ON DETERMINING THE NUMBER OF CLUSTERS–A COMPARATIVE STUDY

By

Brian Bies

Kathryn Dabbs

and

Hao Zou

IMA Preprint Series # 2272

(August 2009)

INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS

UNIVERSITY OF MINNESOTA
400 Lind Hall
207 Church Street S.E.
Minneapolis, Minnesota 55455–0436

Phone: 612/624-6066 Fax: 612/626-7370
URL: http://www.ima.umn.edu

On Determining The Number Of Clusters–A Comparative Study

Brian Bies[1], Kathryn Dabbs[2], Hao Zou[3]

Abstract

In this paper, we perform one of the first empirical tests comparing several existing algorithms for determining the number of clusters in a data set (the gap statistic, X-means, G-means, data spectroscopic clustering and self-tuning spectral clustering). We use a large number of data sets randomly generated with varying distributions and parameters. The results show that the G-means and X-means perform best on the majority of test cases. We also explore ways to improve the gap statistic, and formulate the problem in the simplified continuous context to consider its theoretical basis.

1 Introduction

The problem of clustering commonly appears in the analysis of data. Data clustering refers to the process of assigning a set of observations to groups, so that observations in the same group are close or similar to each other. It is an important technique for statistical data analysis used in many different fields.

Formally, we consider $X = \{x_1, x_2, ...x_n\} \subseteq R^d$. We want to find k^* disjoint clusters, $\{C_1, C_2, ...C_{k^*}\}$ and $\{C_l \subseteq X | l \in \mathbb{N} \cap [1, k^*]\}$ which partition X. A number of clustering methods and criteria exist that assign each data point to its corresponding cluster. For example, the widely used k-means algorithm assigns each point to the cluster whose centroid is closest to that point. Let $n_l = |C_l|$. The k-means algorithm seeks to minimize the within-cluster sum of squares:

$$D(X) = \sum_{l=1}^{k^*} \sum_{x_i \in C_l} ||x_i - \mu_l||^2$$

where μ_l are cluster centers,

$$\mu_l = \frac{1}{n_l} \sum_{x_i \in C_k} x_i \, \text{for} \, l = 1, 2, ...k^* \, .$$

This is an NP-hard problem [12], but a number of heuristics have found ways to identify the clusters with reasonable accuracy in polynomial time.

Considered independently of these clustering algorithms, however, is the problem of finding the number of clusters k^* in a data set. Most clustering methods require the number of clusters k^* as an input, but finding the correct k^* is not easy, underlined by the fact that no universally-accepted definition of a "cluster" exists. Usually the choice depends on the characteristics of the data set and the desired resolution of the user. Furthermore, $D(X)$ is a monotonically decreasing function with respect to k, so naturally we will choose the number of clusters that strikes a balance between optimal compression and optimal accuracy of the clustering.

In this paper, we focus on several of the proposed methods of finding the number of clusters in a data set. These include the gap statistic [19], X-means [14], G-means [5], data spectroscopic clustering [16], and self-tuning spectral clustering [23]. A brief description of these methods can be found in Section 3.1. To compare these methods we test them using large samples of randomly generated data sets. We compare the accuracy of these methods under different circumstances, and suggest possible explanations to account for their performances. For the theoretical discussion, motivated by the observation that the gap statistic is computationally expensive, we aim to simplify the gap statistic, and try to make it more efficient. We also investigate the problem of clustering under the continuous case, to gain some insight into theoretical basis for the gap statistic algorithm.

[1] Washington University in St. Louis, St. Louis, MO, bwb4@cec.wustl.edu
[2] University of Tennessee, Knoxville, TN, kdabbs1@utk.edu
[3] Macalester College, St. Paul, MN, hzou@macalester.edu

This paper contributes to the literature in the following ways: it is one of the first studies that empirically compares various algorithms on finding the number of clusters; it suggests explanations based on the performances of the algorithms; finally, it proposes modifications to existing methods and possible directions for future research.

1.1 Related Work

We are primarily interested in determining the number of clusters in a data set. Given that much more literature exists concerning the general clustering problem, however, we will focus mainly on interpreting clustering results with respect to what they imply concerning finding the number of clusters.

Recently researchers have been concerned with closing the gap that exists between the practical applications of clustering algorithms and the theory of clustering a data set. Kleinberg [8] attempts to develop a general framework for clustering theory by proposing several axioms. He then proves that there does not exist a clustering function for which all three of these axioms hold, and as a results shows that determining the number of clusters in a data set based on these axioms is an poorly-defined problem.

However, relaxations of these axioms, proposed by Ackerman and Ben-David [2], allow the problem of determining the number of clusters to be well-defined. Similarly, in an earlier paper [1], Ackerman and Ben-David study clusterability, or the amount of clustered structure present in a given data set, and determine that several measures of clusterability are not equivalent. As a consequence, we can again see that the number of clusters in a data set may vary widely depending on one's notions of clusterability.

In addition to developing a general theory of data clustering, many researchers focus on theory applied to specific clustering algorithms. Perhaps one of the most widely used clustering algorithms is k-means. Ostrovsky, et al. [12] deal specifically with variations of Lloyd's method for k-means. This method "seeds" the algorithm with some initial centers, and is highly sensitive to the choice of initial centers. The authors introduce an efficient, probabilistic process for seeding Lloyd's method. Using this process, it is shown that if the data is well-clusterable by some clusterability condidtion, then various Lloyd-type methods return near optimal results.

Also concerned with the K-means problem, Meila [11] investigates the stability of optimal clusterings. Although K-means is NP-hard, when the dataset is well-clusterable, it is easy to find a near-optimal solution. Hence, it is easy to find the number of clusters in a well-clusterable data set. Several algorithms for estimating the number of clusters use a variant of k-means. Because of this, any theoretical or practical advances in this area may also be applicable to the problem of determining the number of clusters. In addition to these general clustering results, several papers outline specific methods for determining the number of clusters. These are discussion in Section 2.1.

The rest of the paper is divided as follows: Section 2 contains an empirical evaluation of the five methods mentioned above; the section is further divided into 3 subsections: overview of algorithms, data generation, and results comparison. Section 3 contains theoretical discussions about the Gap Statistic, as well as possible improvements to the algorithm; Section 4 summarizes our results and suggests directions for future research. A derivation of theoretical results and summary of the experimental data can be found in the Appendices.

2 Experimental Procedures and Results

In this section, we will provide an overview of the five methods compared in our study, and then present empirical evaluations of the results. All these methods return the number of clusters estimated as an output.

2.1 Overview of the Algorithms

X-means (Pelleg and Moore) [14]

We use the X-means algorithm described and implemented by Pelleg and Moore [14]. This algorithm begins with one cluster, and continues to add clusters until the centers selected appear to model the data "well enough". More formally, the algorithm consists of:

1. Given the initial clusters, run k-means on the entire data set until convergence.

2. Split each resulting cluster center into two children, and run a local k-means on each cluster and its pair of children.

3. For each cluster, compare the Bayesian Information Criterion (BIC) score[15] of the parent cluster to the combined BIC score of the two child clusters, and keep whichever group has a higher score (the group that appears to model the data better).

4. Go back to step 1.

The k-means algorithm used in X-means is actually the accelerated k-means algorithm developed by Pelleg and Moore[13]. This algorithm has identical output to k-means, but uses a kd-tree[4] to improve the algorithm's speed in low numbers of dimensions. Statistics about the points contained in its bounding hyper-rectangle are stored at each node, which allows the algorithm to run k-means iterations faster when there are large volumes that are entirely closer to one center than any others. In addition, the X-means algorithm keeps a "blacklist" of clusters that do not split into children or lose points to other clusters. In this way, X-means can decrease the number of splits it needs to perform on each iteration.

X-means takes as input a data set $X = \{x_1, ..., x_n\} \subset R^d$ and the number of iterations (splits) to be performed. X-means then returns the best model found after completing all iterations. We use 50 iterations for all trials (which could potentially return 2^{49} cluster centers, since the number of centers may double every iteration if all split).

G-means (Hamerly and Elkan) [5]

We use the G-means algorithms described and implemented by Hamerly and Elkan[5]. This algorithm is quite similar to X-means in that it begins with a single cluster, then iteratively splits centers and runs k-means to convergence until it finds the "correct" number of clusters. The algorithm proceeds as follows:

1. Given the initial centers, run k-means on the entire data set to convergence.

2. For each resulting cluster, test whether the data appear to come from a Gaussian distribution using the Anderson-Darling statistic[3].

3. If all clusters appear Gaussian, return k, the number of clusters in the final iteration.

4. Otherwise, split every cluster that does not appear Gaussian into two child clusters, and go back to step 1.

G-means uses the Anderson-Darling statistic to test if clusters appear Gaussian. Because this test is one-dimensional, each cluster is tested as follows:

1. Split the parent into two children, along the direction of the first principal component of the cluster.

2. Run k-means for the two child centers on the single cluster's data.

3. Project the data onto the resulting vector separating the two children (i.e. $v = c_1 - c_2$, where c_1, c_2 are the converged child centers).

