k-means initialisation algorithms: an extensive comparative study

A thesis submitted for the degree of
Master of Science by Dissertation in Computer Science

Simon Harris
School of Computer Science and Electronic Engineering
University of Essex

March 8, 2021

Abstract

The k-means data clustering algorithm, whilst widely popular, is not without its drawbacks. In this work, we are particularly interested in the sensitivity of k-means to its initialisation, in the form of a set of initial centroids. Since the cluster recovery performance of k-means can potentially be improved by better initialisation, numerous algorithms have been proposed with the intention of producing better initial centroids. However, despite several decades since k-means was first formalised, it is still unclear which initialisation algorithm should be used in any particular clustering scenario. With this in mind, we empirically compare 17 published k-means initialisation algorithms by running them against 6,000 synthetic and 28 real-world data sets. The synthetic data sets were produced under many different configurations, allowing us to explore how each algorithm performs in each scenario. Hence, the results of our experiments may be particularly useful for those considering k-means for a non-trivial clustering scenario.

This work also introduces a new set of software libraries originally developed for use as part of our experiments, including implementations of the k-means initialisation algorithms covered in this work, along with data cleaning and data generation tools. These tools are made freely available to the wider community under a permissive open source licence. This is done in part to make sure that the research presented is replicable, but also in the hope that the research and results shared here may be of value to future researchers and developers in the field. As with all open source projects, contributions from the wider community are invited.

It is intended that the research should be placed in its historical context, and so we conduct an extensive literature review, including an overview of previous comparable surveys, followed by more detailed coverage of the specific algorithms included in our survey as described in the respective original literature.

Keywords: Data clustering; k-means; k-means initialisation.

Dedication

This work is dedicated to my big sister Rebecca who sadly passed away late in its production, even though she wouldn't have found it particularly interesting.

Shine on you crazy diamond.

Contents

List of Algorithms

List of Figures

1 Introduction

1.1 Data clustering

Data clustering is an important and extensively-studied technique in the fields of exploratory data analysis, data mining and pattern recognition. It comprises data-driven algorithms capable of partitioning unlabelled data into meaningful groups, each composed of similar entities. Such algorithms are widely used to address practical problems in various areas of research such as business, medicine, geographical information systems and many more [1, 2, 3, 4].

The main aim of any clustering algorithm is to partition a data set with points $\{x_1, x_2, ..., x_N\}$ in the V-dimensional space $\mathbb{R}^V$ [5] into K clusters $S = \{S_1, S_2, ..., S_K\}$, so that each cluster is composed of homogeneous data points. Homogeneity in this case is typically defined with the help of a similarity or dissimilarity measure (for details, see Section 1.2). There has been a considerable amount of research effort in data clustering, leading to many algorithms applying different strategies to partition a given data set. We direct interested readers to the many surveys such as [6, 7, 8] and the references therein.

Data clustering algorithms can broadly be categorised into three main classes: partitional, hierarchical, and density-based. In their original form, partitional clustering algorithms identify K clusters in a data set, so that each data point is assigned to exactly one cluster and no data point is left unassigned. In this way, the intersection of any two clusters $S_i, S_j \in S$ with $i \neq j$ is empty, and clusters can be described as exclusive. Versions of these algorithms have been developed using fuzzy logic so that a data point belongs to each of the K clusters with different degrees of membership, typically between 0 and 1, in total summing to 1 [9].

Hierarchical clustering algorithms identify more than one cluster along with the tree-like relationship that exists between them. In other words, clusters may be nested within other clusters. The usual approaches are top-down (divisive) and bottom-up (agglomerative).

Density-based clustering algorithms define clusters as contiguous areas of higher density separated by contiguous areas of lower density. This approach is particularly suited to data containing outliers—data points differing greatly from others in the data set—or noise [4]. The density of an area, or density thresholds, can be calculated in a number of ways, as can the distance between clusters, and indeed the interpoint distance (for details see [10, 11], and references therein).

A further dichotomy that can be made is that clustering can be described as complete or partial [4]. In a complete clustering, every data point is assigned to a cluster. By contrast in partial clustering this may not be the case, and data points may remain unassigned. This approach is suited to scenarios where it cannot be assumed that every data point belongs to a meaningful group and therefore may, as with density-based clustering, may also be suited to data containing outliers, for example.

The selection of which clustering approach to employ is not necessarily straightforward. Indeed, the definition of what constitutes a cluster or a “good” clustering solution may not always be clear, and can be highly specific to the particular application in question [4]. Thus, the choice of clustering approach may have to be driven by the nature of the data and the aims of the researchers involved [12]. The difficulties involved have even led to the question of whether clustering is an art or a science being raised [13].

1.2 *k*-means

In this work, we are interested in a particular data clustering algorithm, k-means [14, 15]. This is arguably the most popular partitional clustering algorithm there is [7, 2], and implementations of k-means can be found in major software packages used for data analysis, including MATLAB, R, and Python’s scikit-learn library [16, 17, 18].

k-means aims to partition a data set X containing N points into K homogeneous clusters $S = \{S_1, S_2, ..., S_K\}$. The dissimilarity between $x_i, x_t \in X$ can be measured using the squared Euclidean distance

$$d(x_i, x_t) = \sum_{v=1}^{V} (x_{iv} - x_{tv})^2, \tag{1.1}$$

where V is the number of features describing each $x_i \in X$. Each cluster

$S_k \in S$ is represented by a centroid $c_k \in \mathbb{R}^V$, which—assuming dissimilarity is measured using Euclidean distance—is the component-wise mean over all $x_i \in S_k$. That is $c_{kv} = |S_k|^{-1} \sum_{x_i \in S_k} x_{iv}$ for $v = 1, 2, ..., V$. Given all of the above, we can see that k-means minimises the criterion

$$W = \sum_{k=1}^{K} \sum_{x_i \in S_k} d(x_i, c_k). \tag{1.2}$$

This, therefore, is the objective function for k-means. The above has been given several names, but here and throughout we use the term Sum of Squared Errors or SSE.

The steps of the k-means algorithm are show in Algorithm 1.

Algorithm 1: The k-means Algorithm

```
1. Select $K$ centroids $C = \{c_1, c_2, ..., c_K\}$, using a chosen
   initialisation strategy.
2. For each $x_i \in X$, calculate the distance between $x_i$ and each
   $c_k \in C$ using (1.1). Assign $x_i$ to the cluster $S_k$ represented by
   the nearest $c_k$.
3. Update each $c_k \in C$ to the component-wise mean over all
   $x_i \in S_k$.
4. If Step 3 produced changes to the centroids, go to Step 2.
   Otherwise, the algorithm has converged.
```

The strengths of the k-means algorithm are well-documented:

1. The k-means algorithm is relatively straightforward to understand and implement.
2. The algorithm is computationally efficient, with linear time complexity with regard to the number of data points N [19, 20, 21].
3. Given an initial set of centroids, k-means is proven to converge deterministically in a finite number of steps [20, 22].

That k-means will always converge can easily be understood if we consider that for a given data set, there are only ever a finite number of possible clusterings, specifically K^N, and that k-means monotonically decreases SSE, in other words will never move from one clustering solution to a worse solution [23].

We acknowledge that despite its strengths and popularity, k-means is not without its weaknesses:

1. The algorithm will find K clusters even in non-grouped data.
2. Use of the Euclidean distance (1.1) can lead to a bias towards spherical or Gaussian clusters.
3. Clustering is performed based on all features, and all features have the same weight of contribution to the clustering. In other words, there is no direct support for scenarios where different features may have different degrees of relevance.
4. The algorithm can be sensitive to outliers, since an outlier will always be assigned to a cluster.
5. Given an initial candidate solution, in the form of K initial centroids, k-means will converge to a local optimum, but is not guaranteed to arrive at the global optimum. Thus k-means is highly dependent on the selection of initial centroids.

In this work, we are primarily interested in point (5). The tendency of k-means to arrive at a non-globally optimal solution is perhaps the most salient and problematic of its shortcomings. Indeed this issue is noted in countless published works describing k-means.

We can define local optima in this scenario by noting that for any locally optimal clustering solution for a given dataset, there exists a solution with a lower value for SSE as per the objective function (1.2). In fact, [24, 25] calculate that for a dataset of modest size, with $N = 200$ and $V = 8$, there are potentially thousands of local optima which could be arrived at.

Another definition [26] of a local optimum is that it is a clustering solution that cannot be improved by moving single points between clusters, whereas the global optimum solution cannot be improved in any way.

Given this dependence on the selection of initial centroids, many diverse algorithms have been proposed with the intention of identifying "better" initial centroids for k-means. Such proposed solutions include employing genetic algorithms [27], binary search [19], *kd*-trees [28], probability distribution [23] and more. Proposed algorithms may be deterministic or non-deterministic. Section (2.1) presents a number of surveys and comparisons.

We note that despite the considerable research effort invested in the subject, there appears to be little to no clear guidance on which initialisation strategies might be most appropriately applied to specific clustering endeavours. It is this guidance to which this work seeks to contribute.

1.3 Outcomes

There are two main facets to this work. The first is an extensive empirical comparison of 17 k-means initialisation algorithms, involving running the algorithms against 6,000 synthetic data sets and 28 freely-available real-world data sets. The data sets are described in Section 3.1. The results of the experiments are reported and discussed in Section 4 with the intention of contributing to the guidance described above.

The second main facet is the development of an open source software library written in Python, and comprising:

- An example implementation of k-means
- Implementations of the 17 k-means initialisation algorithms
- A small framework to run the algorithms (including the use of parallelisation where available) and pass the resulting centroids to k-means
- The synthetic data set generation tools used to prepare the experiments
- The tools used for downloading and cleaning the real-world data sets used in the experiments

This work is presented in several sections. Section 2 is comprised of a full background to our research, including a literature review of prior surveys of k-means initialisation algorithms and other related works, followed by a detailed description of each of the algorithms covered in our experiments. Section 3 describes our experimental methodology, while our results and discussion thereof are presented in Section 4. Finally, the software libraries developed to facilitate our experiments are introduced in Section 5.

2 Background and Literature Review

In this section we cover a selection of the research which precedes and informs the current work. Section 2.1 is a summary review of prior surveys and empirical comparisons of k-means initialisations, along with related research works of a similar nature which provide context for the current work. Section 2.2 describes the 17 initialisation algorithms compared in our experiments, including formal listings of the steps of those algorithms where appropriate.

2.1 Prior surveys

Several surveys and empirical comparisons of k-means initialisation algorithms have previously been published. These vary greatly in extent in terms of number of algorithms and data sets used, and may have disparate underlying aims. This section is intended to provide context for our research by presenting a brief overview of a number of such surveys.

2.1.1 Peña & Lozano 1999

In what may be the earliest survey of its kind to be published, Peña & Lozano [29] present an empirical comparison of four k-means initialisation algorithms run against just three data sets.

