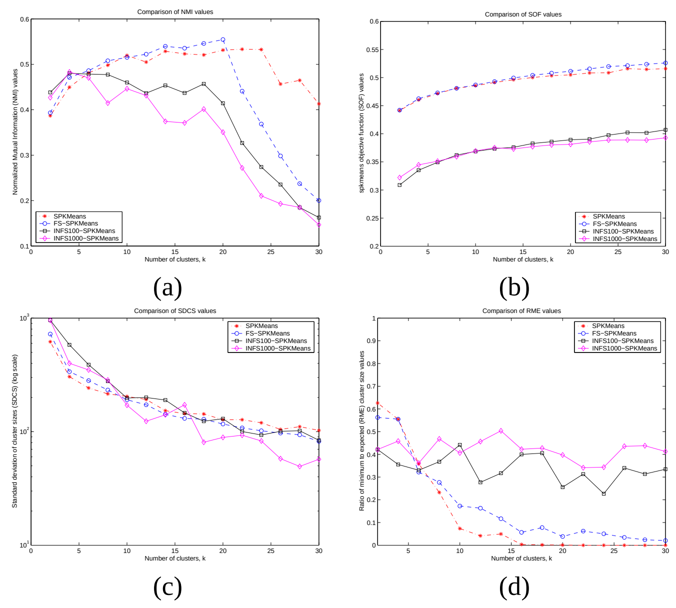


Fig. 6. Comparison between streaming and static algorithms on the Yahoo20 data: (a) the normalized mutual information values, (b) the `spkmeans` objective function values, (c) the standard deviation in cluster sizes, and (d) the ratio of minimum to the expected cluster size values, averaged over 10 runs of each algorithm.

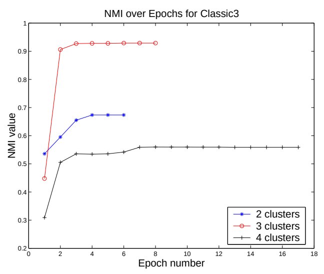
The streaming algorithms being biased towards the balancing criterion, performs poorly in terms of the NMI in this dataset that has highly unbalanced natural clusters. Due to this bias, they give significantly better RME values as compared to the static algorithms (Fig. 6(d)). Like the other two datasets, the SOF values achieved by the static algorithms are significantly better than those by the streaming ones (Fig. 6(b)). Also, just like the other datasets, there is not much difference in the SDCS across all the algorithms (Fig. 6(c)).

2) Learning Curves: To get a better understanding of how quickly `sfs-spkmeans` reaches a steady state solution, we study it closely on three *randomly chosen* runs on the three datasets for different number of clusters. Note that we do not average over multiple runs since then the corresponding epochs of the same run will be lost. The results are presented for both `sfs100-spkmeans`. and `sfs1000-spkmeans` in Figs. 7 and 8 respectively.

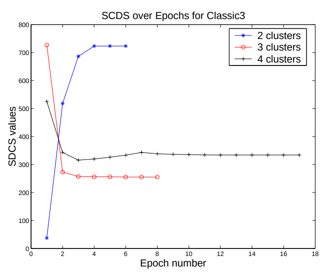
Consider Fig. 7 first. In the Classic3 dataset, the NMI for $k = 3$, the correct number of clusters, shoots up in the 2nd epoch itself and stays at that level from that point onwards till convergence (Fig. 7(a)). This shows that the data set has a very simple structure and a single epoch was sufficient to capture it. For $k = 2$ and $k = 4$, the algorithm does not get any structure in the data as shown by the NMI plots. In the News20 dataset, there is an initial increase in NMI for $k = 10, 20, 30$ (Fig. 7(d)). However, the plot for $k = 30$ plateaus at a much lower value of NMI than that for $k = 10, 20$. It is interesting to note that although the NMI values for $k = 10, 20$ are quite similar in the first few epochs, the algorithm for $k = 20$ eventually finds a better structure in the data – this fact is reflected in the plots as the NMI values for $k = 20$ crosses that for $k = 10$ before convergence. The behavior in the Yahoo dataset is quite similar to that observed for the News20 dataset in terms of the NMI.