4. Use the Anderson-Darling statistic to check the projected one-dimensional data.

Note that each iteration involves only one splitting of clusters. First, the parents clusters are split along their first principal component, and k-means is run in each individual cluster. Then the clusters are tested to see if they fit a Gaussian distribution, and finally k-means is run again, globally, on the combined new and old cluster centers.

G-means takes as input the a data set $X = \{x_1, ..., x_n\} \subset R^d$ and the confidence level, α, desired to be used for each statistical test. We used the default value of $\alpha = 0.001$ for all trials.

Data Spectroscopic Clustering (Shi, Belkin, and Yu) [16]

This algorithm takes as input a data set $X = \{x_1, ..., x_n\} \subset R^d$. From this data an $n \times n$ affinity matrix A is constructed where each entry is a function of the Euclidean distance between two data points. The eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ and eigenvectors $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n$ of the affinity matrix A are then calculated. The number of clusters is then estimated by the number of eigenvectors $\mathbf{e}_j$ which have no sign changes up to precision ϵ_j. A vector $\mathbf{e} = (e_1, e_2, ..., e_n)$ has no sign changes up to precision ϵ if either $\forall i \quad e_i > -\epsilon$ or $\forall i \quad e_i < \epsilon$.

Self-tuning Spectral Clustering (Zelnik-Manor and Perona) [23]

This algorithm takes as input a data set $X = \{x_1, ..., x_n\} \subset R^d$ and a range of possible numbers of clusters $k = 1, 2, ..., K$. For all tests we use $K = 25$. For each data point x_i the local scale is calculated. The locally scaled affinity matrix A is constructed where each entry is a function of the Euclidean distance between two data points and the scale computed for each of the two data points. The eigenvectors corresponding to the K largest eigenvalues are then computed and form the matrix $E = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_K]$. Using the incremental gradient descent scheme, the rotation R which best aligns the columns of E with the canonical coordinate system is recovered, and the cost of the alignment is calculated for each possible number of clusters $k = 1, 2, ..., K$. The number of clusters is then estimated to be the largest cluster number with minimal alignment cost.

The Gap Statistic [19]

Given a data set $X = \{x_1, ..., x_n\} \subset R^d$, the pairwise squared Euclidean distance between x_i and x_j, $1 \leq i, j \leq n$, is $d_{ij} = \|x_i - x_j\|^2$. If we have clustered the data into k clusters $C_1, C_2, ...C_k$, let $D_l = \sum_{x_i, x_j \in C_l} d_{ij}$ be the sum of the pairwise squared distances for all points in cluster C_l. The gap statistic looks at W_k, the within-cluster dispersion which is considered to be an error measure of clustering. W_k is defined as $W_k = \sum_{l=1}^{k} \frac{1}{2n_l} D_l$. A typical plot of W_k against the number of clusters is shown in Figure 1:

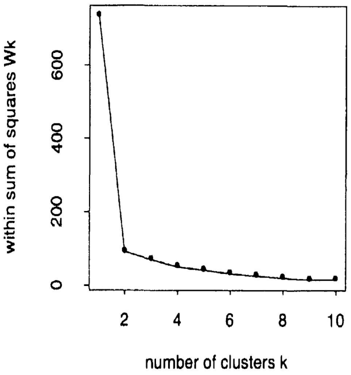


Figure 1: A typical plot of the within-cluster dispersion against the number of clusters [19]

It is easy to see that W_k is a decreasing function of k. The rate of decrease peaks at some "elbow point", after which the decrease flattens markedly. Common heuristics involve trying to find this "elbow point", which would be the appropriate number of clusters in the data set. The gap statistic takes as input the data set, a sequence of numbers of clusters we would like to use, and the number of uniform reference distributions to generate. The method then proceeds as follows:

- Cluster the data (the gap statistic is applicable to any partitioning algorithm [19], and we use "partition around medoids" (PAM) [7] in our test), for all number of clusters from $k = 1, 2, ..., K$. The upper bound K is set to be larger than or equal to k^*, the true number of clusters in the data set. To save computation time, we use the true number of clusters as the upper bound throughout our tests.
- Compute the values W_k for each value of k.
- Uniformly generate B reference data sets over the range of the observed data. Find the corresponding within-cluster dispersion W_k^*.
- Compute the gap statistic using

$$Gap(k) = (1/B) \sum_{b=1}^{B} \log(W_k^*) - \log(W_k).$$

- Of the B copies of reference data sets, calculate the standard deviation of $\log(W_k^*)$. To account for simulation error, use $s_k = sd_k\sqrt{1 + 1/B}$ instead of sd_k in next step.
- The optimal number of clusters is found by the smallest k such that $Gap(k) \geq Gap(k + 1) + s_{k+1}$.

In short, the uniform reference distribution enables us to compute the expected value of $\log(W_k^*)$ rather easily.

2.2 Data Generation

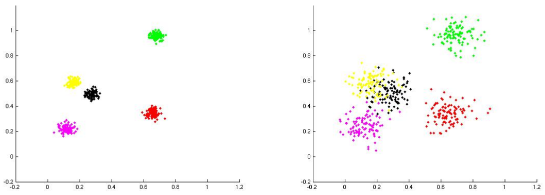
We generated our test data sets to attempt to measure the algorithms' change in performance as data characteristics were varied. To begin with, we considered the effect of independently varying both the number of clusters, k, and the number of dimensions, d. In the process, the relationship between the changing parameters was carefully considered, so that the true cause of variation in the results would not be obscured.

For example, suppose we want to generate a sample file with $k = 5$ clusters in $d = 2, 5,$ and 10 dimensions. The simple process of choosing cluster centers uniformly randomly in a unit box, and then generating points with spherical Gaussians of variance $\sigma < 1$ centered about each center, may seem good enough at first glance. However, for increasing d, each cluster will be expected to occupy less of the total space. As a result, when tested on these three sample files, algorithms may appear to do better as the number of dimensions increase, because the clusters appear to be more "spread out" in higher dimensions.

This intuitive idea of being "spread out" can be formalized by choosing some measurement for the density of a group of clusters. In this paper we only apply our definition to collections of spherical Gaussian distributions of equal variance, though it could certainly be extended to other distributions. A good first choice for a density measurement is simply the average number of clusters centers per unit volume:

$$\text{Density}(C) = k/\,\text{Volume}(C)$$

Where we can choose Volume(C) to be the volume of the unit hypercube. If we let the variance of the clusters increase, however, we see that two clusterings of equal Density$(\cdot)$ may not appear equally "spread out" (see Figure 2).



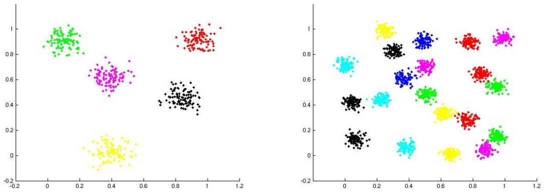
(a) $\sigma^2 = 0.0005$ (b) $\sigma^2 = 0.005$

Figure 2: Even though $k = 5$, $d = 2$ and the number of data points is the same for both pictures, the data on left, with a lower variance, look much more "clusterable" than the data on the right.

We must pick our density measurement, then, so that it takes into account the variance of the clusters as well. We choose the average number of cluster centers per ball of radius $m\sigma$, where σ is the standard deviation of the clusters, and m is a user-supplied value. Since we cannot calculate the density of the clusters before generating them, we will satisfy ourselves with simply choosing the parameters such that the expected number of cluster centers per ball of radius $m\sigma$ is equal to 1. In this way, all samples generated will have an equal expected density of 1. It is a simple probability exercise to show that this gives us a relationship between the variables of the form:

$$\sigma = \frac{1}{m\sqrt{\pi}}\sqrt[d]{\frac{(\frac{d}{2})!}{k}}$$

So, if we would like to vary the number of clusters and dimensions while keeping a constant density throughout the clusterings, we must choose the standard deviation as given above. While m can be increased or decreased according to the density measure desired, throughout our experiments we choose $m = 4.5$, a value large enough to ensure that the clusters are quite separated on average. Example data sets for $d = 2$, $k = 5, 20$ are shown in Figure 3.



(a) 5 clusters

(b) 20 clusters

Figure 3: By using the equation for standard deviation given above, we can keep the expected density of the data sets equal even while we vary k and d.

We also compared the algorithms on spherical uniform distributions. Again, we generated the cluster centers uniformly randomly in a unit box, and we chose a radius for the distribution:

$$R = \frac{1}{2k}$$

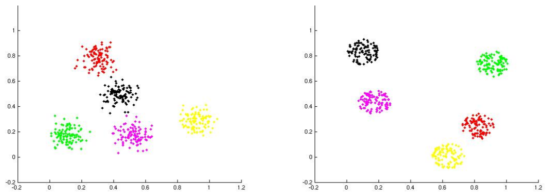
To generate points uniformly randomly inside a sphere of radius R about the centers, we chose a random vector from a spherical normal distribution, scaled it to length 1 (which gives us a uniform random point on the surface of a sphere [10]), and randomly chose a new radius with probability:

$$p(r) = \frac{1}{R^d} dr^{d-1}, 0 < r < R$$

We divided our data sets into two groups: one in which a well-separated condition was rigorously enforced for each data set, and another in which this separation condition was not enforced. For both uniform and Gaussian distributions, this allowed us to test if significant overlap between clusters was a contributing factor to the performance of the algorithms.