The algorithms explored are: i) a completely random partitioning of the data set, which we term Random Partition and describe in Section 2.2.1; ii) the selection of random points, also known as observations, from the data set as initial centroids, which we term Random Centroids and describe in Section 2.2.2; iii) an algorithm described by MacQueen [14], which is essentially Random Centroids, but with a modification whereby initial centroids are recalculated as each remaining data point is sequentially assigned to them; and iv) the Kaufman Approach (KA) presented in [30] and formally described in [29, Figure 3]. In this last approach, the first initial centroid is taken to be the most central point in the data set, and subsequent initial centroids are selected individually from the as-yet unselected data points, choosing the point which would have the highest number of data points assigned to it were it selected. The process continues until K initial centroids have been chosen. Of the four algorithms covered, three are non-deterministic with only KA being deterministic.

The data sets used are three well-known real-world data sets, specifically Iris, Ruspini and Glass sourced from the UCI Machine Learning Repository [31].

A notable aspect of the Peña & Lozano survey is its concern for reducing the effects of "instance order", in other words the order of points within the data set, which is stated to be one of the two key factors influencing the clustering performance of k-means, alongside the choice of initial centroids. However, the implementation of k-means used, [29, Figure 1], appears to be slightly distinct from the more widely-accepted algorithm described in for example [1], [25] and in Algorithm 1 above. The key difference is that in the modified k-means, centroids are recalculated every time a data point is reallocated to a different cluster, rather than after each full iteration over the k-means assignment and update steps, Steps 2 and 3 of Algorithm 1. In this way, we can see that the order of data points will influence the choice of initial centroids.

It is also interesting that while the survey treats the MacQueen algorithm as a k-means initialisation algorithm—as which it might certainly be used—at no point does the original author claim it to be such, rather, stating it to be "the k-means procedure" itself [14, p. 283]. We conjecture that the procedure described by MacQueen may represent what was termed k-means prior to the work of Forgy [15].

Finally, we note that, unlike many practitioners in later years, Peña & Lozano do not presume to know the number of clusters K in advance. Rather, several values are explored for each data set, for example values of 2, 7 and 10 for the Glass data set.

Results are compared to the output of a Genetic Algorithm (GA) developed by Peña & Lozano, an approach which they deem to have the potential to evolve towards the global optimum with probability arbitrarily close to 1. The results indicate that the KA and Random Partition initialisations consistently outperform the MacQueen and Random Centroids approaches in terms of cluster recovery. Furthermore, KA outperformed Random Partition on two out of three data sets. Still, the authors of [29] recommend continued usage of the more traditional Random Partition approach until more research into KA has been performed.

2.1.2 He et al. 2004

He et al. present a comparative study of initialisation methods for iterative refinement clustering algorithms, using k-means as a specific example [32]. k-means is chosen due to its simplicity, its satisfactory clustering performance especially where input data displays a Gaussian distribution, and also the fact that k-means initialisation methods may easily be applied to other iterative refinement algorithms. He et al. state that very few such surveys had been carried out at the time of their writing, and so endeavour to fill this gap in the literature.

Their study includes five initialisation algorithms including Random Centroids; a slightly modified version of Random Centroids based on perturbation of the mean of the input data, termed R-MEAN [33]; the Kaufman Approach covered by Peña & Lozano [29] (see Section 2.1.1); and the Katsavounidis, Kuo & Zhang (KKZ) algorithm we discuss in Section 2.2.4. The KKZ algorithm was also published in 1994, so might be considered to be cutting edge at the time the He et al. survey was conducted.

The final algorithm considered is the Simple Cluster Seeking method (SCS) [34], an early k-means variant implemented in the SAS commercial data analytics software. He et al. acknowledge that SCS was not originally conceived as an initialisation algorithm, rather as a clustering method in its own right. They further note that it suffers from the drawback of being sensitive to the ordering of the input data points, and also that it requires an additional threshold parameter to be supplied.

The algorithms are run against 25 two-dimensional synthetic data sets—the results from three of which are reported—and four real-world data sets. Results are compared using the "Cluster Compactness" (Cmp) and "Cluster Separation" (Sep) indices from [35].

In terms of findings, He et al. report that for synthetic data sets, there is little to choose between each method when considering the Cmp index, however SCS and KKZ show markedly better performance than the other methods in terms of the Sep index. Generally speaking, experiments using the real-world data sets lead to results consistent with those from synthetic data sets. In concluding, He et al. appear to favour KKZ over SCS due to the latter's shortcomings discussed above, not least the requirement for an additional parameter to be set, whereas the operation of KKZ is fully automated.

2.1.3 Steinley & Brusco 2007

Just three years after the He at al. survey described above, Steinley & Brusco [25] present a critical evaluation comparing a significantly more extensive 12 techniques for initialising k-means clustering. These include the Milligan, Bradley & Fayyad and Intelligent k-means approaches—all described in Section (2.2)—along with the strategies implemented by two commercial statistical and analytical software packages, SAS and SPSS.

Whilst it is not explicitly stated, it may be reasonable to assume that the approach ascribed in [25, p. 105] to SAS is the SCS approach described under He at al. above. Certainly both appear to suffer from the data instance order effects identified by He et al.

Steinley & Brusco state that at the time of their writing, the cluster recovery performance of these algorithms had not been directly compared, and so perform two experiments which are termed Simulation 1 and Simulation 2.

In Simulation 1, all 12 algorithms are run against synthetic data sets generated using the techniques described in [36]. The data sets varied in several ways, including: the number of clusters $K \in \{4, 6, 8\}$; the number of features $V \in \{4, 6, 8, 10\}$; and the degree and type of overlap. In this simulation, Steinley & Brusco chose not to vary the number of data points, consistently using 200 per data set, having previously found [37, 24, 38] this particular variation to have "negligible" effects on cluster recovery for the number of clusters and features under investigation.

Alongside comparing initialisation algorithms, Steinley & Brusco also report on within-data set effects across all algorithms, finding that factors such as the number of features V and clusters K have very little bearing on the relative success of each initialisation method. By contrast the methods were highly sensitive to the relative density and overlap of the clusters. These findings should be taken within the context of the experiments, however, where only a small number of clusters and features were considered.

Simulation 2 is similar to Simulation 1, but requires the processing of greater amounts of data. Here, $K \in \{5, 10, 20\}$, $V \in \{25, 50, 125\}$ and in this simulation the number of data points is varied, with $N \in \{200, 1000, 5000\}$.

For practical reasons, in this simulation the size and relative density of clusters is kept constant.

In this simulation, the Milligan method again consistently displays the most successful cluster recovery across all sets of factors. However, it is noted that this method requires the generation of a proximity matrix of size $N \times N$, which may be prohibitive due to the limitations of available computer memory. In such cases, Global k-means (Section 2.2.6) is recommended. The survey authors further recommend the use of Random Partition with many thousands of restarts in cases where a practical time limit is not necessary.

2.1.4 Celebi et al. 2013

Celebi et al. [39] perform a comparative study of eight linear time complexity k-means initialisation algorithms incorporating an empirical comparison. Six of the algorithms are non-deterministic, including: Random Partition, attributed by the survey authors to Forgy [15]; Random Centroids, attributed by the survey authors to MacQueen [14]; Bradley & Fayyad; and k-means++, described in Section (2.2). Two deterministic algorithms are also considered, namely "PCA-Part" and "Var-Part" as presented by [40].

The algorithms are compared using 32 real-world data sets and a considerable number of synthetic data sets. Generation of the synthetic data sets employs 576 configurations varying in: cluster overlap; number of features V ranging from 2 to 64; number of data points N ranging from 1,024 to 65,536; and number of clusters $K \in \{2, 4, 6, 8, 10, 12\}$. Furthermore, data sets are generated at three clustering complexity levels, defined as a function of Rand index [41], van Dongen [42], and Variation of Information [43]. In total, 12,288 data sets are generated.

One of the central aims of the survey is to compare, alongside cluster recovery performance, the computational efficiency of the algorithms, defined as the total CPU time incurred by the initialisation algorithm itself, plus k-means. Perhaps counter-intuitively, it is found that there is very little discrepancy in this regard between the algorithms, and this is attributed to the observation that more elaborate initialisation algorithms tend to lead to k-means requiring fewer iterations to converge, and vice versa. It is of course noted that in practical use, non-deterministic algorithms will typically be restarted numerous times, which will increase execution time proportionately.

In reporting their results, Celebi et al. make several recommendations:

- Random Partition and Random Centroids should not be used.
- Where data sets are very large and and applications are time-critical, a deterministic algorithm—in this case Var-Part or PCA-Part—should always be chosen.
- Conversely, where data sets are small enough, Bradley & Fayyad and the "greedy" variant of k-means++ are recommended, since these particular non-deterministic algorithms converge quickly enough that it is feasible to run them with many hundreds of restarts in a reasonable time frame.

A final interesting observation is that in scenarios where only an approximate clustering is required, the latter two algorithms and the two deterministic algorithms can potentially provide sufficiently successful initial clusterings that k-means may not need to be run at all.

2.1.5 Zahra et al. 2015

Zahra et al. [44] compare 20 initialisation algorithms within the context of exploring the use of a k-means clustering-based algorithm in recommender systems. Recommender systems guide users through overwhelming quantities of data towards resources which might be of particular interest. A familiar application of this is seen on e-commerce websites where products are recommended based on previous purchases.

The algorithms covered include Random Centroids (described in Section 2.2.2) and k-means++ (Section 2.2.10), whilst the remaining 18 are novel. The algorithms are run against five data sets including book and movie ratings data, and music listening data from Last.fm, with the aim of clustering users.

Given the specific context, and that so many of the algorithms are novel, the paper is perhaps not exactly to be described as a survey, and results must only be generalised with some caution. However, since it empirically compares k-means initialisation algorithms, there is clear relevance to the current work.

Algorithms include several which apply uniform, hypergeometric or Poisson distribution to the data and then use the Random Centroids approach to select initial centroids. Further algorithms exploit the concept of "power users", for example $\mathit{KMeansPlus}^{\mathit{Power}}$, which selects as initial centroids K users based on the number of ratings a user has provided.

Results are explored in great detail, and the analysis is highly specific to the recommender system problem, so interested readers are directed to the original paper. That said, the results of two algorithms, $\mathit{KMeansPlus}^{\mathit{ProbPower}}$ and $\mathit{KMeansPlus}^{\mathit{LogPower}}$ tend to stand out in terms of criteria such as within-cluster similarity and Mean Absolute Error. These two algorithms again exploit the "power user" concept, both identifying the most "powerful" user as the first initial centroid, and selecting as subsequent centroids users with increasing probability based on various measures of dissimilarity to selected centroids. This may suggest that incorporating knowledge of the problem domain into the design of initialisation algorithms could be an interesting avenue for further research.