In the Classic3 dataset, the SDCS values for $k = 3, 4$ are significantly better than that for $k = 2$ (Fig. 7(b)). A similar pattern is observed for the other two datasets. Note that we are showing results for three values of k for all the datasets – one value $\underline{k}$ is less than the number of natural clusters in the data, one value k^* is (approximately) equal to that, and the third value $\overline{k}$ is greater than that. The general pattern we observe is that the algorithm performs well in terms of the NMI for $k = \underline{k}, k^*$, and performs well in terms of the SDCS for $k = k^*, \overline{k}$. Thus, the values at or around k^* are always in the group that performs well for these measures, whereas the performance for the other values of k suffer in one measure or the other. Also, the RME values for $k = k^*$ are higher or at least as good as that for $k = \underline{k}, \overline{k}$. Thus, we note that the algorithm performs the best for the performance measures under consideration at values close to the natural number of clusters, or, in other words, gets the structure in the data quite well. This is a very desirable quality for a clustering algorithm.

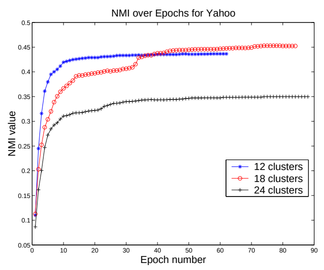
A similar behavior is observed for all the datasets while studying the learning curves generated by `infs1000-spkmeans` in Fig.8. The number of epochs required to converge is normally higher than that for the $L = 100$ case since the effective learning rate, being inversely proportional to L, is much lower in this case. The NMI values for k^* after convergence is higher than the others for the Classic3 and News20 datasets (Figs. 8(a),(d)). For the Yahoo dataset, because of its complicated structure, the preset maxEpoch value of 100 is reached before the algorithms converges – the NMI values for $\underline{k}$ and k^* are still increasing asymptotically and the value for $\underline{k}$ is slightly higher when the epochs are terminated (Fig. 8(g)). As before, the SDCS values for $k^*, \overline{k}$ are better than that for $\underline{k}$ across all datasets.



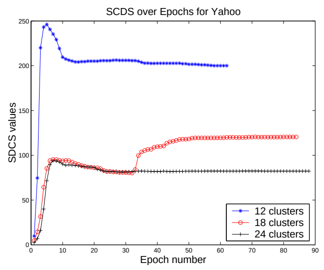
(a)



(b)

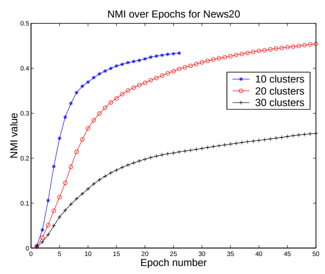


(c)

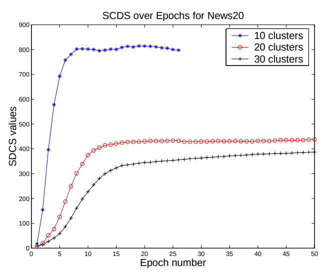


(d)

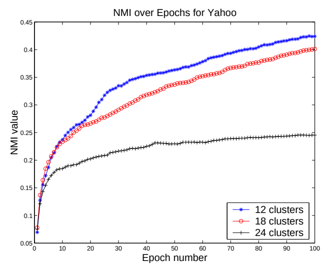
Fig. 7. Comparison of NMI and SDCS values over epochs (`infs100-spkmeans`) on particular runs for the Classic3 and Yahoo20 datasets.



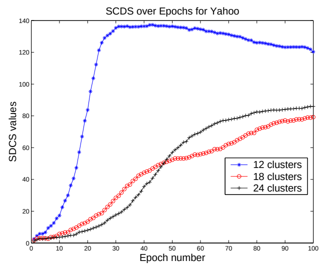
(a)



(b)



(c)



(d)

Fig. 8. Comparison of NMI and SDCS values over epochs (`infs1000-spkmeans`) on particular runs for the News20 and Yahoo20 datasets.

Also, the RME values for $k^*, \underline{k}$ are better than that for $\overline{k}$. Thus, we once again see that the algorithm tends to have the best balance across all the performance metrics for values of k close to the natural number of clusters.

VII. Concluding Remarks

Obtaining a balanced solution is an explicit goal in certain clustering applications irrespective of the underlying structure of the data. More commonly, obtaining clusters of comparable sizes is not a stated objective, but some amount of balancing helps in countering poor initializations in iterative clustering algorithms that converge to only a local optimum. The problem of poor initialization is exacerbated when both the input dimensionality as well as the number of clusters sought are high, thereby vastly expanding the solution space. In addition to incorporating a conscience mechanism to competitive learning, a variety of other approaches have been proposed over the years [55] to overcome poor initializations in iterative clustering algorithms that are otherwise attractive because of simplicity, low computational complexity, etc.