- For the Gaussian distributions, the cluster centers were chosen with the condition that they must be at least 4.5σ apart, where σ is the standard deviation of each Gaussian random variable.
- For the uniform distributions, the centers must be at least $2.5R$ apart, where R is the radius of the ball in which points are uniformly generated.

Because the Gaussian distributions extend indefinitely, whereas the uniform distributions have a fixed boundary, the separation conditions are not identical for both distributions. For the uniform distribution, clusters more than $2.5R$ apart are guaranteed to have at least $0.5R$ distance between their closest points. For the Gaussian distribution, clusters more than 4.5σ apart have a low probability of overlapping (having a point from one cluster be closer to another center than its true center), but it is still possible that centers may overlap in a few of the test cases. The separation conditions were chosen to try to equalize this constraint among both of the distributions, however. Example data sets for both distributions with $d = 2$, $k = 5$ and the separation conditions enforced are shown in Figure 4.



(a) Gaussian Distribution (b) Uniform Distribution

Figure 4: Test data sets for $k = 5$ for both distributions with the separation conditions enforced. The separation does not appear exactly equivalent, but comparable.

Finally, we tested the effect of uniform cluster sizes against widely varying cluster sizes. For half of the data sets, we used a fixed number of points (100) generated per cluster, while for the other half, the number of points generated for each cluster was a uniform random integer on the interval [10, 500].

The different combinations of test cases are summarized in Table 1. Fifty sample test files were generated for each configuration.

Parameter	Range Values
k	2, 5, 10, 15, 20
d	2, 10, 20, 30
Type of distribution	Gaussian, Uniform
Separation Condition	Enforced, Not Enforced
Number of Points per Cluster	Fixed at 100, Random in [10, 500]

Table 1: Summary of test data sets.

2.3 Empirical Results

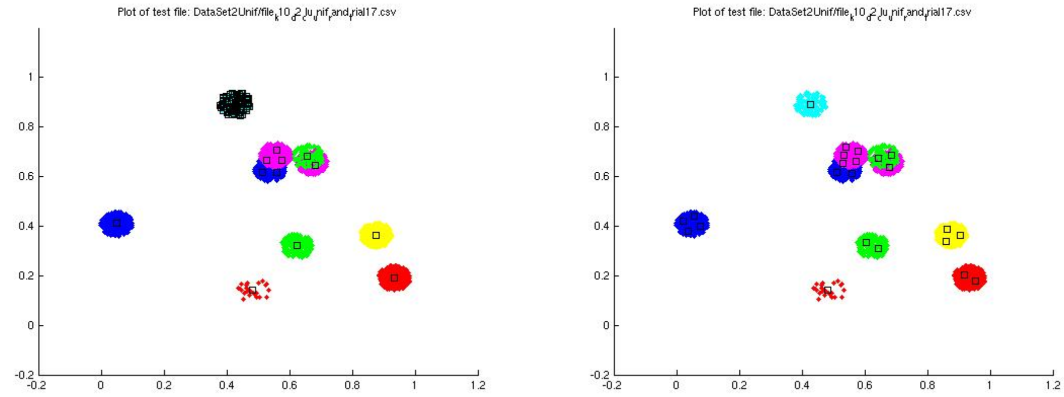
A complete summary of the collected output is given in Appendix 6. Output is normalized by the *correctness ratio*, which is calculated as $\hat{k}/k^*$, where $\hat{k}$ is the number of clusters returned by the algorithm, and k^* is the true number of clusters for the data set. Plots are given for each configuration of the binary conditions (separation condition enforced/not enforced, data distributed normally/uniformly, points per cluster fixed/random), and for each algorithm. Selected plots are reproduced in this section to ephasize important trends.

X-means

The first noticable thing from looking at the X-means plots shown in Appendix 6 is the algorithm's remarkably regular correctness. Aside from a few small humps in the uniform distributions, the returned number of clusters is always very close to the true number. Upon inspection of the actual clusters returned, it becomes apparent that these humps are due to a few extraordinarily far off answers pulling the mean up. For example, for one of the configurations with $d = 20$, $k = 20$, X-means returns the correct answer, 20, for 49 of the test files, but returns 447 for a single case. This results in an average of 28.54 clusters returned, giving an average ratio of 1.427. All noticable humps in more than two dimensions are due to rare cases that are incorrect by an order of magnitude or more influencing the overall mean.

The cause of this is likely as follows: X-means attempts to determine the true number of clusters by performing the number of iterations desired (we used 50 for all cases), and then returning the number of clusters in whichever clustering maximized the BIC score over all iterations. Because all cluster centers are split each iteration, certain configurations of the points in the clusters can lead to runaway growth in the number of cluster centers in a given cluster. An example of this problem in two dimensions with $k = 10$ is shown in Figure 5(a). We see that X-means does quite well even for the overlapping clusters, but that one cluster has 69 cluster centers! The humps in the plots for uniform data seem to indicate that this extreme overestimation is more common in uniform data, possibly because the BIC statistic used assumes normality of the underlying distribution.

The zoomed-in plots show little noticable pattern, except for a slight dip in the ratio when the separation condition is not enforced (this is expected due to the possibility of clusters overlapping), and a tendency for the algorithm to return better results as the number of dimensions increases.



(a) X-means Results

(b) G-means Results

Figure 5: Location of centers output by X-means and G-means for $k = 10$, $d = 2$, the separation condition not enforced, with a uniform distribution, and a random number of points per cluster. Boxes represent the location of cluster centers returned by the algorithms.

G-means

Like X-means, G-means exhibits relatively good performance when compared to the other algorithms. On uniform data with a random number of points per cluster, however, a slight bow-curve is noticable in the ratio plot as the number of dimensions increases.

The zoomed-in plots in Appendix 6 emphasize this curved shape. For the majority of our test sets, the best results are obtained with 10-dimensional data. In addition, when compared to other algorithms, the G-means results appear quite regular across the different numbers of clusters. This may be due to the fact that G-means appears to overestimate the number of centers in each individual cluster with some proportion, as described below.

A plot of G-means output is shown above in Figure 5(b) next to the X-means output for the same test set. We see that, whereas X-means does quite well on most of the clusters, but extremely overestimates on a single cluster, G-means quite regularly overestimates the number of cluster centers in a given cluster. This may be due to two things:

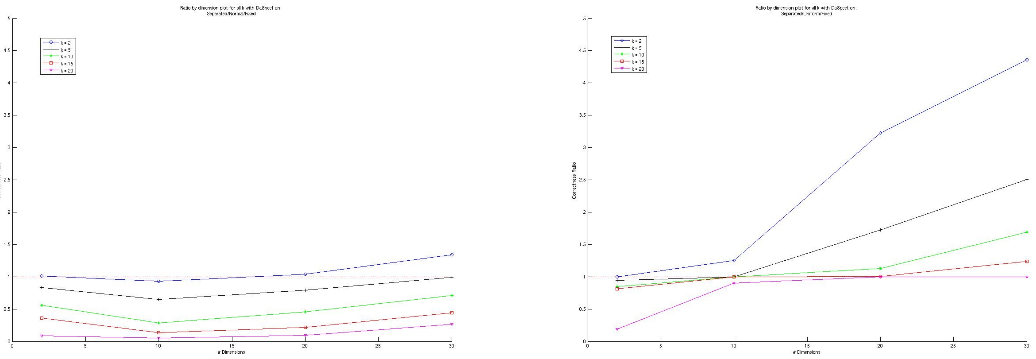
- The Anderson-Darling statistic used in G-means specifically tests for normality of the underlying distribution. Certain configurations of uniformly distributed data may appear more "normal" when split into several groups.
- As mentioned above, G-means takes as input a significance level, α, to be used in the Anderson-Darling test. We used the default parameter of $\alpha = 0.001$, which may be too lax, especially on uniform data.

The fact that increasing the number of clusters has no effect on algorithm's results indicates that G-means may find all the true clusters quite well, but then fail by overestimating the number of centers in each cluster with some probability. This probability seems to be dependent on both the significance level chosen and the underlying distribution of the cluster. If this is true, we would then expect the same proportion of clusters to be overestimated, regardless of the true number of clusters.

In addition, G-means tends to overestimate the number of clusters more when the points per cluster are randomly determined. If clusters are overestimated with some proportion, this may indicate that the proportion is also dependent on the size of the clusters. That is, G-means may be more likely to overestimate the number of centers in clusters with a larger number of points.

Data Spectroscopic Clustering

The data spectroscopic clustering algorithm exhibits the most noticeable difference in results over all the algorithms tested when varying the distribution.



(a) Normal Distribution

(b) Uniform Distribution

Figure 6: Performance of Data Spectroscopic Clustering on different distributions.

Other tests using either the normal distribution or the uniform distribution give similar results to those shown above in Figure 6, implying that the algorithm is not as sensitive to the enforcement of the separation condition or the number of points per cluster as to the underlying distribution of the data set. In tests using the normal distribution, the algorithm tends to underestimate the number of clusters, more severely as the number of clusters increases, but is relatively consistent over different dimensions. In tests using the uniform distribution, however, the algorithm tends to overestimate the number of clusters, more severely as the number of dimensions increases and the number of clusters decreases.