A further conclusion presented is that while a k-means-based approach can bring cost savings and efficiency enhancements to recommender systems, the suitability of a hard partitioning approach to clustering in a scenario where users may have complex, diverse opinions may be questioned. For this reason, fuzzy approaches such as Fuzzy C-means (FCM) are often used [44, p. 182]. It is demonstrated that improved initial centroid selection can yield even greater improvements in clustering performance for FCM than for k-means.

2.1.6 Do Carmo Nicoletti & de Oliveira 2019

Do Carmo Nicoletti & de Oliveira [45] present an empirical evaluation of five k-means initialisation algorithms, expanding on their previous work in [46]. The algorithms covered include k-means++ (Section 2.2.10), Single Pass Seed Selection (Section 2.2.12), and Khan & Ahmad's Cluster Center Initialization Algorithm [47].

In all, 14 data sets are used. Seven of these are real-world data sets, including six sourced from the UCI Machine Learning Repository [31], and the Ruspini data set found in [48]. The remaining seven are synthetic data sets sourced from prior publications including [49, 50] and from online sources.

Experiments are run against each data set using 20 restarts for the non-deterministic algorithms. Three cluster validity indices (CVIs) are employed, namely the Dunn index, Silhouette index and Rand index, which is the precursor to ARI (Section 3.2), and not corrected-for-chance.

Results are presented and discussed individually in some detail for each data set, k-means with Random Centroids (Section 2.2.2) being used as a baseline for comparison. In concluding, do Carmo Nicoletti & de Oliveira observe that no one algorithm clearly emerged as the most successful, but that Cluster Center Initialization Algorithm and Single Pass Seed Selection generally showed good results.

An interesting further observation is that results from the three CVIs do not always agree. For example, cases are identified in which Rand index indicates a relatively successful cluster recovery rate, and yet Dunn, an internal CVI, reports that the quality of clustering is low. This seems to occur in particular where clusters are not particularly dense or spherical.

2.1.7 Fränti & Sieranoja 2019

Fränti & Sieranoja [51] seek to investigate some of the main factors—specifically data set characteristics—which can lead to the deterioration of k-means performance, and the extent to which better initialisation can alleviate this deterioration.

Nine algorithms are used in the experiments including Random Centroids, Random Partition, k-means++ and Bradley & Fayyad. The remaining five might conceivably be described as novel, each having been developed with the intention of exemplifying distinct styles of initialisation algorithm. The styles are referred to by the names Maxmin (for example Erisoglu, Calis & Sakallioglu, Chiang & Mirkin [52]), Sorting ([53, 54]), Projection-based ([55, 56]), Luxburg [13] and Split ([57]).

As with Zahra et al. above, the specific aims and the choice of algorithms might raise the question of whether the Fränti & Sieranoja work may strictly qualify as a survey. However, it does empirically compare k-means initialisation algorithms, and furthermore we find the exploration of how far the performance of k-means may be improved by the choice of initialisation to provide highly relevant context for the current work.

The experiments make use of a suite of data sets termed "clustering basic benchmark", developed by Fränti & Sieranoja and introduced in [21]. The data sets vary in several characteristics, specifically: number of clusters K; dimensionality V; cluster overlap; structure (termed "Birch", as two data sets from [58] are included); and balance, in other words the relative number of data points per cluster. This last, we term "cardinality" in this work.

The clustering basic benchmark is designed to offer a controlled environment consisting of data sets which are known to be able to be "solved"—in other words correctly clustered—using SSE as the objective function. Thus any clustering failures (defined in [51] as the inability to find the correct clustering within 5,000 restarts) must be ascribed to the choice of clustering algorithm rather than of objective function. To further this aim, the suite is designed to provide a sufficiently challenging clustering problem whereby only very good algorithms will successfully locate the correct centroids.

The experiments presented are rigorous, and the detailed results certainly make for interesting reading: in general, the choice of k-means initialisation is deemed to have limited influence on the clustering performance of k-means when compared to simply performing more restarts of a random initialisation. The main exception to this finding is the scenario whereby clusters are clearly separated, in other words display little or no overlap. With more overlap, k-means itself displays good performance, but as overlap decreases, the choice of initialisation becomes crucial. In such cases, Maxmin, Luxburg and Split perform the best, in extreme cases solving the data sets without requiring k-means iterations at all. The implication here is that improved performance is achieved by employing a more appropriate algorithm altogether, rather than the choice of k-means initialisation.

2.2 Algorithms covered

In this section we present each of the k-means initialisation algorithms explored in our experiments. For each algorithm we offer a brief summary, noting where possible the original authors' motivations for the design of their algorithms. The summary is followed where appropriate by a concise formal listing of the steps of the algorithm. Having had to track down and decipher the published algorithms ourselves, we believe that bringing all of the algorithms together in one place and presenting them in a consistent, formal manner is a valuable contribution to the literature.

A further fundamental motivation for this section is that, in order to ensure the reproducibility of our work, we state any assumptions that were made or arbitrary decisions that had to be taken in reproducing the published experiments. This latter includes, for example, values chosen for any parameters required, and the reasons motivating the selection of values for said parameters.

Finally, in this section we report any difficulties which were encountered in running the algorithms against the many data sets used in our experiments, since in not all of the 1,874,708 individual experimental runs was it possible successfully to find results[1].

2.2.1 Random Partition

This initialisation is the simplest and almost certainly the oldest initialisation strategy there is, dating from Forgy's 1965 work on k-means [15]. This method involves partitioning the data set entirely at random, effectively by assigning each data point $x_i \in X$ a uniformly random number from $1, ..., K$. Clearly there is no intelligence or logic motivating the design of this algorithm, and initial clusters will display no homogeneity whatsoever, except by coincidence. This seems to contradict the inherent aim of clustering. That being said, the minimal computational overhead involved may in many cases be desirable.

```
Algorithm 2: Random Partition
  Input: Data set $X$; number of clusters $K$
  Output: Initial centroids
    1. Assign each $x_i \in X$ to a cluster $S_k \in S$, chosen uniformly at
       random.
    2. The initial centroids are the component-wise means of each
       cluster.
```

As a non-deterministic algorithm, it is conventional to re-run or "restart" the algorithm many times, selecting the initial centroids leading k-means to the most successful cluster recovery from all restarts. It is common to see 10-50 restarts used, while at the upper extreme [24, 25] recommend 5,000 or more restarts where this is practically feasible. As discussed in Section 3.3, we perform 50 restarts for each non-deterministic algorithm we experiment with.

Despite its rather basic nature, we include this algorithm as the original

[1]The precise numbers of unsuccessful runs are shown in Table C.1 of Appendix C.3.

k-means initialisation, and as a baseline against which more elaborate algorithms may be compared. Indeed, subsequent initialisation algorithms are often compared against Random Partition in their original publication.

2.2.2 Random Centroids

This is another "random" approach, but in this case rather than all data points being assigned to random clusters, K points are selected uniformly at random from the data set. Given data with reasonably well-defined clusters, it is easy to see that randomly-selected data points are more likely to fall within more densely-populated areas, and therefore clusters. That said, the algorithm offers no protection from two or more initial centroids being selected from the same densely-populated area of the data, nor does it provide any guarantee that outlying data points will not be selected as initial centroids.

This approach is often attributed in the literature to Forgy, though this is refuted by [39] and in particular by MacQueen in 1967 [14, p. 294], who both explicitly state that that work employed Random Partition. The survey by Celebi et al. (Section 2.1.4) attributes Random Centroids to MacQueen.

Random Centroids is used as the initialisation strategy for Continuous k-means [59], and [25] attributes the method to that work. However, as Continuous k-means comprises both an initialisation and a modification to the update step (Step 3) of k-means itself, in this work we consider only the initialisation.

As with Random Partition, this algorithm is often used a basis for comparing novel k-means initialisation algorithms, so we find it important to include it in this work. As with all non-deterministic algorithms, we perform 50 restarts.

2.2.3 Milligan

Milligan 1980 [60] seeks to examine "the effect of six types of error perturbation on fifteen clustering algorithms", but in doing so provides an interesting side-by-side comparison of the performance of the algorithms themselves. The algorithms used include hierarchical algorithms such as Single Link, Weighted Average and Ward's Minimum Variance [61], along with methods attributed to Forgy [15] and MacQueen [14].

It is found that k-means can produce "excellent" cluster recovery performance when initial seeds are obtained from hierarchical clustering methods [60, p. 339]. In the same year, Milligan & Isaac [62] find Ward's method to give generally good results when clusters are well separated. Thus, both Peña & Lozano in their 1994 survey (see Section 2.1.1) and Steinley & Brusco in their 2007 survey (see Section 2.1.3), suggest using this as an initialisation for k-means. Utilising Ward's method is therefore the approach we adopt in our experiments.

Given a data set X of cardinality N, Ward's algorithm begins with a clustering containing N single data points. It then iteratively merges the two clusters with the lowest Ward distance, given by

$$d_w(S_k, S_l) = \frac{|S_k||S_l|}{|S_k| + |S_l|} d(c_k, c_l), \tag{2.1}$$

where S_k, $S_l \in S$, with c_k and c_l being their centroids, respectively. The above identifies the two clusters, the merging of which leads to the lowest increase in within-cluster variance. This process is repeated until the data is combined into K clusters. To select initial centroids for k-means from the output of such a hierarchical algorithm, we simply take the component-wise mean of each of the clusters found. The steps of this algorithm are shown as Algorithm 3.

```
Algorithm 3: Milligan
  Input: Data set $X$; number of clusters $K$
  Output: Initial cluster centres

    1. Set $S = \{S_1, S_2, ..., S_N\}$, so that each cluster contains one data
       point of $X$.

    2. Identify $S_k$, $S_l \in S$, the two clusters with minimum (2.1). Merge
       $S_k$ and $S_k$. Repeat this step until $|S| = K$.

    3. Return the centroids of each cluster in $S$.
```

2.2.4 Katsavounidis, Kuo & Zhang

Katsavounidis, Kuo & Zhang (KKZ) describe a "New Initialization Technique for Generalized Lloyd Iteration" [63]. This is a deterministic algorithm designed to maximise the minimum distance between candidate initial centroids and those already selected, the motivation being that data points far apart from each other are likely to fall into different clusters.

The algorithm begins by calculating the Euclidean[2] norm

$$||x_i||_2 = \sqrt{\sum_{v=1}^{V} x_{iv}^2} \tag{2.2}$$

of each data point, being the Euclidean distance between that point and the zero vector. We note therefore that this algorithm implicates a data normalisation process ensuring the component-wise mean over all $x_i \in X$ is zero (which we adopt, and describe in Section 3.1.3). The data point with the greatest norm is chosen as the first initial centroid and subsequently, for each point $x_i \in X$, the distance between x_i and its closest already-chosen initial centroid is calculated:

$$d_i = \min_{c_k \in C} d(x_i, c_k). \tag{2.3}$$

The data point with the greatest value of d_i is chosen as the next initial centroid, and the latter step iterates until a total of K initial centroids are found. This is what has been termed a "Maxmin" algorithm. As such, KKZ has been compared [39] to similar algorithms such as [65], the key difference being that the latter chooses the first centroid arbitrarily. The steps of the algorithm are formally presented in Algorithm 4.