In this paper, we focused on applications where the data is normalized to lie on the surface of a hypersphere. For such datasets, the clustering problem was posed as a maximum likelihood estimation of k vMF distributions that are assumed to have generated the observed data. This generative model can be adapted, if need be, to provide various degrees of balancing. In the process, we derived certain existing, heuristically proposed algorithms (spkmeans, FSCL) from first principles. The empirical results were also encouraging in that they were are both superior (using extrinsic as well as intrinsic criteria) as well as significantly better balanced. This was true even for the highly imbalanced Yahoo20 dataset.

This paper assumes that the vMF distribution is a good model for clustering directional data, and the number of clusters k is known apriori. In general, there may be clustering problems where these assumptions may not hold good. Model selection techniques for choosing an appropriate family as the generative model and choosing the right number of clusters has been extensively studied in the literature. The techniques range from information theoretic criteria[56], [57] for model comparison to purely Bayesian approaches such as reversible jump MCMC [58]. In this paper, we work with the vMF distribution and show empirical justifications for the choice. A possible future work can be a more extensive study of spherical distributions [15] for choosing an appropriate family using model selection methods. Determining a suitable value for k when the desired number of clusters is not known beforehand is a long studied problem with no universally accepted solution [1]. One approach is to incrementally increase the number of clusters as more data is examined. Several competitive learning variants already exist to do this for non-normalized data (see [23] and references cited therein). It may be worthwhile to derive an incremental k algorithm for normalized data as well. Alternatively, one can first obtain solutions for different values of k and then select a suitable one based on an appropriate model selection criterion. Finally, from a practical viewpoint, a third alternative for reaching a suitable k is to partition the data using a conservatively high value of k, and then do a few steps of agglomerative clustering, as is done, for example, in the classic ISODATA algorithm [1].

Acknowledgements

The authors would like to thank Inderjit Dhillon for valuable discussions.

References

[1] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. New Jersey: Prentice Hall, 1988.

[2] J. Ghosh, "Scalable clustering," in *The Handbook of Data Mining*, N. Ye, Ed. Lawrence Erlbaum Assoc., 2003, pp. 247–277.

[3] P. Smyth, "Clustering sequences with Hidden Markov Models," in *Advances in Neural Information Processing Systems-9*, M. J. M.C. Mozer and T. Petsche, Eds., vol. 9. MIT Press, 1997, pp. 648–654.

[4] J. A. Blimes, "A gentle tutorial of the EM algorithm and its application to par estimation for gaussian mixture and hidden markov models," U. C. Berkeley, Tech. Rep., April 1998.

[5] K. Rose, "Deterministic annealing for clustering, compression, classification, regression, and related optimization problems," *Proc. IEEE*, vol. 86, no. 11, pp. 2210–39, 1998.

[6] T. S. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," in *Advances in Neural Information Processing Systems-11*, M. S. Kearns, S. A. Solla, and D. D. Cohn, Eds., vol. 11. MIT Press, 1999, pp. 487–493.

[7] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.

[8] B. Scholkopf and A. Smola, *Learning with Kernels*. MIT Press, 2001.

[9] P. Indyk, "A sublinear-time approximation scheme for clustering in metric spaces," in *Proceedings of the 40th Symposium on Foundations of Computer Science*, 1999, pp. 154–159.

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

[11] A. Strehl and J. Ghosh, "Relationship-based clustering and visualization for high-dimensional data mining," *INFORMS Journal on Computing*, vol. 15, no. 2, Spring 2003. [Online]. Available: http://strehl.com/download/strehl_joc02.pdf

[12] J. H. Friedman, "An overview of predictive learning and function approximation," in *From Statistics to Neural Networks, Proc. NATO/ASI Workshop*, V. Cherkassky, J. Friedman, and H. Wechsler, Eds. Springer Verlag, 1994, pp. 1–61.

[13] K. Chang and J. Ghosh, "A unified model for probabilistic principal surfaces," *IEEE Trans. PAMI*, vol. 23, no. 1, pp. 22–41, Jan 2001.

[14] G. Hinton, P. Dayan, and M. Revow, "Modeling the manifolds of images of handwritten digits," *IEEE Transactions on Neural Networks*, vol. 8, pp. 65–74, Jan 1997.

[15] K. V. Mardia, "Statistics of directional data," *J. Royal Statistical Society. Series B (Methodological)*, vol. 37, no. 3, pp. 349–393, 1975.

[16] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine Learning*, vol. 42, no. 1, pp. 143–175, January 2001.