Self-Tuning Spectral Clustering

The self-tuning spectral clustering algorithm tends to underestimate the number of clusters, except in the case of two clusters. This could be a result of the input values chosen for the possible number of clusters ($k = 1, 2, ..., 25$ in all tests). The upper bound $K = 25$ is significantly larger in proportion when $k^* = 2$ than when $k^* = 20$, and the algorithm has more room to underestimate as the true number of clusters increases. In addition, for low dimensions, not enforcing the separation condition causes more variance in the results over different numbers of clusters. For example, spectral-tuning clustering underestimates the number of clusters for $k* = 5, 10, 15, 20$ but severely overestimates the number of clusters for data sets with two clusters, as shown in Figure 7. The underlying distribution and the number of points per cluster, on the other hand, cause no significant difference in the results.

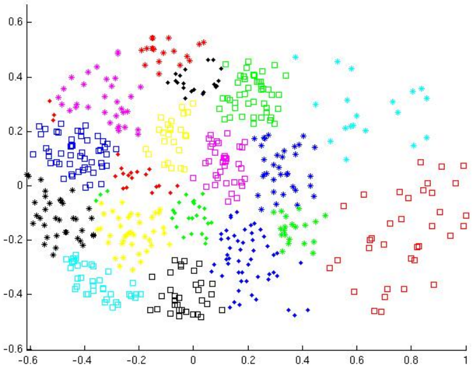
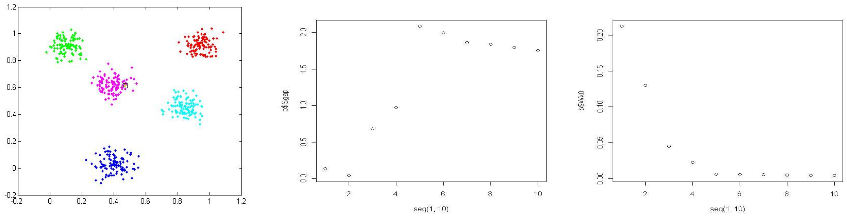


Figure 7: Clustering returned by Self-Tuning Spectral Clustering for $k = 2$ and $d = 2$, no separation condition, uniform distribution, and a random number of points per cluster. Different colors or symbols correspond to data points in different clusters.

The Gap Statistic

The Gap Statistic was the slowest of the algorithms we compared, spending upwards of six hours on data sets in with 20 clusters in 30 dimensions. As a result, we were unable to obtain results for $d = 20, 30$ and $k^* = 15, 20$. The version of the gap statistic code we have makes use of the PAM algorithm in the data partitioning process, which is slower and more robust than the simpler k-means algorithm. Furthermore, k-means output is nondeterministic, and highly dependent on the randomly generated initial centers. Because of this, we used the PAM method throughout our experiments. The plots in Appendix 6 show that the algorithm underestimates the number of clusters more as k^*, the true number of clusters in the data set, increases. This can probably be explained by the input parameters, similar to the self-tuning spectral clustering above. When k^* is small, the output can only be chosen from a small selection of numbers, namely $1, 2, ..., k^*$, so there is little room for underestimation. The underestimation is less pronounced when d increases.

An example test case where the gap statistic underestimates the number of clusters is shown in Figure 8(a) below. In this particular data set, $k^* = 5$ and $d = 2$. It is clear from the plot of the data that there should be five clusters in the data set.



(a) Data Plot

(b) Gap(k) VS. k

(c) W_k VS. k

Figure 8: An illustration of a particular data set where the number of clusters returned does not equal the true number of clusters

Figure 8(b) shows that despite the fact that a global maximum exists, the algorithm picks the first k such that Gap(k) >Gap($k + 1$) + sd_k. Since the standard deviation is usually very small, the algorithm basically looks for the first local maximum. This explains why in this case $k = 1$ is picked. Figure 8(c) shows the within-cluster dispersion curve. It does not seem that an obvious elbow point exists in the curve, although one could argue that the value of W_k stays almost constant past $k = 5$, and therefore $k = 5$ is the elbow point and also the correct number of clusters.

There is no noticeable difference when we enforce the separation condition; nor do the underlying distribution and number of points per cluster appear to matter significantly. This may be because, for a small number of clusters, it is unlikely that the data points will be clumped together even if we do not require them to be well separated. The fact that the distribution does not matter agrees with the literature[19] that unlike other methods (described above), the performance of the gap statistic is not influenced by the distribution of the data points. As more results will give us a more complete picture of the algorithm's performance, we search for ways to make the gap statistic more efficient in Section 3.

General Discussion of Results

Few general trends spanning all algorithms are apparent. All algorithms, except for the data spectroscopic algorithm, performed worse on data sets in two dimensions than in higher dimensions. Because the algorithms are so widely varied, this result may be due to either a bias in the generated data in low dimensions, or an underlying property of clustering which makes the problem harder in low dimensions. The data spectroscopic algorithm's exemption from this trend is quite surprising.

G-means was the only algorithm that showed any response to varying cluster sizes. It tends to overestimate the number of clusters more when the points per cluster are randomly determined than when they are fixed. The other algorithms performed almost identically regardless of the number of points per cluster.

The k-means based algorithms (X-means, G-means), seem to be the most influenced by the separation condition. When it is not enforced, they tend to underestimate the number of clusters. This is expected as some clusters will be significantly overlapping. The other algorithms, with the exception of the self-tuning spectral algorithm's abysmal performance for $k = 2$ without the separation condition, show little response to the removal of the separation condition.

This may be misleading, however. Even when the separation condition is not enforced, the cluster variances are scaled as described above so that few of the clusters will be overlapping in the expected case. The effects of these occasional overlaps may be only noticable on the k-means algorithms because they do so well in the case when the clusters are not overlapping, while the effects on the other algorithms are overshadowed by other problems with their performance.

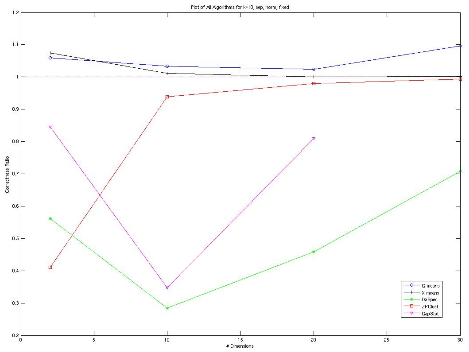


Figure 9: Comparison of Algorithms performance with $k = 10$, the separation condition enforced, normal distribution, and fixed number of points per cluster.

A comparison of all algorithms is shown in Figure 9 for a data set with a normal distribution with a fixed number of points per cluster, and the separation condition enforced. Overall, the best results seem to be found by X-means (except for the occasional extremely far off result), with G-means a close second. While the exact running time was not recorded, some algorithms finished the test sets in a matter of hours (G-means), while others took days (X-means, Data Spectroscopic), or even weeks (Self-Tuning Spectral, Gap Statistic). The self-tuning spectral and gap statistic algorithms ran so slowly in higher dimensions and number of clusters that we were unable to finish testing on all of the data sets.

It should be noted in a direct comparison that the algorithms can be broadly divided into two groups: those which require as an input parameter the range of values to try, and those which do not. Requiring a range of values as an input parameter for only some of the algorithms leads to two problems in the comparison:

- First, we were able to chose a “good” range of values for the algorithm to try based on our knowledge of the true number of clusters. Because it is infeasible to run the algorithms on an extremely large range (for example, from 1 to the number of data points), their use requires some vague idea of the true number of clusters. In effect, they weaken the condition imposed by most clustering algorithms – that the true number of clusters be known *a priori* – and instead require only that the true number of clusters can be confirmed to be within some relatively small range.

- Second, if the user knew a range for the true number of clusters, this would lead to even more effective variations of the algorithms that require no range. For example, we have seen that X-means generally returns either an answer very close to the correct number of clusters, or an answer off by an order of magnitude or more. If a range for the true number of clusters were known, the X-means output could be refined to return only the best value within this range. This simple modification would drastically increase the average quality of X-mean’s solution. (Pelleg’s X-means implementation does in fact allow for this, but we chose not to utilize this feature in our comparison).

3 Further Work: The Gap Statistic and the Elbow Point

In this section we describe some theoretical considerations of the gap statistic algorithm. In Section 3.1 we consider modifications of the gap statistics algorithm that may increase its running time dramatically. In Section 3.2 we examine the “elbow” phenomenon from a simplified continuous version of the data clustering problem.

3.1 Characterization of the Elbow Point

The gap statistic identifies the number of clusters by comparing the original $\log(W_k)$ curve with its expected values under uniform reference distributions. A simpler way is to fully exploit the intuitive meaning of an “elbow point”.[22] This paper defines the elbow point as the point where the second derivative is maximized. In fact, because we are dealing with a discrete case here (the possible numbers of clusters k are discrete integers), we are not able to get precise measures of the second derivatives. However, by making use of the second derivative approximation: $f''(k) \approx f(k-1) + f(k+1) - 2f(k)$ for a function f, we can approximate values of W_k'', where W_k is a function of k. We can then find the value of k that maximizes the value of W_k'', and that k is our choice for the number of clusters. We can also interpret the approximation $f''(k) \approx f(k-1)+f(k+1)-2f(k)$ as

$$f''(k) \approx \frac{f(k-1) - f(k)}{k - (k-1)} - \frac{f(k) - f(k+1)}{(k+1) - k}$$

. The elbow point is thus characterized as the point where there is the greatest change in slopes, which is how we would expect the elbow point to behave intuitively.