As can be seen, the KKZ algorithm requires no inputs or parameters beyond the data set X and the number of clusters K, which can be considered a positive characteristic of, or even fundamental requirement for, a k-means initialisation algorithm [51]. On the other hand, due to the Maxmin strategy, it is known to be susceptible to influence by outlying data points [64, 66].

In [63], the KKZ algorithm is tested against three data sets, specifically the Baboon, Lena and Boat images from the USC-SIPI image database [67]. Results are presented in comparison with one other algorithm, a splitting algorithm from [68], which is described as the "primary competitor" to

[2]While [63] does not explicitly specify *which* norm to use in Step 1, the documentation for the R `kkz()` function [64] states that it uses squared Euclidean norm, so we choose not to disagree.

Algorithm 4: Katsavounidis, Kuo & Zhang

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
1. Set $C = \emptyset$. Identify the data point $\underset{x_i \in X}{\operatorname{argmax}} ||x_i||_2$ using (2.2),
   copy its values to $c_1$ and add $c_1$ to $C$
2. Identify the data point $\underset{x_i \in X}{\operatorname{argmax}}\ d(x_i, c_1)$, copy its values to $c_2$
   and add $c_2$ to $C$.
3. Identify $\underset{x_i \in X \wedge x_i \notin C}{\operatorname{argmax}}\ d_i$ using (2.3), copy its values to a new
   centroid $c_k$ and add $c_k$ to $C$.
4. If $|C| < K$ go to Step 3.
```

KKZ. The results are formalised as Mean Squared Error (MSE) and show an improvement over the splitting algorithm in all cases. The improvement appears to become more pronounced as either or both of the number of clusters K and the number of data points N increases.

2.2.5 Bradley & Fayyad

Bradley & Fayyad introduce an algorithm with the intention of "Refining Initial Points for k-means clustering" [69].

There are two main stages to this algorithm: firstly, J random subsamples of the data set are chosen and clustered using a slightly modified version of k-means. The modification is simply that once the clustering is complete, if any of the K clusters is found to have no members—which can easily happen when clustering small subsamples of the data—the centroids of those clusters are discarded, and the data points which are furthest from their assigned centroids are chosen to replace them.

Once this process is complete, it is likely that more than K candidate initial centroids have been found, indeed up to $K \times J$ candidate centroids may be found. These candidate centroids are in turn clustered using k-means in a step termed "smoothing".

Due to the fact that it works on subsamples of the data, this algorithm is claimed to be particularly applicable to very large data sets. Further, given a Gaussian distribution, the random subsampling will tend towards finding data points close to the modes of the data, in other words within meaningful clusters. That being said, Bradley & Fayyad acknowledge that the initial subsampling may well introduce noise in the form of outlying data points, and this is the reason for the "smoothing" procedure.

A notable aspect, and it could be argued drawback, of this algorithm is its requirement that several parameters be set, introducing an unfortunate dependency on a human being to make certain decisions. That being said, Bradley & Fayyad do make strong recommendations for the values of said parameters based on their experiments, which are detailed below and corroborated by [25].

The parameters in question are: the number of subsamples J; the size of the subsamples n; and a set of tentative initial centroids, or "starting point" SP. Bradley & Fayyad's findings suggest a value of 10 for J, with n being $\frac{N}{J}$, in other words one tenth of the data set. The starting point can in theory be any set of K initial centroids, though Bradley & Fayyad suggest a random initialisation, which we adopt, whilst acknowledging that this effectively makes the algorithm non-deterministic.

Algorithm 5: Bradley & Fayyad

```
Input: Data set $X$; number of clusters $K$; number of subsamples $J$; sample size $n$; set of initial centroids $SP$
Output: Initial centroids
    1. Set $X_t$ to be a subsample of $X$ such that $|X_t| = n$, for $t = 1, 2, ..., J$.
    2. Set $C'_t$ to be the set of $K$ centroids generated by a modified version of $k$-means (see Section 2.2.5) on $X_t$, initialised with $SP$, for $t = 1, 2, ..., J$.
    3. Set $C' = \bigcup_{t=1}^{J} C'_t$.
    4. Set $C_t$ to be the set of $K$ centroids generated by $k$-means on $C'$, initialised with $C'_t$, for $t = 1, 2, ..., J$.
    5. Set $C$ to be the set of centroids $C_t$ which has the lowest criterion output (1.2).
```

In [69], the algorithm is tested against several data sets. Firstly, data sets including Iris from the UCI Machine Learning Repository [31] are used. The results are reported as being "of no interest" as the data sets are "too easy", since they are small enough that multiple restarts of a random initialisation are feasible and there is no need for a more scalable algorithm in these cases. Results from synthetic data sets and larger real-world data sets, including Image Segmentation and the Reuters Information Retrieval data set, appear to be more promising in terms of both "information gain" and "distortion". However, results are compared only with a random initialisation.

2.2.6 Global *k*-means

Likas, Vlassis & Verbeek [70] note that at the time of their writing (2003), despite several attempts at addressing the problem of k-means being highly dependent on its initialisation, no technique had yet gained wider acceptance than a random initialisation with multiple restarts.

The original authors therefore introduce an algorithm named "Global k-means". This is a popular and interesting algorithm with ambitious aims, in that as well as finding K good initial centroids—as with other k-means initialisation algorithms which we discuss—it may be able to find good initial centroids for each of $1, 2, ..., K-1$ partitioning solutions. The algorithm is deterministic and does not require additional parameters to be set, beyond the usual inputs, being the number of clusters K and the data set X.

Global k-means proceeds by finding the optimal centroid for the case where $K = 1$, which will clearly be the component-wise mean of the entire data set. The second initial centroid is found by executing k-means using each $x_i \in X$ as the second initial centroid. The clustering that leads to the least value for the objective function (1.2) is deemed to be the solution for this case. The process repeats, running k-means using the centroids from the $K-1$ solution and x_i as the initialisation, until the pre-defined number of centroids are found.

The original authors acknowledge [70, p. 452] that in its original formulation, the algorithm is computationally complex, requiring N executions of k-means for each value $k = 1, ..., K$, a concern echoed in the survey by Celebi (Section 2.1.4).

To mitigate this, [70] offers two potential methods for speeding up execution, albeit whilst possibly compromising cluster recovery performance to a small extent. These are the "fast" Global k-means algorithm, and an ini-

Algorithm 6: Global k-means

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
  1. Set $c_{1v} = N^{-1} \sum_{x_1 \in X} x_{iv}$ for $v = 1, 2, ..., V$, and $C = \{c_1\}$.
  2. For each $x_i \in X$, run $k$-means with initial centroids $C \cup \{x_i\}$.
     Set a new centroid $c_k = x_i$ for that $x_i$ leading to the lowest
     (1.2). Add $c_k$ to C.
  3. If $|C| < K$, go to Step 2. Otherwise output $C$.
```

tialisation using k-d trees. In this work, however, we concern ourselves only with Global k-means in its initial formulation, as we are primarily interested in cluster recovery performance.

In [70], Global k-means is tested against three data sets: real-world data sets, namely Iris and Image Segmentation; and a synthetic data set sourced from [71]. The results seem to show that Global k-means provided a cluster recovery at least as successful as k-means with a random initialisation and N restarts (where N is the number of data points) on all occasions. Further, Likas, Vlassis & Verbeek claim the clustering solutions gained from Global k-means to be “experimentally optimal”, seemingly implying that the global optimum may be found. However, this has since been proven not to be the case [25, 72].

2.2.7 Yuan et al.

Yuan et al. present “A new algorithm to get the initial centroids”, with the intention of identifying initial centroids consistent with the distribution of the data [73].

The algorithm works by first calculating the proximity matrix of Euclidean distances between each point in the data set, and choosing the two closest points as the initial members of a cluster A_m. Subsequently, the nearest points to the cluster A_m in X are added until the cardinality of A_m reaches a pre-defined threshold. The data points in A_m are removed from X and the process repeated until K clusters have been found.

In addition to the number of clusters K, and the data set X, the algorithm requires one further parameter, named α. This is used in the calcula-

Algorithm 7: Yuan et al.

```
Input: Data set $X$; number of clusters $K$; factor $\alpha$
Output: Initial cluster centres
  1. Set $m = 1$.
  2. Set $A_m = \{x_i, x_j\}$, where $x_i, x_j \in X$ and $\forall x_t, x_\tau \in X$,
     $d(x_i, x_j) \leq d(x_t, x_\tau)$. Set $X = X \setminus A_m$.
  3. Set $x_\iota = \operatorname{argmin}_{x_\iota \in X} \min_{x_t \in A_m} d(x_\iota, x_t)$. Add $x_\iota$ to $A_m$, and remove $x_\iota$
     from $X$. Repeat until $|A_m| = \alpha \frac{N}{K}$.
  4. If $m < K$, set $m = m + 1$ and go to Step 2.
  5. Return the component-wise mean of $A_m$ (for $m = 1, 2, ..., K$) as
     an initial centroid.
```

tion of the "threshold" discussed above, which is defined as $\alpha \times N/K$, where N is the number of data points in X, and $0 < \alpha \leq 1$. With the proviso that the value of α may be dependent on the data set, the original authors found the value of 0.75 to be a good choice, and so we adopt this in our experiments.

The original authors run the new algorithm against four real-world data sets: Wine, Iris, Balance Scale and Car, and results compared to "standard" k-means—which we take to be a random initialisation—with 10 restarts, leading to a total of 40 experimental runs. Measured using the Accuracy Score CVI, the new algorithm is reported to display better clustering performance in 38 of those 40 cases. Needless to say, as a deterministic algorithm, its results are also stable.

We do note that the results are not adjusted for chance, and so in our experiments we use an accuracy measure that is adjusted for chance, and perform 50 restarts of all non-deterministic initialisation algorithms.

2.2.8 Hand & Krzanowski

Hand & Krzanowski [26] introduce a non-deterministic simulated annealing based algorithm, where k-means is run repeatedly and, with each iteration, "perturbations" are introduced. In other words, data points are moved at random between clusters before k-means is again run.

The intention is to introduce the possibility that the search process will be moved to a different part of the feature space at any stage of the process, and so may find a pathway to the globally optimum solution rather than a locally optimum one. Accordingly, with each iteration, the probability of perturbation is decreased as it is hoped that the algorithm begins to approach the global optimum.