[17] I. S. Dhillon, J. Fan, and Y. Guan, "Efficient clustering of very large document collections," in *Data Mining for Scientific and Engineering Applications*, V. K. R. Grossman, C. Kamath and R. Namburu, Eds. Kluwer Academic Publishers, 2001.

[18] A. Banerjee and J. Ghosh, "On scaling up balanced clustering algorithms," in *Proc. 2nd SIAM Intl Conf on Data Mining*, April 2002, pp. 333–349.

[19] P. S. Bradley, K. P. Bennett, and A. Demiriz, "Constrained k-means clustering," Microsoft Research, Tech. Rep., May 2000.

[20] S. Grossberg, "Adaptive pattern classification and universal recoding: 1. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121–134, 1976.

[21] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, pp. 75–112, 1985.

[22] S. Grossberg, "Cortical dynamics of three-dimensional form, color and brightness perception, i: Monocular theory," *Perception and Psychophysics*, vol. 41, pp. 87–116, 1987.

[23] Y. J. Zhang and Z. Q. Liu, "Self-splitting competitive learning: A new on-line clustering paradigm," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 369–380, March 2002.

[24] J. C. Bezdek and S. Pal, *Fuzzy Models for Pattern Recognition*. Piscataway, NJ: IEEE Press, 1992.

[25] D. deSieno, "Adding conscience to competitive learning," in *IEEE Annual Int'l. Conf. on Neural Networks*, 1988, pp. 1117–1124.

[26] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, no. 3, pp. 277–290, 1990.

[27] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[28] M. Kearns, Y. Mansour, and A. Y. Ng, "An information-theoretic analysis of hard and soft assignment methods for clustering," in *Proceedings of Uncertainty in Artificial Intelligence*, 1997, pp. 282–293.

[29] S. Basu, A. Banerjee, and R. Mooney, "Semi-supervised clustering by seeding," in *Proc. 19th Intl Conf on Machine Learning*, 2002, pp. 19–26.

[30] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[31] J. Sinkkonen and S. Kaski, "Clustering based on conditional distributions in an auxiliary space," *Neural Computation*, 2001, accepted for publication.

[32] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra, "Clustering on hyperspheres using Expectation Maximization," Department of Computer Sciences, University of Texas, Tech. Rep. TR-03-07, February 2003.

[33] J. N. Kapur and H. K. Kesavan, *Entropy Optimization Principles with Applications*. Academic Press, 1992.

[34] N. W. McLachlan, *Bessel Functions for Engineers*. Oxford University Press, 1955.

[35] A. Banerjee and J. Ghosh, "Frequency sensitive competitive learning for clustering on high-dimensional hyperspheres," in *Proc. IJCNN'02, Honolulu*, May 2002, pp. 1590–1595.

[36] A. S. Galanopoulos and S. C. Ahalt, "Codeword distribution for frequency sensitive competitive learning with one dimensional input data," *IEEE Trans. Neural Networks*, vol. 7, no. 3, pp. 752–756, 1996.

[37] A. S. Galanopoulos, R. L. Moses, and S. C. Ahalt, "Diffusion approximation of frequency sensitive competitive learning," *IEEE Trans. Neural Networks*, vol. 8, no. 5, pp. 1026–1030, Sept 1997.

[38] V. Ramamurti and J. Ghosh, "On the use of localized gating in mixtures of experts networks," in *(invited paper), SPIE Conf. on Applications and Science of Computational Intelligence, SPIE Proc. Vol. 3390*, Orlando, Fl., April 1998, pp. 24–35.

[39] N. Gershenfeld and A. Weigend, "The future of time series: Learning and understanding," in *Time Series Prediction*, A. S. Weigend and N. Gershenfeld, Eds. Addison Wesley, 1993, pp. 243–264.

[40] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: Perceptron, Madaline and Backpropagation," *Proc. IEEE*, vol. 78, no. 9, pp. 1415–1442, Sept 1990.

[41] M. C. Mozer, "Neural network architectures for temporal sequence processing," in *Time Series Prediction*, A. S. Weigend and N. Gershenfeld, Eds. Addison Wesley, 1993, pp. 243–264.

[42] J.-P. C. S. Dehaene and J.-P. Nadal, "Neural networks that learn temporal sequences by selection," *Proc. National Academy of Sciences, USA*, vol. 84, pp. 2727–2731, May 1987.

[43] B. W. Stiles and J. Ghosh, "A habituation based neural network for spatio-temporal classification," *Neurocomputing*, vol. 15, pp. 273–303, July 1997.