The above computation greatly simplifies the original gap statistic in that we no longer need to compute B copies of W_k^* based on the reference data sets. However, we still must calculate W_k for each value of $k = 1, 2, ..., K$ by finding the partition based on PAM, and computing W_k from the partition. One way to save more computation time is to use the techniques of optimizing a function. One possible technique is the golden section search algorithm. We are interested in the function $W_{new}(k) = W_{k-1} + W_{k+1} - 2W_k$, where $W_{new}(k)$ is a function of k. To find the k that maximizes this function, we start evaluating the function at the boundary values ($k = 1$ and $k = K$ where K is the upper boundary on the range given to the algorithm), and pick the next evaluation point according to the golden ratio[6]. This way, we will only need to evaluate a few values of W_k (and hence $W_{new}(k)$), saving a lot of computation time. One potential problem with this method is that the values of k are discrete integers, but the golden section search will likely tell us to evaluate at points that are non-integers as the algorithm proceeds. In this case we could choose to evaluate at the integer that is closest to the optimal point of evaluation.

This proposed new idea is motivated by our experience that the gap statistic runs too slowly, and improving the running time would make it practical for a much larger range of data sets than it is currently. We believe that this modification would be more computationally feasible than the original gap statistics algorithm on data sets with large numbers of clusters and dimensions.

3.2 Continuous Case

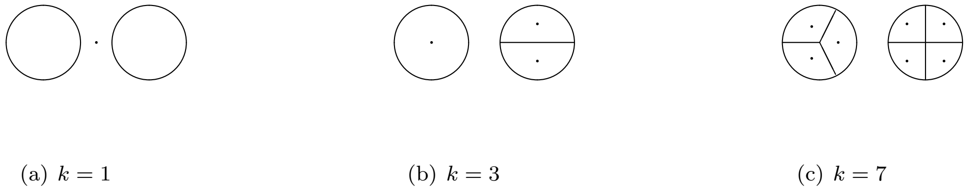
To gain a better theoretical understanding of the elbow phenomenon, we consider a simple continuous version of the clustering problem. Our version is defined as follows:

- We are given two circular areas, A_1, A_2, in two dimensions, with centers a_1, a_2, and $\|a_1 - a_2\| = 2$. Both circles have the same radius, ε.
- For a given k, we seek a placement of k centers that divide the areas into k smaller areas, $C_1, C_2, \ldots, C_k$, according to the Voronoi partition of the k points.
- We seek the configuration of k points that minimizes the generalized continuous within cluster sum of squares over these areas, W_k. We use

$$W_k = \sum_{i=1}^{k} \left[\int_{C_i} \| x - \bar{x}_i \| \, \mathrm{d}x \right], \text{where } \bar{x}_i = \frac{1}{|C_i|} \int_{C_i} x \mathrm{d}x$$

That is, $\bar{x}_i$ is the centroid of region C_i.

An example of center configurations for a few k is shown in Figure 10. To simplify things further, aside from the $k = 1$ case where the center is placed directly between the two circles, we assume that no center controls areas from both circles, and that the optimal placement of points divides the two circles into a roughly equal number of wedges.



(a) $k = 1$ (b) $k = 3$ (c) $k = 7$

Figure 10: Wedge configurations for various k. Dots correspond to cluster centers.

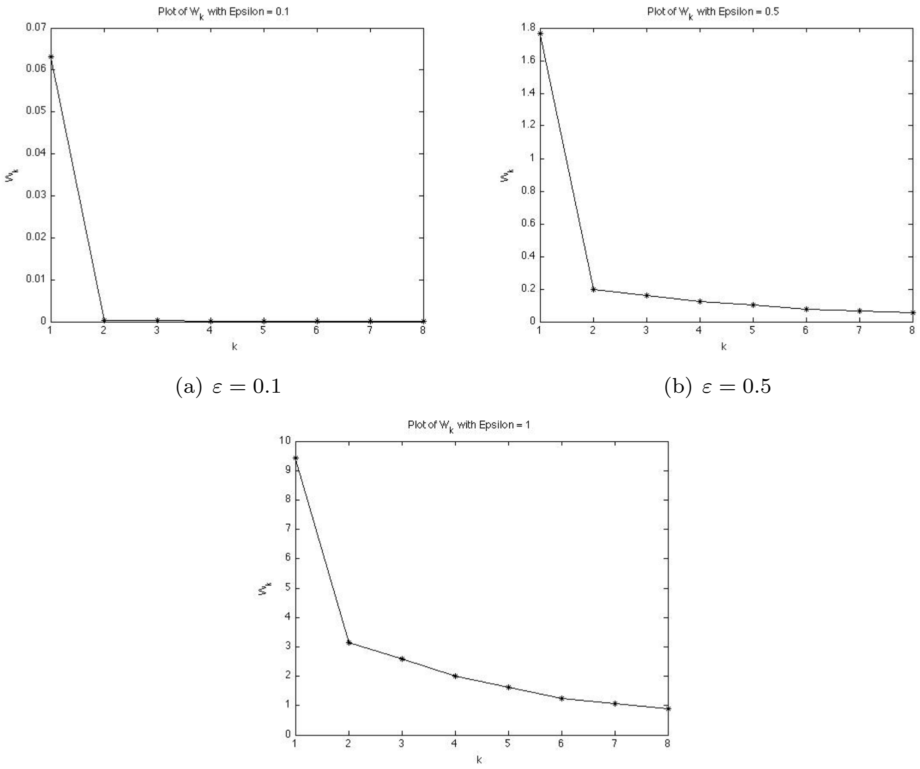
For $k = 1$ it is simple to show that $W_1 = \pi(\varepsilon^4 + 2\varepsilon^2)$. We are able to prove (see Appendix 5) that for all $k > 1$ we have:

$$W_k = \begin{cases} 2i * T_i & k = 2i \\ (i-1) * T_{i-1} + i * T_i & k = 2i - 1 \end{cases}$$

where

$$T_i = \varepsilon^4 \left[\frac{\pi}{2i} - \frac{4i}{9\pi} \sin^2\left(\frac{\pi}{i}\right) \right]$$

A plot of the within cluster sum of squares function is shown for various radii in Figure 11. We see a noticable "elbow" for small ε, but the prominence of the elbow decreases as ε increases to 1 (at which point the circles are touching).



(a) $\varepsilon = 0.1$

(b) $\varepsilon = 0.5$

(c) $\varepsilon = 1$

Figure 11: The elbow decreases as ε increases.

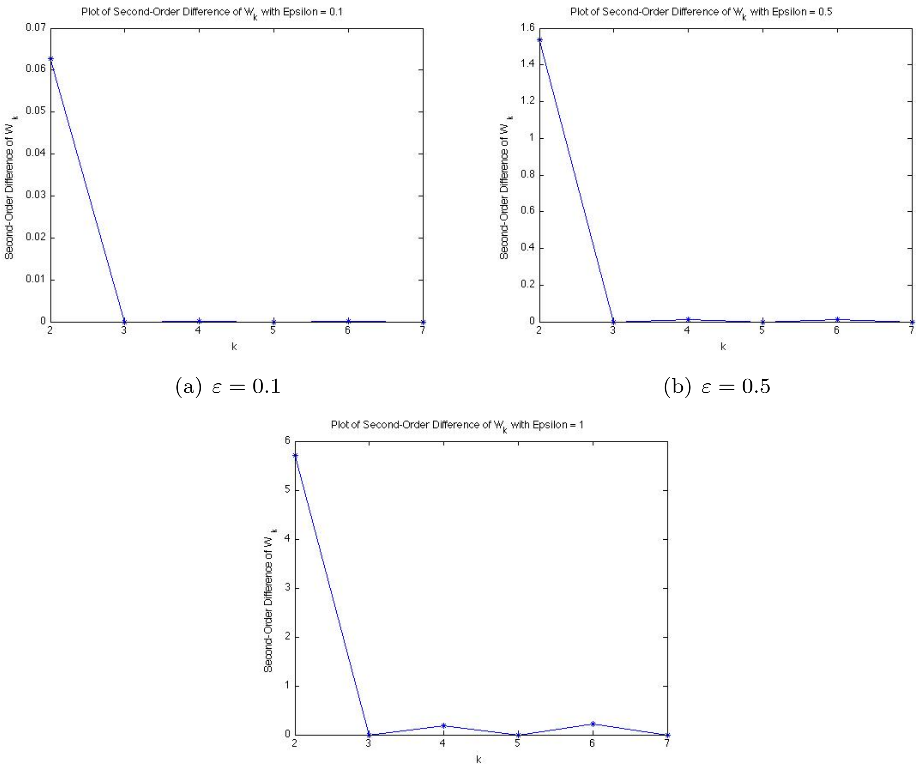
To consider the correctness of the heuristic mentioned above of maximizing the second-order differences of W_k, we also plot these for several ε in Figure 12. The function is maximized at $k = 2$, as expected. We see a pair of local maximums for higher k, which become more prominent as ε increases.