Algorithm 8: Hand & Krzanowski

```
Input: Dataset $X$; number of clusters $K$; probability $\alpha$; learning
  rate $\beta$; iterations threshold $T$
Output: Initial centroids
  1. Run $k$-means on $X$, generating a clustering $S = \{S_1, S_2, ..., S_K\}$
     with a criterion output $\text{SSE}_0$ measured with (1.2). Set $t = 1$.
  2. Perturb each $S_k \in S$ by moving each $x_i \in S_k$ to a different
     (arbitrary) cluster with probability $\alpha$. Repeat $k$-means, leading
     to $\text{SSE}_t$.
  3. Set $t = t + 1$, and $\alpha = \alpha\beta$. Stop if either SSE stabilises, or if
     $t = T$. Otherwise go to Step 2.
```

The steps of Hand & Krzanowski's algorithm are formally presented in Algorithm 8). It can be seen that the algorithm depends upon the setting of three parameters. The first, α, is the probability of "perturbation" at each iteration, in other words the likelihood of each data point being moved to a different cluster. β is a multiplier used to reduce α on each iteration, thereby reducing the likelihood of perturbation occurring as SSE decreases. Finally, T is the maximum number of iterations.

Whilst the introduction of parameters may be considered a drawback, Hand & Krzanowski unambiguously state the values chosen, and we follow their recommendations, setting $\alpha = 0.3$, $\beta = 0.95$, and $T = 100$. In Step 3, the SSE is deemed to have stabilised (indicating the algorithm has converged) if it is unchanged in 10 iterations.

In [26] several experiments are carried out using the S-PLUS default initialisation, which uses the result of a group-average hierarchical clustering as its initial centroids, as a baseline for comparison.

The first simulation investigates whether cluster recovery improvements can be made by applying Hand & Krzanowski's iterative refinement algorithm to the S-PLUS default implementation. Both are run against 540 synthetic data sets. In this simulation the results from the S-PLUS default are shown to be improved upon by the iterative refinement in 284 cases, or 52.6%.

The second simulation investigates whether the cluster recovery of the S-PLUS default can be outperformed by a random initialisation with 20 restarts. It is reported that the best of 20 random restarts outperforms the S-PLUS default in 62.3%, an improvement on the iterative refinement.

Finally, the original authors experiment with a combination of the best of 20 restarts plus the iterative refinement, which is found to improve upon the default by 63.7%.

Whilst the differences in clustering performance between the two most successful approaches do not appear to be significant, both show a marked improvement over the S-PLUS default, and so the original authors conclude that reliance on the default options of software packages for clustering is not necessarily wise.

2.2.9 Intelligent *k*-means

Intelligent k-means, introduced by Mirkin 2005 [1], is a successful k-means initialisation algorithm [52] which can be used to identify both the number of clusters, K, and a set of good initial centroids deterministically. It identifies anomalous clusters in the data, selecting their centroids as initial centroids for k-means.

The Intelligent k-means initialisation algorithm (Algorithm 9) is comprised of two main routines, the *Anomalous Pattern* and the *Iterated Anomalous Pattern for Intelligent k-means.* The latter seeks to find good initial centroids, and employs the former to do so.

The Anomalous Pattern proceeds by attempting to find the group of data furthest removed from a given reference point in the data. In its application to clustering, the reference point is taken to be the "centre" of the data: if the data is normalised to have zero mean—a process which we adopt in our experimental setup (see Section 3.1.3)—this will the "origin" or zero vector, but may otherwise be the component-wise mean of the data set, assuming the distance measure used is Euclidean distance (1.1). The anomalous point therefore is the data point the greatest distance from the reference point.

Since only distances from a single reference point are required to be calculated, the Anomalous Pattern avoids the computational overhead of calculating a proximity matrix of the entire data set, as some k-means initialisation algorithms do, for example Single Pass Seed Selection (Section 2.2.12) and Yuan et al. (Section 2.2.7).

```
Algorithm 9: Intelligent $k$-means
  Input: Data set $X$; number of clusters $K$; cardinality threshold $\theta$.
  Output: Initial centroids
    1. Set $C = \emptyset$, and $c_c$ to be the component-wise mean over all
       $x_i \in X$,
    2. Set $c_t$ to be equal to the data point $x_t \in X$ that is the farthest
       from $c_c$ as per (1.1).
    3. Run $k$-means on $X$ using $c_c$ and $c_t$ as initial centroids. Do not
       allow $c_c$ to move in the centroid update step of $k$-means. This
       will form clusters $S_c$ and $S_t$.
    4. If $|S_t| \geq \theta$, add $c_t$ to $C$. In any case remove all $x_i \in S_t$ from $X$.
       If there are still data points in $X$ go to Step 2.
    5. Run $k$-means on the original data set setting $K = |C|$, and using
       all $c_k \in C$ as initial centroids.
```

Intelligent k-means iteratively applies the Anomalous Pattern to identify one cluster S_t and its centroid c_t at a time, by performing alternating minimisation using

$$W = \sum_{x_i \in X} d(x_i, c_t) + \sum_{x_i \in X} d(x_i, c_c),$$

where c_c is the component-wise average over all $x_i \in X$.

Besides the number of clusters K and the data set X, Intelligent k-means accepts one further parameter, θ, being the minimum cardinality required for a cluster to be added to C. The background for this is that, besides identifying good initial centroids, the algorithm can be used to address the problem of discovering the number of clusters present.

In all our experiments we make the assumption that the number of clusters K is known, and in any case it would be unfair for us to provide more information (in this case K) to the other algorithms than we do to Intelligent

k-means. Hence, in our experiments we set $\theta = 1$ and of the four stopping conditions suggested in [1], we deem the algorithm to have converged when all data has been clustered.

This approach can lead to a situation whereby the number of clusters found by Intelligent k-means is not equal to the pre-defined K. In a several cases, the number of clusters found was too small, and we were not able to obtain results for these cases. This, perhaps not surprisingly, becomes more common as K increases, and the number of instances of this occurring can be seen in Table C.1 of Appendix C.3.

Where the number of clusters found is greater than the pre-defined K—which is likely when setting $\theta = 1$—we select the initial centroids in two different ways:

1. Select the K initial centroids identified by intelligent k-means that have the highest cardinality for S_t. These may be assumed to be the K most representative initial centroids. This approach is suggested in [74].

2. Select the first K initial centroids identified by intelligent k-means. These could be considered to be the centroids the K most anomalous clusters.

The former method we refer to as Intelligent k-means (Cardinality), and the latter we refer to as Intelligent k-means (First).

2.2.10 *k*-means++

Arthur and Vassilvitskii introduce the k-means++ algorithm [23]. To this day, this remains arguably the single most popular k-means initialisation algorithm, and in fact is the default initialisation in MATLAB and Python's scikit-learn library. As with Random Centroids, the first initial centroid is selected uniformly at random from X. However, further centroids are selected with probability

$$P(x_i) = \frac{D(x_i)^2}{\sum_{x_t \in X} D(x_t)^2}, \tag{2.4}$$

where $D(x_i)$ is the shortest distance from a data point $x_i \in X$ to the nearest centroid already chosen. Algorithm 10 formally describes the steps of k-

means++.

Algorithm 10: k-means++

```
Input: Data set $X$; number of clusters $K$
Output: Initial centroids
  1. Select a data point $x_i \in X$ uniformly at random, and copy its
     values to $c_1$
  2. Select a further data point $x_i \in X$ with probability given by
     (2.4), and copy its values to a new centroid $c_k$.
  3. Repeat Step 2 until a total of $K$ initial centroids have been
     chosen.
```

As with Random Centroids, motivations for this strategy acknowledge that, given a Gaussian data distribution, the first centroid is relatively likely to be found within a higher-density area of the data, ideally corresponding to a meaningful cluster. Similarly, selecting subsequent initial centroids with increasing probability based on their distance from already-selected points introduces a tendency towards finding points in different clusters.

While k-means++ therefore addresses one of the shortcomings of Random Centroids identified above, due to the probabilistic nature of the the selection of subsequent centroids, it is of course still possible that outlying data points may be selected, leading to poor clustering performance. Therefore Arthur & Vassilvitskii perform 20 restarts per experiment. To remain consistent with our treatment of other non-deterministic initialisation algorithms, we perform 50 restarts.

2.2.11 Erisoglu, Calis & Sakallioglu

Erisoglu, Calis & Sakallioglu [75] present a deterministic algorithm to find initial centroids based on the concept of finding the two features which the original authors believe best describe the dataset over two axes.

The first or “main” axis is taken to be the feature with the highest value for the variation coefficient

$$cv_v = \left| \frac{\sigma_v}{\bar{x}_v} \right|, \qquad v = 1, ..., V \tag{2.5}$$

where σ_v and $\bar{x}_v$ are the standard deviation and mean of feature v over all $x_i \in X$, respectively. The second axis is taken to be the feature with the lowest absolute correlation with the main axis.

Once the data is reduced to two axes, the first tentative initial centroid is taken to be the data point furthest from the mean on the two axes, measured by Euclidean distance. Each subsequent tentative centroid is chosen as the data point with the greatest sum of distances—again, considering only the two features identified as the main and secondary axes—from the already-chosen tentative centroids. The intention, therefore, is to find centroids which are well separated from each other.

The process is repeated until K such centroids are found. Finally all data points are clustered around the tentative centroids—still considering only two features—and the component-wise means of the resulting clusters are deemed to be the initial centroids to be passed to k-means itself. The steps of the algorithm are shown in Algorithm 11.

```
Algorithm 11: Erisoglu, Calis & Sakallioglu
  Input: Data set $X$; number of clusters $K$
  Output: Initial centroids

    1. Identify the feature $v'$ as that with the highest coefficient of
       variation. Set $C = \emptyset$.
    2. Identify the feature $v''$, as that with the lowest absolute Pearson
       correlation to $v'$.
    3. Set $c_1 = \underset{x_i \in X}{\operatorname{argmax}}\, d(x_{iv'v''}, (\bar{x}_{v'}, \bar{x}_{v''}))$, and add $c_1$ to $C$.
    4. Set $c_2 = \underset{x_i \in X}{\operatorname{argmax}}\, d(x_{iv'v''}, c_{1v'v''})$, and add $c_2$ to $C$.
    5. If $|C| < K$, set a new centroid
       $c_k = \underset{x_i \in X}{\operatorname{argmax}} \sum_{c_l \in C} d(x_{iv'v''}, c_{lv'v''})$, and add $c_k$ to $C$. Repeat
       this step as necessary.
    6. Perform a clustering around $C$ using only the features $v'$ and $v''$.
       Return the component-wise means of the resulting clusters as
       the initial centroids.
```

A worked example is provided in [75], which we find helpful. The example uses the well-known Iris data set, and shows clearly the steps to finding the initial centroids, which we were able to reproduce.

The original authors measure the cluster recovery performance of their algorithm when run against five real-world data sets, including Iris, and measure the outcome using three CVIs: error percentage, Rand index, and Wilks' lambda. In all cases the proposed algorithm is show to outperform the mean of a Random Centroids approach with 10 restarts. Further results are presented which appear to show the proposed algorithm to outperform two further algorithms [47, 76], although results are presented using only error percentage.