[44] P. Turney, "The management of context-sensitive features: A review of strategies," in *Proceedings of the ICML-96 Workshop on Learning in Context-Sensitive Domains*, Bari, Italy, July 1996. [Online]. Available: http://ai.iit.nrc.ca/bibliographies/context-sensitive.html

[45] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA: ACM Press, 2001, pp. 97–106. [Online]. Available: citeseer.nj.nec.com/hulten01mining.html

[46] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *In Proc. of the Annual Symp. on Foundations of Computer Science (FOCS 2000)*, November 2000.

[47] G. K. Gupta and J. Ghosh, "Detecting seasonal trends and cluster motion visualization for very high dimensional transactional data," in *Proc. First Siam Conf. On Data Mining, (SDM2001)*, 2001, pp. 115–129.

[48] H. G. C. Tråvén, "A neural network approach to statistical pattern classification by "semiparametric" estimation of probability density functions," *IEEE Transactions on Neural Networks*, vol. 2, no. 3, pp. 366–377, 1991.

[49] V. Ramamurti and J. Ghosh, "Structurally adaptive modular networks for non-stationary environments," *IEEE Transactions on Neural Networks*, vol. 10, no. 1, pp. 152–60, 1999.

[50] R. M. Neal and G. E. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*, M. I. Jordan, Ed. MIT Press, 1998, pp. 355–368.

[51] J. C. Principe, J.-M. Kuo, and S. Celebi, "An analysis of the gamma memory in dynamic neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 331–337, March 1994.

[52] A. Strehl, J. Ghosh, and R. Mooney, "Impact of similarity measures on web-page clustering," in *Proc. 17th National Conference on Artificial Intelligence : Workshop of Artificial Intelligence for Web Search (AAAI 2000), Austin, Texas, USA*. AAAI, July 2000, pp. 58–64.

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

[54] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1985.

[55] M. Meila and D. Heckerman, "An experimental comparison of several clustering and initialization methods," Microsoft Research, Tech. Rep. MSR-TR-98-06, 1998.

[56] H. Akaike, “A new look at statistical model identification,” *IEEE Transactions on Automatic Control*, vol. AU-19, pp. 716–722, 1974.

[57] G. Schwatz, “Estimating the dimension of a model,” *The Annals of Statistics*, vol. 6, no. 2, 1978.

[58] P. J. Green, “Reversible jump markov chain monte carlo computation and bayesian model determination,” *Biometrika*, vol. 82, pp. 711–732, 1995.

PLACE
PHOTO
HERE

Arindam Banerjee received the B. Engg. degree in Electronics and Tele-communication Engineering from Jadavpur University, India, in 1997 and the M. Tech. degree from the IIT, Kanpur, India, in 1999. He is currently pursuing his Ph.D. in the Computer Engineering area of the Department of Electrical and Computer Engineering at the University of Texas at Austin.

His research interests include data mining, machine learning, pattern recognition, semi-supervised learning, information geometry, convex analysis, statistical learning theory, and probability theory.

PLACE
PHOTO
HERE

Joydeep Ghosh was educated at IIT Kanpur (B. Tech ’83) and The University of Southern California (MS, Ph.D, ’88). Subsequently he joined the the Department of Electrical and Computer Engineering at the University of Texas, Austin, where he is a Full Professor since 1998, and holder of the Archie Straiton Endowed Fellowship. Dr. Ghosh directs the Laboratory for Artificial Neural Systems (LANS), where his research group is studying the theory and applications of adaptive pattern recognition, data mining including web mining, and multi-learner systems.

Dr. Ghosh has published more than 200 refereed papers and edited 10 books. He has received six best paper awards, including the 1992 Darlington Prize for the Best Journal Paper from IEEE Circuits and Systems Society, and the Best Applications Paper at ANNIE’97.

Dr. Ghosh served as Conference Co-Chair of Artificial Neural Networks in Engineering (ANNIE)’93 -ANNIE’96, ANNIE ’98-2003, and in the program or organizing committees of several conferences on data mining and neural networks each year. More recently, he co-organized workshops on Web Mining (with SIAM Int’l Conf. on Data Mining, 2001 and 2002) and on Parallel and Distributed Data Mining (with KDD-2000). He was a plenary speaker for ANNIE’97, MCS2002 and letters editor for IEEE Trans. Neural Networks (1998-2000). He is currently an associate editor of Pattern Recognition, Neural Computing Surveys and Int’l Jl. of Smart Engineering Design.