The fact that an elbow shows up even on our very simplified theoretical exercise lends credence to the gap statistic hueristic. We observe that the elbow shrinks as the clusters get closer together, indicating perhaps that the gap statistic might work well only for well-separated clusters. The plot of second-order differences further confirms our belief. There is a large global maximum at the true number of clusters, but also a pair of smaller, local minima which grow in size as the clusters move closer. For more complicated data sets, these local maxima may prove troublesome for the gap statistic, especially on poorly-separated datasets.

4 Conclusion and Recommendations for Future Work

In this paper we compared a variety of widely used algorithms for determining the number of clusters on many different data sets. Our results demonstrate that the k-means based algorithms routinely outperform the other algorithms on most test cases. Further, we note a strong indication that the clustering problem is more difficult for all algorithms in low number of dimensions. This increase in performance in higher dimensions, however, comes at a cost – many algorithms are computationally infeasible in more than ten dimensions.

It should be pointed out that these results are only guaranteed to be applicable to the types of data sets used in our tests. In his paper developing G-means, Hamerly demonstrates that G-means performs much better than X-means on clusters that are generated from a non-spherical Gaussian distribution. Spectral clustering algorithms have been developed specifically for use on data which is organized into complex or unbalanced configurations as opposed to data which fits a more standard model like the uniform or normal distributions. An example of such a data set is the ring data set.



(a) $\varepsilon = 0.1$

(b) $\varepsilon = 0.5$

(c) $\varepsilon = 1$

Figure 12: The local maxima become more noticable as ε increases.

Finally, there is much potential for expansion on our theoretical work. Adding clusters, dimensions, and changing the shapes of the clusters or configurations of the centers in the continuous case may all lead to different results. In particular, it is unlikely that dividing the areas into wedges will be optimal for arbitrarily large k. There may be a secondary elbow at the point at which the optimal placement switches from wedges to some other configuration. Similarly, while we assumed that (aside from the $k = 1$ case), cluster centers would control points only in one of the two circles, it seems likely that when the radius grows large enough, the ideal placement for three centers will require a center in each circle and a center between them. Again, it would be interesting to consider the radius at which this switch occurs, and how the elbow point responds. It would be interesting to implement the gap statistic modification described in Section 3.1, and compare its results with those predicted from the theoretical work. In addition, this modification would allow further data to be collected on the gap statistic algorithm on data sets with more clusters or a higher number of dimensions, because of the substantial increase in running time.

Acknowledgements

This paper reports on the results obtained during the 2009 IMA Interdisciplinary REU, June 28 to July 31. We gratefully acknowledge the IMA for providing this opportunity. We would also like to thank Gilad Lerman and Mark Iwen for helpful suggestions and advice. Finally, we would like to thank Yi Wang for collecting the codes which we used in this work.

References

[1] M. Ackerman and S. Ben-David,“ Which Data Sets are ‘Clusterable’? – A Theoretical Study of Clusterability”, preprint, 2008.

[2] M. Ackerman and S. Ben-David, "Measures of Clustering Quality: A Working Set of Axioms for Clustering", preprint, 2009.

[3] T. W. Anderson and D. A. Darling, "Asymptotic Theory of Certain 'Goodness of Fit' Criteria Based on Stochastic Processes", *The Annals of Mathematical Statistics*, 23(2):193–212, 1952.

[4] J. Bently, "Multidimensional divide-and-conquer", *Communications of the ACM*, 23(4):214–229, ACM, 1980.

[5] G. Hamerly and C. Elkan, "Learning the k in k-means", *NIPS*, 2004.

[6] J. Keifer, "Sequential Minimax Search for a Maximum", *Proceedings of the American Mathematical Society*, 1953.

[7] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, New York, 1990.

[8] J. Kleinberg, "An Impossibility Theorem for Clustering", preprint, 2002.

[9] B. Manthey and H. Roglin, "Improved Smoothed Analysis of the k-Means Method", preprint, 2008.

[10] G. Marsaglia, "Choosing a Point from the Surface of a Sphere", *The Annals of Mathematical Statistics*, 43(2):645–646, JSTOR, 1972.

[11] M. Meila, "The Uniqueness of a Good Optimum for k-Means", *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[12] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy, "The Effectiveness of Lloyd-Type Methods for the k-Means Problem", *SODA*, 2007.

[13] D. Pelleg and A. Moore, "Accelerating Exact k-means Algorithms with Geometric Reasoning", *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 277–281, New York, NY, 1999.

[14] D. Pelleg and A. Moore, "X-means: Extending K-means with Efficient Estimation of the Number of Clusters", *Proceedings of the 17th International Conference on Machine Learning*, 727–734, P. Langley, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 2000.

[15] G. Schwarz, "Estimating the Dimension of a Model", *The Annals of Statistics*, 6(2):461–464.

[16] T. Shi, M. Belkin, and B. Yu, "Data Spectroscopy: Eigenspaces of Convolution Operators and Clustering", preprint, 2008.

[17] N. Srebro, G. Shakhnarovich, and S. Roweis, "When is Clustering Hard?", *PASCAL Workshop on Statistics and Optimization of Clustering*, 2005.

[18] N. Srebro, G. Shakhnarovich, and S. Roweis, "An Investigation of Computational and Informational Limits in Gaussian Mixture Clustering", *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[19] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the Number of Clusters In a Data Set Via the Gap Statistic", *Journal of Royal Statistical Society*, 2001.

[20] N. X. Vinh, J. Epps, and J. Bailey, "Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary?", *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[21] M. Yan and K. Ye, "Determining the Number of Clusters Using the Weighted Gap Statistic", *Biometrics*, 63:1031–1037, 2007.

[22] S. Yue, X. Wong, M. Wei, "Application of Two-Order Difference to Gap Statistic", *Transactions of Tianjin University*, 14(3), 2008.

[23] L. Zelnik-Manor and P. Perona, "Self-Tuning Spectral Clustering", *NIPS*, 2004.

5 Appendix A – Derivation of Continuous W_k Formula

As mentioned above, we seek a generalization of the within cluster sum of squares from the discrete case:

$$W_k = \sum_{l=1}^{k^*} \left[\frac{1}{|C_l|} \sum_{i=1}^{n_l} \|x_i - \bar{x}_l\|^2 \right], \text{where}\, \bar{x}_l = \frac{1}{|C_i|} \sum_{i=1}^{n} x_i, \text{and}\, n_l = |C_l|$$

Assuming a uniform distribution of data points within the two circles, we can generalize the above formula to the continuous case, as:

$$W_k = \sum_{l=1}^{k^*} \left[\frac{1}{|C_l|} \int_{C_l} \|x - \bar{x}_i\|^2 \mathrm{d}x \right], \text{where}\, \overline{x}_i = \frac{1}{|C_i|} \int_{C_i} x \mathrm{d}x$$

That is, the mean in the discrete case corresponds to the centroid in the continuous case, and the normalized sum of squares for a single cluster in the discrete case corresponds to the second moment of area for a cluster about its centroid in the continuous case.

To derive an expression for W_k in the continuous case, we consider dividing a single circle into i wedges, as described above. In particular, we find T_i, the second moment of area for one of the i wedges about its centroid. The total within cluster sum will then be equal to the sum of each of the clusters.

$$W_k = \sum_i T_i$$

Since T_i will be the same for all wedges in the same circle, we consider just a single wedge, symmetric along the x-axis. It's centroid is then located along the x-axis, and if we take the center of the circle to be at the origin, we can find the x-coordinate of the centroid as:

$$\bar{x}_i = \frac{1}{|C_i|} \int_{C_i} x \mathrm{d}A = \frac{i}{\pi \varepsilon^2} \int_{-\frac{\pi}{i}}^{\frac{\pi}{i}} \int_0^{\varepsilon} r^2 \cos\theta \mathrm{d}r \mathrm{d}\theta = \frac{2}{3} \frac{i\varepsilon}{\pi} \sin\left(\frac{\pi}{i}\right)$$

We then find the second moment of area about the centroid, T_i. Again we use polar coordinates to simplify the process:

$$\begin{aligned} T_i = \int_{C_i} \|x - \bar{x}_i\|^2 \mathrm{d}A &= \int_{-\frac{\pi}{i}}^{\frac{\pi}{i}} \int_0^{\varepsilon} (r^3 - 2r^2 \cos\theta \bar{x}_i + r\bar{x}_i^2) \mathrm{d}r \mathrm{d}\theta \\ &= \varepsilon^4 \left[\frac{\pi}{2i} - \frac{4i}{9\pi} \sin^2\left(\frac{\pi}{i}\right) \right] \end{aligned}$$

For all $k > 1$, if k is even, both circles will be split into an equal number of wedges. If k is odd, one cluster will have an additional wedge. Thus we have:

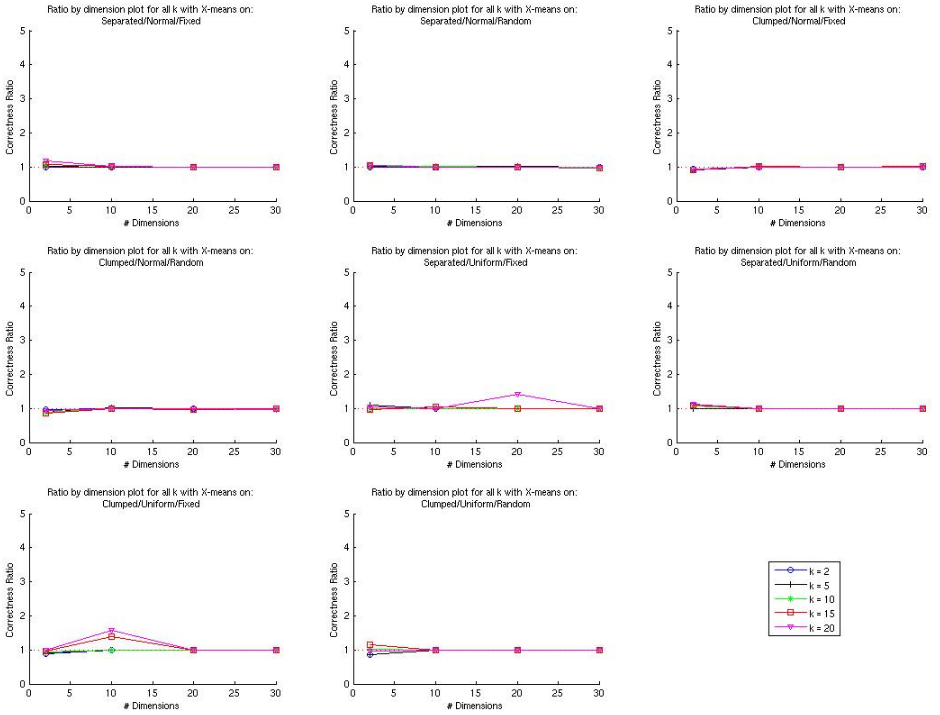
$$W_k = \begin{cases} 2i * T_i & \text{if } k = 2i \\ (i-1) * T_{i-1} + i * T_i & \text{if } k = 2i - 1 \end{cases}$$

6 Appendix B – Experimental Results

Each algorithm's output is given a separate page. The output is organized as follows:

- There are eight plots, one for each configuration of the binary conditions (separation condition enforced/not enforced, data distributed normally/uniformly, points per cluster fixed/random).
- Each plot shows the correctness ratio (y-axis) plotted against the number of dimensions (x-axis) for the different k^*. This ratio is calculated as $\hat{k}/k^*$, where $\hat{k}$ is the number of clusters returned by the algorithm, and k^* is the true number of clusters for the data set.
- To make comparison easier, the axes are held fixed for all plots. Because X-means and G-means tend to have very small errors, zoomed-in plots are provided on an additional page for both of these algorithms to show the variation in their performance.

X-means



Ratio by dimension plot for all k with X-means on: Separated/Normal/Fixed

Ratio by dimension plot for all k with X-means on: Separated/Normal/Random

Ratio by dimension plot for all k with X-means on: Clumped/Normal/Fixed

Ratio by dimension plot for all k with X-means on: Clumped/Normal/Random

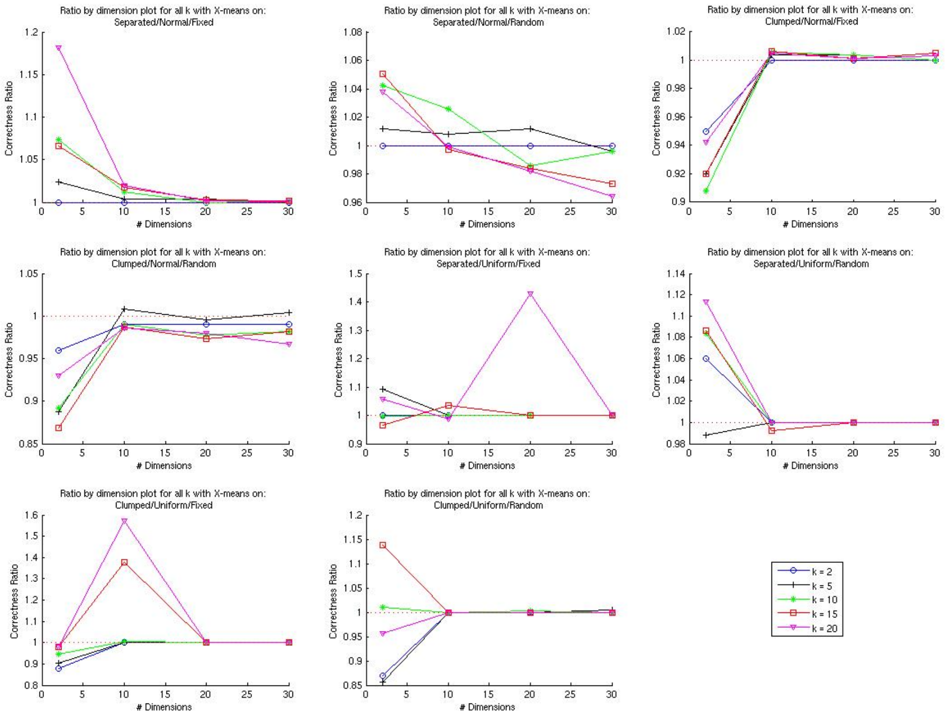
Ratio by dimension plot for all k with X-means on: Separated/Uniform/Fixed

Ratio by dimension plot for all k with X-means on: Separated/Uniform/Random

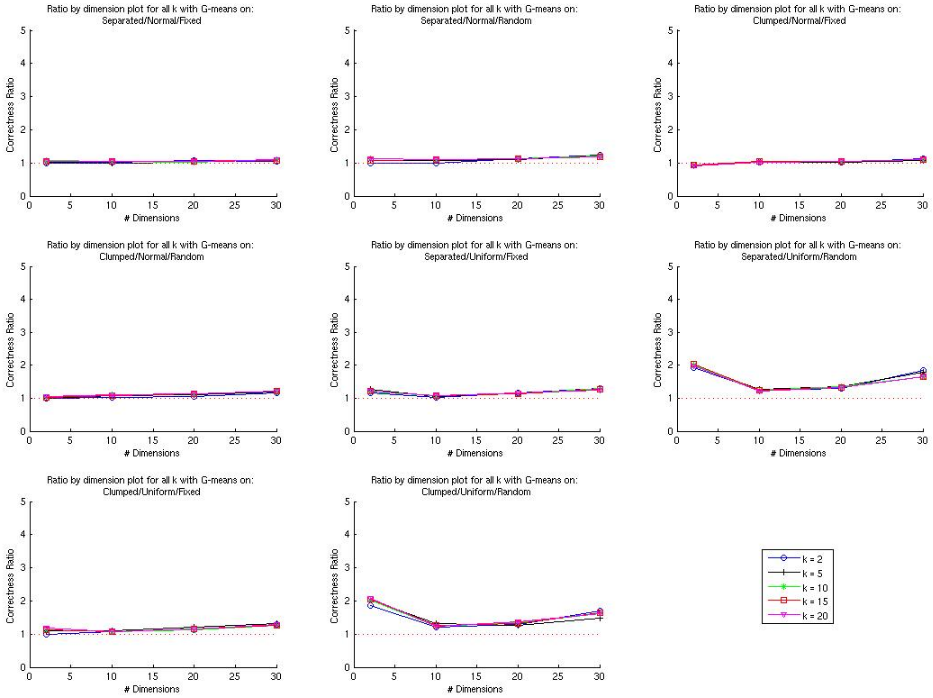
Ratio by dimension plot for all k with X-means on: Clumped/Uniform/Fixed

Ratio by dimension plot for all k with X-means on: Clumped/Uniform/Random

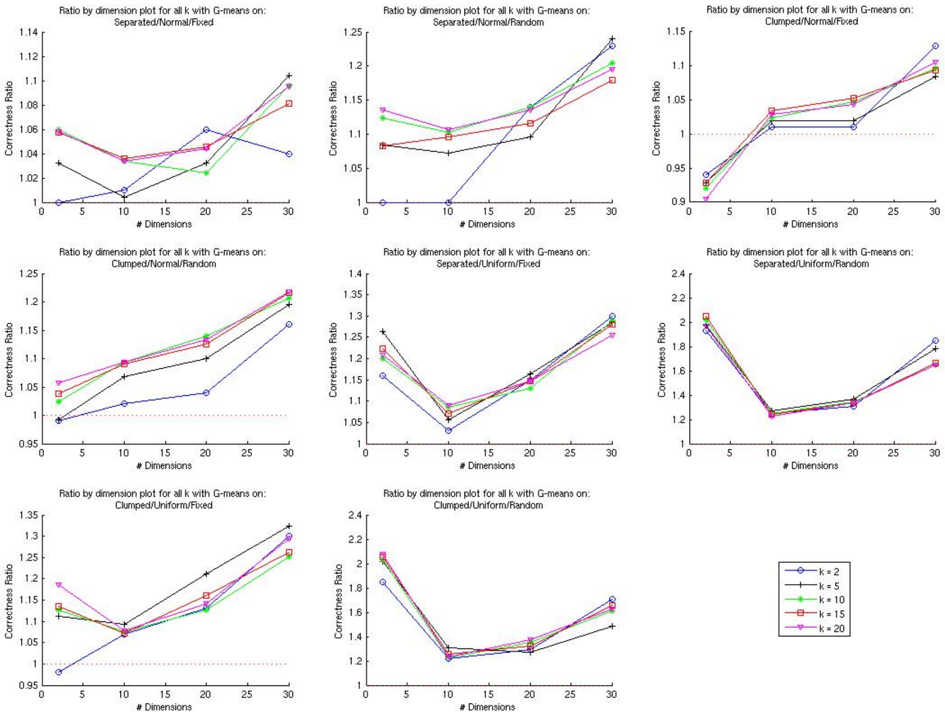
X-means (Zoomed-in)