In our experiments, we found it necessary to make two assumptions in order to reproduce the original authors' experiments. Firstly, [75] presents a equation for a correlation coefficient subtly different to Pearson's correlation coefficient, but we were only able to reproduce their results using Pearson's instead. Hence, we have adopted the latter in our experiments. Secondly, the variation coefficient (2.5) will lead to division-by-zero errors in cases where data sets are normalised to have zero mean, a process which we adopt (see Section 3.1.3). Therefore, since each of our features will have the same mean, we find it reasonable to use simply standard deviation as the variation coefficient.

A further consideration is that, given this algorithm calculates cumulative distances to all previously found centroids, it may select two nearby data points provided they have a large cumulative distance [51]. In our experiments, this algorithm sometimes even re-selected a previously found centroid, particularly where $K \geq 5$ (see Figure 4.9). Indeed, where $K = 20$ we were not able to successfully produce any partitions, with the final clustering step based on just two features frequently leading to empty clusters. The number of cases where successful clusterings were found is show in Table C.1 of Appendix C.3.

In general, we instinctively find the approach of selecting a hard number of features (exactly two in this case) to be somewhat fragile. We conjecture that the original authors were probably attempting to reduce complexity. However, there is no reason to believe exactly two features will always represent all characteristics of a given data set.

2.2.12 Single Pass Seed Selection

Pavan et al. propose a “robust seed selection algorithm for k-means type algorithms [66]. Single Pass Seed Selection is inspired by k-means++ and is in fact described by the original authors as an extension thereof, the key distinction being that the newer algorithm is deterministic, hence “single pass”. We find it important to clarify that whilst this algorithm may be abbreviated to SPSS, it is in no way related to the statistical software of that name.

The key modifications are twofold. Firstly, the very first initial centroid is not chosen uniformly at random from the data set. Rather, having calculated the proximity matrix between all data points, the point which has the lowest total distance to all other data points is selected. Subsequent initial centroids are identified based on ensuring a minimum distance from previously selected centroids.

Algorithm 12: Single Pass Seed Selection

```
Input: Data set $X$; number of clusters $K$
Output: Initial centroids

  1. Set $c_1 = \underset{x_i \in X}{\operatorname{argmin}} \sum_{x_t \in X} d(x_i, x_t)$
  2. For each $x_i \in X$ set $D(x_i) = \min_{x_i \in X} \sum_{x_k \in X} d(x_i, c_k)$.
  3. Set $y$ to be the the sum of distances of the $\frac{N}{K}$ nearest data
     points to the last centroid added to $C$.
  4. Find the index $l$, such that $\sum_{t=1}^{l} D(x_t)^2 \geq y > \sum_{\tau=1}^{l-1} D(x_\tau)^2$.
     Set a new centroid $c_k = x_l$, and add it to $C$.
  5. If $|C| < K$ go to Step 2.
```

Pavan et al. run the algorithm against 10 real-world data sets, compare the results against the output of k-means++ with 20 restarts. The original authors present the resulting cluster recovery performance of each algorithm measured using the Silhouette score CVI, along with the number of iterations required by k-means itself to converge subsequent to the initial centroids being selected.

According to the results discussed above, in most cases, Single Pass

Seed Selection appears to perform roughly as well as the most successful k-means++ restarts, and the results are of course stable, since the algorithm is deterministic. In particular, the number of iterations required by k-means appears noticeably reduced in cases where the data is of a higher dimensionality.

2.2.13 Hatamlou Binary Search

Hatamlou [19] introduces a deterministic binary search algorithm with the intention of identifying good initial centroids for k-means. In this algorithm each centroid $c_k \in C$ is initially selected from a different part of the data set X. The algorithm then optimises the location of each of these centroids, with respect to (1.2), by exploring around them. This optimisation process occurs iteratively until either a maximum number of iterations is reached, or, the centroids have converged. The algorithm, which requires no extra parameters beyond the number of clusters K and the data set X, is formally presented as Algorithm 13.

The original author validates the algorithm by experimenting on six real-world data sets sourced from the UCI Machine Learning Repository [31], including the well-known Iris, Wine and Glass data sets. The results presented in [19] are compared against those of seven other algorithms sourced from [77]. Success is measured using the output of (1.2), and the F-Measure. The results shown on the original paper are certainly good. Hence, we find it would be of interest to explore how this algorithm performs in a more comprehensive set of experiments.

2.2.14 Khan's Seed Selection Algorithm

Khan 2012 [3] presents an "initial seed selection algorithm" with two explicit aims beyond finding better initial centroids. It is intended firstly to require no extra parameters which would necessitate decisions being made by a human being, and secondly, to provide replicable results—in other words to be deterministic.

This is a relatively straightforward algorithm designed to infer a putative initial partitioning of the data by locating the widest $K - 1$ Euclidean distances between data points, yet based on a single feature. The data points delineated by these "gaps" are deemed to be members of the same initial

Algorithm 13: Hatamlou Binary Search

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres

  1. Set $SSM_v = \max_{x_{iv} \in X} x_{iv}$ and $g_v = \left( SSM_v - \min_{x_i \in X} x_{iv} \right) / K$, for
     $v = 1, 2, ..., V$.
  2. Set $c_{kv} = \min_{x_i \in X} x_{iv} + g_v(k-1)$, for $k = 1, 2, ..., K$ and
     $v = 1, 2, ..., V$.
  3. Assign each $x_i \in X$ to the cluster $S_k$ represented by its nearest
     centroid $c_k$, and calculate (1.2).
  4. Set $k = 1$ and $v = 1$.
  5. Set $c_{kv} = c_{kv} + SSM_v$, and recalculate (1.2).
  6. If there is no improvement: (i) if $SSM_v < 0$, set
     $SSM_v = -\frac{1}{2} SSM_v$, or (ii) if $SSM_v > 0$, set $SSM_v = -SSM_v$.
  7. If $v < V$, set $v = v + 1$, or if $v = V$ and $k < K$, set $k = k + 1$. In
     any of these two cases, go to Step 5.
  8. If the termination criterion is not met, go to Step 4.
```

clusters, with the intention of increasing the distances between points in separate clusters. The steps of the algorithm are shown here as Algorithm 14.

The decision to attempt to find an initial clustering based on a single feature of the data set is certainly interesting. Unfortunately no method is provided in [3] to select the feature to be used, so we conclude that this—contrary to the first stated aim of the algorithm—constitutes a parameter which must be supplied to the algorithm. Further, given no guidance on how to select this value, we have little option but to select it at random, thereby rendering the algorithm non-deterministic, contrary to the second aim stated above. Accordingly, we treat it as such in our experiments, and perform 50 restarts as with all other non-deterministic algorithms explored in this work.

In [3], the algorithm is tested against four real-world data sets and one synthetic dataset with normal distribution. Results are compared to those

```
Algorithm 14: Khan's Seed Selection Algorithm
  Input: Data set $X$; number of clusters K; initial feature for clustering $v'$
  Output: Initial centroids
    1. Sort the data points $\{x_{iv'} : x_i \in X\}$ in terms of increasing magnitude, such that $x_{1v'}$ and $x_{nv'}$ have the minimum and maximum magnitudes, respectively.
    2. Set $D_i = x_{(i+1)v'} - x_{iv'}$ for $i = 1, ..., n-1$.
    3. Sort $D$ in descending order without changing its indices. Identify the $K-1$ values of $i$ related to the $K-1$ highest values of $D$, leading to $(i_1, ..., i_{(K-1)})$.
    4. Sort $(i_1, ..., i_{(K-1)})$ in ascending order.
    5. The corresponding set of indices of data points $x_i \in X$ which are the lower bounds of clusters $S_1, S_2, ..., S_K$ are defined as $(i_0, i_1 + 1, ..., i_{K-1} + 1)$, where $i_0 = 1$.
    6. The centroid $c_k$ is calculated as the component-wise mean of the $x_i$ values falling within the upper and lower bounds calculated above.
```

from k-means++ (Section 2.2.10). Measured by the sum of squared distances criterion (1.2), cluster recovery performance appears to be better than k-means++ in the cases of just two out of five data sets. That being said, the original authors concede that this is not the primary aim of the algorithm. Running time is shown to be consistently improved over k-means++, being reduced by up to 89.11% in the case of the well-known Iris data set.

2.2.15 Onoda, Sakai & Yamada

Onoda, Sakai & Yamada [78] introduce two related deterministic k-means initialisation algorithms, one based on independent component analysis (ICA) and a second based on principal component analysis (PCA). The algorithms, which require no further parameters beyond the number of clusters K and the data set X, are presented formally as Algorithms 15 and 16.

In the experiments presented in [78], the algorithms outperform Ran-

```
Algorithm 15: Onoda, Sakai & Yamada ICA
  Input: Data set $X$; number of clusters $K$
  Output: Initial cluster centres
    1. Extract $K$ independent components $IC_1, IC_2, ..., IC_K$ from $X$.
    2. Select $K$ initial centroids $c_1, c_2, ..., c_K$, selecting $c_k = x_i \in X$
       with a minimum $\frac{IC_k x_i}{|IC_k||x_i|}$
```

```
Algorithm 16: Onoda, Sakai & Yamada PCA
  Input: Data set $X$; number of clusters $K$
  Output: Initial cluster centres
    1. Extract $K$ principal components $PC_1, PC_2, ..., PC_K$ from $X$.
    2. Select $K$ initial centroids $c_1, c_2, ..., c_K$, selecting $c_k = x_i \in X$
       with a minimum $\frac{PC_k x_i}{|PC_k||x_i|}$
```

dom Centroids, k-means++ and KKZ (for details on these algorithms, see Sections 2.2.2, 2.2.10 and 2.2.4, respectively), when the ratio between the number of features and the number of data points is less than 10. Also, their initialisation seems to be faster than k-means (with 100 restarts).

Given the use of independent component analysis or principal component analysis, it is unclear how this algorithm could produce a set of initial centroids if $V < K$. In our experiments we were therefore unable to produce results for such cases, and the number of successfully completed experimental runs may be seen in Table C.1 of Section C.3.

2.3 Summary

Whilst it would be impractical to implement and experimentally analyse every single k-means initialisation algorithm ever proposed, we hope that those included in this work provide a reasonable cross-section of the approaches that have been taken over several decades. We also hope that the historical context for this work, both as a survey and an empirical study, is clear, and that it is not unreasonable to suggest this may be the most extensive study of published k-means initialisation algorithms undertaken thus far.

We now turn to our experiments. We aim to perform an unbiased comparison of the algorithms discussed above, and so in Section 3 we present our experimental methodology and notes on our implementation, with the intention that our experiments should be reproducible.

3 Experimental Setup

In this section we detail the process employed to evaluate the k-means initialisation algorithms described in Section 2.2, including: the data sets against which the algorithms were run, both synthetic and real-world; the cluster validity index used to measure cluster recovery performance; and any further assumptions made or decisions taken.

Each of the algorithms was run against each of an extensive collection of 6,028 data sets, formally described in the following sections.

3.1 Data sets

We generated a total of 6,000 synthetic data sets, which are detailed in Section 3.1.1, and sourced 28 real-world data sets, detailed in Section 3.1.2. All data sets are labelled, allowing the use of external cluster validity indices such as adjusted Rand index (ARI) (Section 3.2) to measure cluster recovery performance.

3.1.1 Synthetic data sets

To provide a controlled environment for our experiments to exercise the covered algorithms, we generated a large number of synthetic data sets, with characteristics varying along several parameters. The exact configurations used are shown in Table 3.1.

Parameter	Value
Clusters (K)	2, 5, 10, 20
Features (V)	2, 10, 50, 100, 1000
Data points (N)	1000
Cluster cardinality	Uniform, random
Within-cluster standard deviation	0.5, 1, 1.5

Table 3.1: The configurations used to generate the synthetic data sets

Cluster cardinality is a description of how many data points belong to each cluster: in the “uniform” case all clusters contain the same number of data points, whilst in the “random” case the cardinality is uniformly

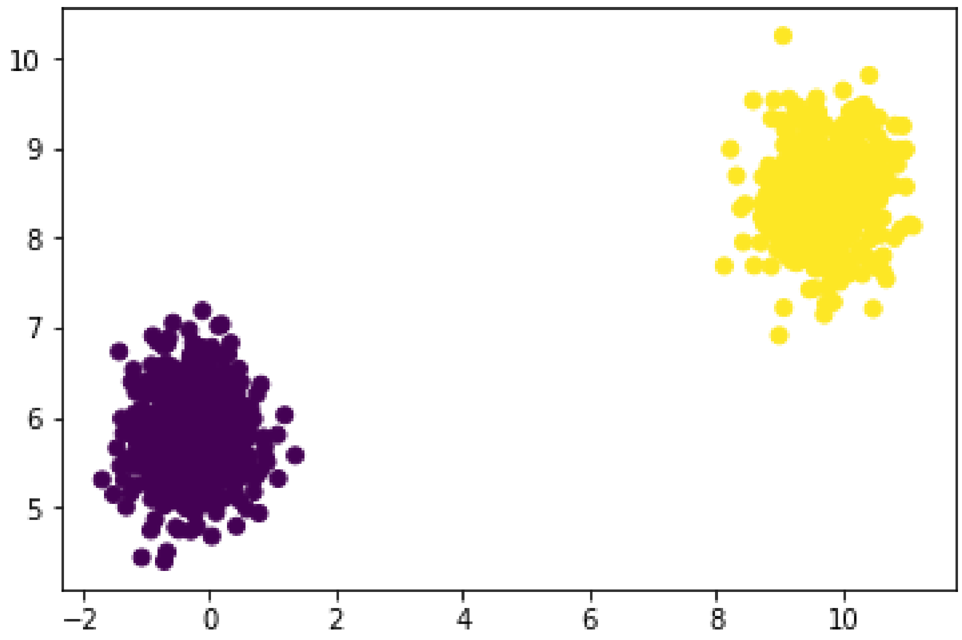

Figure 3.1: Visualisation of synthetic data with $V = 2$, $K = 2$, $\sigma = 0.5$

random, except that we set a minimum constraint that no cluster will contain fewer than 30 points, in other words 3% of the data set. Within-cluster standard deviation is varied with the intention of allowing us to experiment with data sets with greater or lesser overlap between clusters. Figures 3.1 to 3.4 are included to provide a visualisation of the data and its clusters in two dimensions. In the figures, as throughout this work, K is the number of clusters, V the number of features and σ the within-cluster standard deviation.

The values of the parameters in Table 3.1 were chosen with the aim of generating a wide variety of data set characteristics within practical constraints. Whilst many of the lower bounds are self-explanatory—for example, cluster recovery on a data set where $K = 1$ would be a trivial problem and of no interest—intervals and upper bounds are chosen on the understanding that it simply is not possible to run algorithms against data sets of every conceivable size and shape.

All data sets are composed of Gaussian clusters, in other words clusters convex in shape with density increasing towards the centroids, since this is the clustering problem addressed by k-means [32].

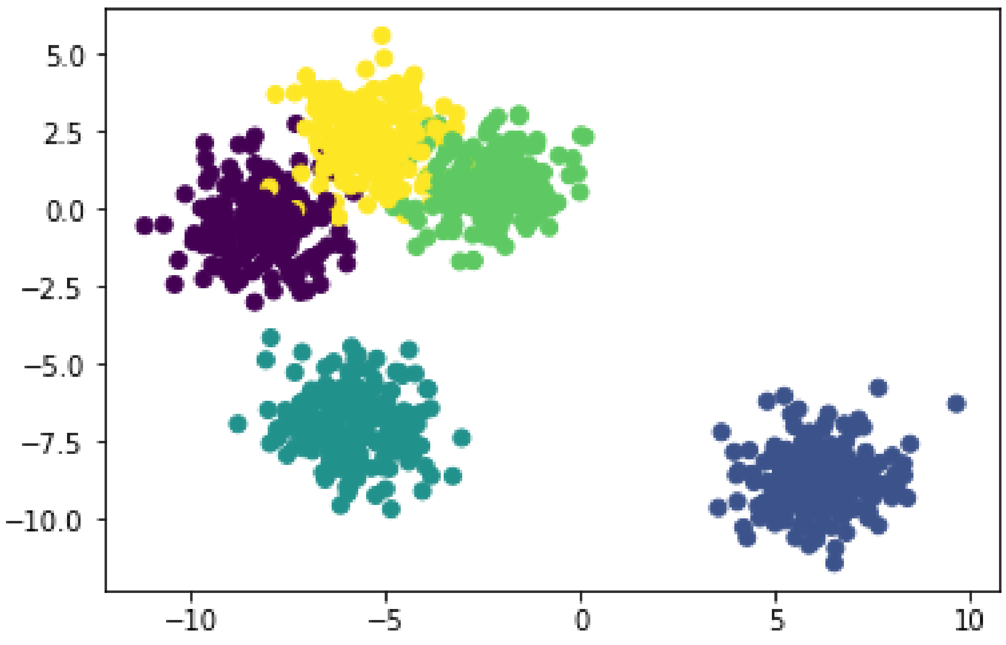

Figure 3.2: Visualisation of synthetic data with $V = 2$, $K = 5$, $\sigma = 1.0$

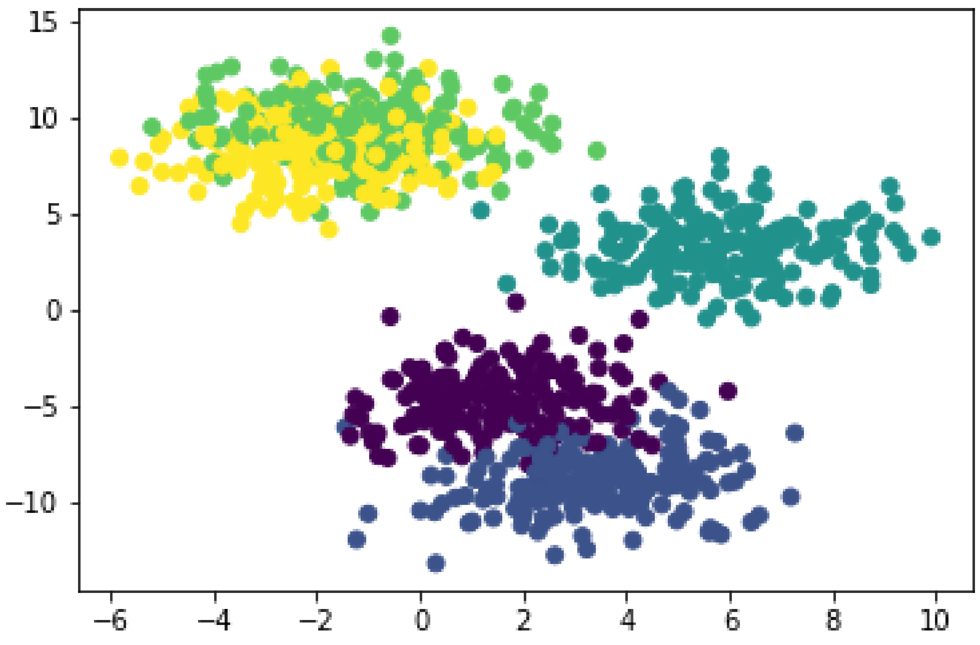

Figure 3.3: Visualisation of synthetic data with $V = 2$, $K = 5$, $\sigma = 1.5$

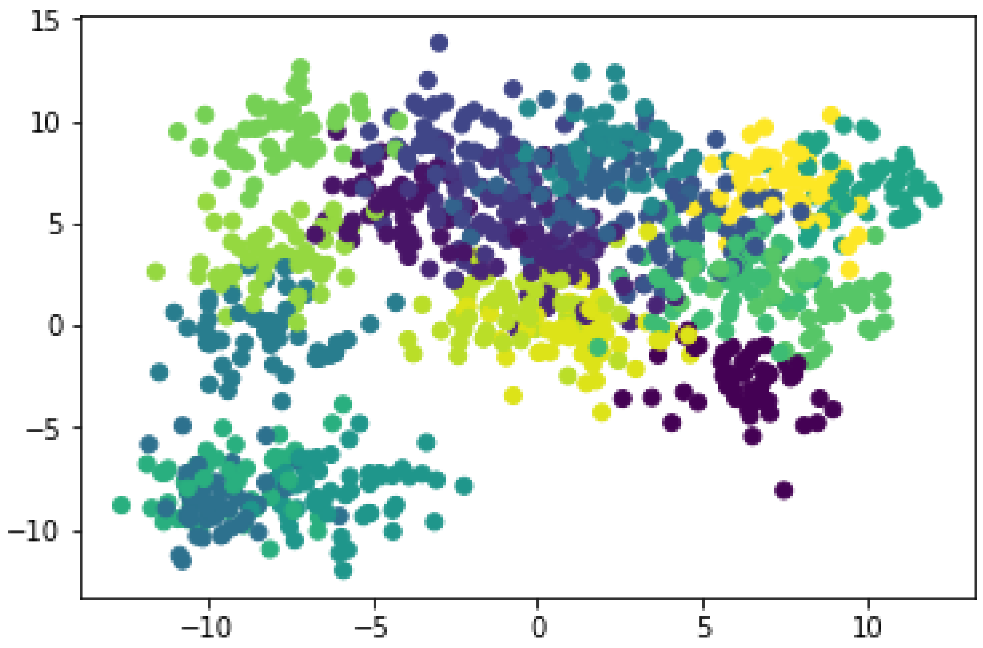

Figure 3.4: Visualisation of synthetic data with $V = 2$, $K = 20$, $\sigma = 1.5$

Since the synthetic data generation process is effectively one of random number generation, at least within the constraints stated, we generate 50 instances of each of the configurations, leading to a total of $4 \times 5 \times 2 \times 3 \times 50 = 6000$ synthetic data sets.

The tools used to generate the synthetic data sets were written in Python and utilised the `make_blobs()` function from the well-known scikit-learn library [18] to create isotropic Gaussian clusters. The tools we wrote to do so are freely available as part of the open-source software library described in Section 5.3.

3.1.2 Real-world data sets

We feel that our experiments would not be complete without the inclusion of real-world data sets. Therefore, 27 data sets were sourced from the UCI Machine Learning Repository [31], and the Chernoff Fossil data set was sourced from [48]. This provides us with 28 real-world data sets with varying characteristics which are detailed in Table 3.2.

The real-world data sets were specifically selected to be comprised of only continuous, numeric data since k-means, while using SSE (1.2) as the

Data set	Data points	Features	Clusters
Avila	10430	10	12
Blood Transfusion	748	4	2
Breast Cancer Wisconsin (Diagnostic)	569	30	2
Breast Cancer Wisconsin (Original)	683	9	2
Breast Tissue	106	9	6
Ecoli	336	7	8
Fossil	87	6	3
Glass Identification	214	9	6
HTRU2	17898	8	2
Haberman's Survival	306	3	2
Iris	150	4	3
Leaf	340	15	30
Letter Recognition	20000	16	26
Libras Movement	360	90	15
Musk 1	476	166	2
Musk 2	6598	166	2
Optical Recognition	3823	62	10
Page Blocks	5473	10	5
Parkinsons	195	22	2
Pen-Based Recognition	7494	16	10
Sonar all	208	60	2
Spambase	4601	57	2
Vehicle Silhouettes	846	18	4
Vertebral Column	310	6	3
Wine	178	13	3
Wine Quality (Red)	1599	11	6
Wine Quality (White)	4898	11	7
Yeast	1484	8	10

Table 3.2: The characteristics of the real-world data sets

objective function, is defined only in the context of data for which a mean can be calculated.

In terms of data cleaning, in a small number of cases rows with missing data were dropped, but in general we deliberately chose sets needing minimal cleaning to reduce the number of arbitrary decisions required to be made.

The tools we wrote to download and clean the data, as well as isolate the class labels for each data set, are made available as a further open-source software library described in Section 5.2.

3.1.3 Data normalisation

In data clustering, as with other exploratory data processing tasks, it is customary to normalise all data in advance. The general point of data normalisation is to reduce each feature to the same scale, thus preventing any specific feature outweighing others, particularly when we come to measure squared Euclidean distance as per (1.1).

Therefore, we adopt this approach. A detailed description of how normalisation was applied to our data sets may be found in Appendix B.

3.2 Cluster validity indices