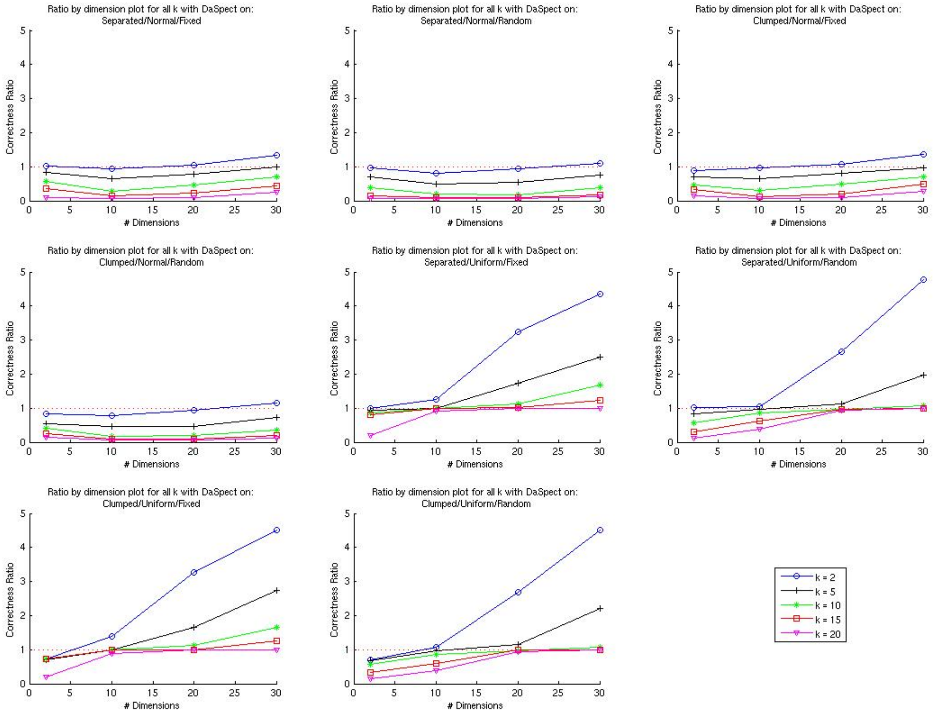
G-means


Ratio by dimension plot for all k with G-means on: Separated/Normal/Fixed
Ratio by dimension plot for all k with G-means on: Separated/Normal/Random
Ratio by dimension plot for all k with G-means on: Clumped/Normal/Fixed
Ratio by dimension plot for all k with G-means on: Clumped/Normal/Random
Ratio by dimension plot for all k with G-means on: Separated/Uniform/Fixed
Ratio by dimension plot for all k with G-means on: Separated/Uniform/Random
Ratio by dimension plot for all k with G-means on: Clumped/Uniform/Fixed
Ratio by dimension plot for all k with G-means on: Clumped/Uniform/Random
Correctness Ratio
# Dimensions
0
1
2
3
4
5
0
5
10
15
20
25
30
k = 2
k = 5
k = 10
k = 15
k = 20

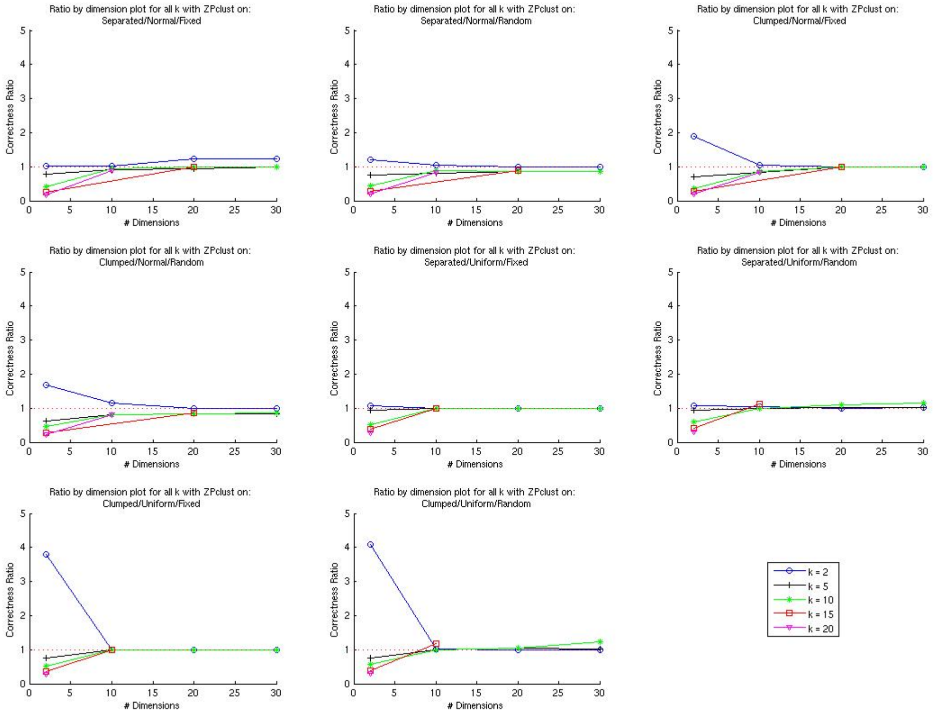
G-means (Zoomed-in)



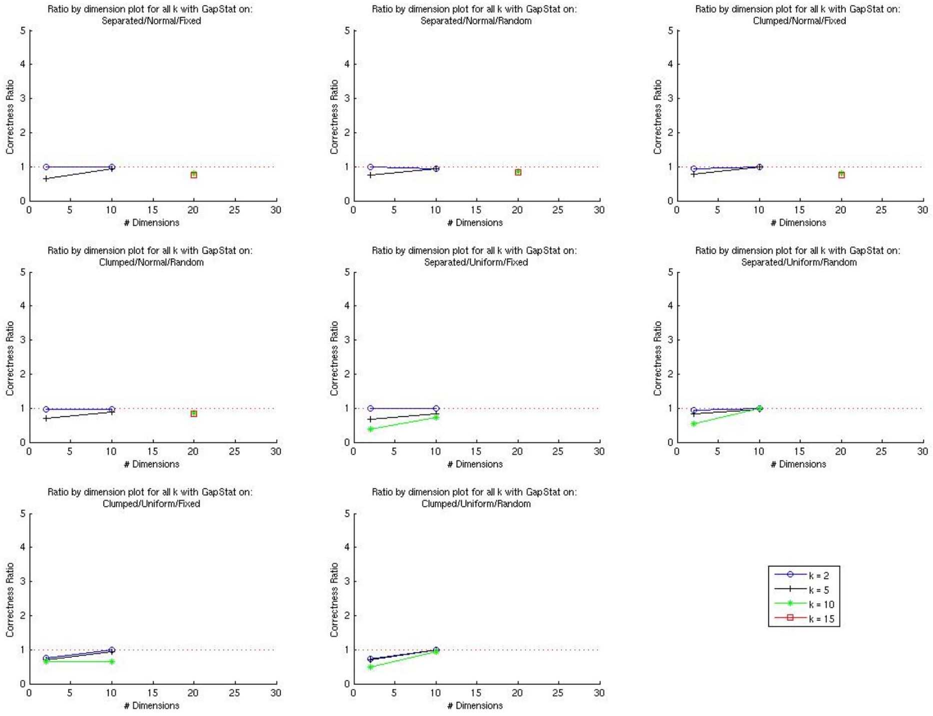
Data Spectroscopic



Self-Tuning Spectral



Gap Statistics


Ratio by dimension plot for all k with GapStat on:
Separated/Normal/Fixed
Ratio by dimension plot for all k with GapStat on:
Separated/Normal/Random
Ratio by dimension plot for all k with GapStat on:
Clumped/Normal/Fixed
Ratio by dimension plot for all k with GapStat on:
Clumped/Normal/Random
Ratio by dimension plot for all k with GapStat on:
Separated/Uniform/Fixed
Ratio by dimension plot for all k with GapStat on:
Separated/Uniform/Random
Ratio by dimension plot for all k with GapStat on:
Clumped/Uniform/Fixed
Ratio by dimension plot for all k with GapStat on:
Clumped/Uniform/Random
Correctness Ratio
# Dimensions
k = 2
k = 5
k = 10
k = 15