In order to evaluate the cluster recovery performance of any algorithm, we need a way to measure that performance objectively. Numerous cluster validity indices (CVIs) exist, and we direct interested readers to an extensive survey of such indices found in [79].

In this work, we adopt the adjusted Rand index (ARI) [41]. This is a corrected-for-chance version of the Rand index [80]. The ARI between the clusterings S, S' is given by

$$ARI(S,S') = \frac{\sum_{kl}\binom{N_{kl}}{2} - [\sum_k \binom{a_k}{2}\sum_l \binom{b_l}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_k \binom{a_k}{2} + \sum_l \binom{b_l}{2}] - [\sum_k \binom{a_k}{2}\sum_l \binom{b_l}{2}]/\binom{N}{2}}, \tag{3.1}$$

where $N_{kl} = |S_k \cap S'_l|$, $a_k = |S_k|$, and $b_l = |S'_l|$.

ARI is an external cluster validity index, in that it evaluates clustering performance based on a pre-specified structure, in this case the class labels. This can be contrasted with an internal cluster validity index, for example SSE (1.2) which can only evaluate clustering performance based solely on the data itself [81]. ARI is found to be "highly recommended" by [82], [83] and [25], and is a viable option for us as we have class labels for all data sets used.

For completeness and interest, we additionally present the results of our experiments using an internal CVI, inertia. CVI is the k-means criterion itself, and the results and discussion can be found in Appendix C.1. In brief, we find a general agreement between the results of the two separate indices.

3.3 Implementation and assumptions

As is typical for k-means clustering, we assume that we know the number of clusters, K in advance. We acknowledge that this may not always be realistic in practice, and whilst discovering K is an interesting research challenge in itself, it is beyond the scope of this work. We direct interested readers to the many relevant publications, for example [52, 84, 85] and the references therein.

As some of the algorithms covered are deterministic and some non-deterministic, we perform 50 restarts of each non-deterministic algorithm. This may be considered generous as many practitioners often perform 20 or fewer restarts. However, in cases with fewer restarts, said practitioners tend to select the results from only the single restart yielding the most successful cluster recovery performance. In our experiments all results are recorded, allowing us to calculate the mean and standard deviation of the cluster recovery performance of each algorithm over multiple experimental runs.

From the summary details of each initialisation algorithm shown in Table 4.2, it can be seen that of the 17 algorithms covered in our experiments, six are non-deterministic with the remainder of course being deterministic. Given 6,028 data sets, and that we perform 50 restarts of the non-deterministic algorithms and a single restart of the remainder, there are a total of $(6028 \times 6 \times 50) + (6028 \times 11)$ or **1,874,708** individual experimental runs to perform.

The experiments were run on the CentOS Linux-based high performance computing cluster at the University of Essex [86], and the results are presented and discussed in the following section.

4 Results and Discussion

In this section we measure the performance of each of the initialisation algorithms discussed in Section 2.2 in two ways. Firstly, we measure cluster recovery by applying the adjusted Rand index (3.1) to compare the clustering produced by each algorithm on each data set against the true labels. Secondly, we measure time performance by counting the number of iterations k-means takes to converge under each initialisation. We do both of these on the synthetic and real-world data sets we experiment with.

To aid reporting, we assign an identifier to each algorithm, which we use in Figures 4.1-4.10, and throughout the text. The identifiers, along with some summary details concerning each algorithm, as shown in Table 4.2. In the table, D/ND is an abbreviation for "deterministic or non-deterministic".

Name	Identifier	D/ND	Year	Source
Random Partition	random_p	ND	1965	[15]
Random Centroids	random_c	ND	1967	[14]
Milligan	milligan	D	1980	[60, 62]
Katsavounidis, Kuo & Zhang	kkz	D	1994	[63]
Bradley & Fayyad	bradley	ND	1998	[69]
Global k-means	globalkm	D	2003	[70]
Yuan et al.	yuan	D	2004	[73]
Hand & Krzanowski	hand	ND	2005	[26]
Intelligent k-means (Cardinality)	ikm_card	D	2005	[1]
Intelligent k-means (First)	ikm_first	D	2005	[1]
k-means++	kmpp	ND	2007	[23]
Erisoglu, Calis & Sakallioglu	erisoglu	D	2011	[75]
Single Pass Seed Selection	singlepass	D	2011	[66]
Hatamlou Binary Search	hatamlou	D	2012	[19]
Khan's Seed Selection Algorithm	khan	ND	2012	[3]
Onoda, Sakai & Yamada ICA	onoda_ica	D	2012	[78]
Onoda, Sakai & Yamada PCA	onoda_pca	D	2012	[78]

Table 4.1: Summary of algorithms used in the experiments

4.1 Cluster recovery on synthetic data sets

Figure 4.1 shows the average cluster recovery, measured using the adjusted Rand index (ARI) for each of the algorithms we experiment with. This particular figure presents results broken down by the number of clusters (i.e. synthetic data sets with 2, 5, 10, and 20 clusters). We can see that on data sets containing two clusters, all algorithms perform similarly on average. This in itself is an interesting result. When we increase the number of clusters some algorithms (eg. hatamlou, khan, and singlepass) begin to perform rather poorly, reaching average values of ARI under 0.5, which is lower than that achieved by the random initialisations (random_c and random_p).

Of course, there are reasons for this. The hatamlou binary search algorithm takes centroids from K different parts of the data, with each part calculated independently for each feature. Hence, K is proportional to the probability of a mismatch between centroids and dense areas, which is probably the reason for the poor result. In the case of khan, centroids are calculated using a single feature. Thus, as K increases the probability of one feature containing information about all K clusters decreases.

The initialisations globalkm, hand, ikmeans_card, ikmeans_first, kkz, kmpp, milligan, and yuan seem to be those that are the most resistant to an increase of K. We should probably mention that ikmeans_card and ikmeans_first were the only initialisations producing a higher ARI for $K = 20$ than for $K = 2$. It seems likely that this is related to the fact that, as discussed in Section 2.2.9 and shown in Table C.1, Intelligent k-means increasingly struggled to find the correct number of clusters as K increased. As we assume K to be known for all initialisation algorithms equally, we were forced to ignore results for those cases, which intuitively seem likely to correspond to the more challenging data sets. Thus, the results for ikm_card and ikm_first may show bias in this specific case.

Figure 4.2 shows the average ARI on the same data sets but now broken down by the number of features in each data set (i.e. 2, 10, 50, 100, and 1000). Generally speaking, the increase in the number of features usually leads to better cluster recovery in our experiments. This is probably because these are synthetic data sets containing Gaussian clusters, and the maximum number of features we experiment with is 1,000. The only exceptions to this trend, with poor results, were hatamlou, khan, and singlepass. The reasons for this are probably similar to those for our experiments broken down by the number of clusters. For instance, an increase in V has a similar effect to that of an increase in K for hatamlou. The initialisations onoda_ica and onoda_pca produced the highest average ARI for $V = 2$, this seem to suggest these are good initialisations for this scenario. However, onoda_ica and onoda_pca are unable to produce a set of initial centroids when $V < K$ (see Section 2.2.15). Hence, the results shown for $V = 2$ only include the scenario where $K = 2$. For the other values of V (i.e. 10, 50, 100, and 1000) globalkm, hand, ikmeans_card, ikmeans_first, kkz, kmpp, and milligan produced the best results, followed closely by yuan.

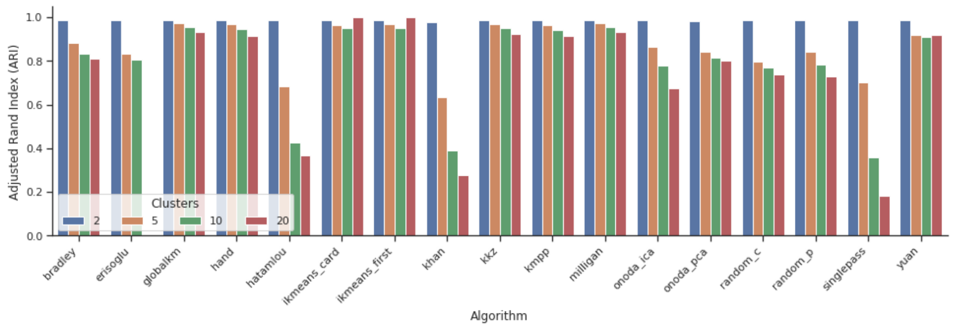

Figure 4.1: Mean ARI over all synthetic data sets broken down by number of clusters.

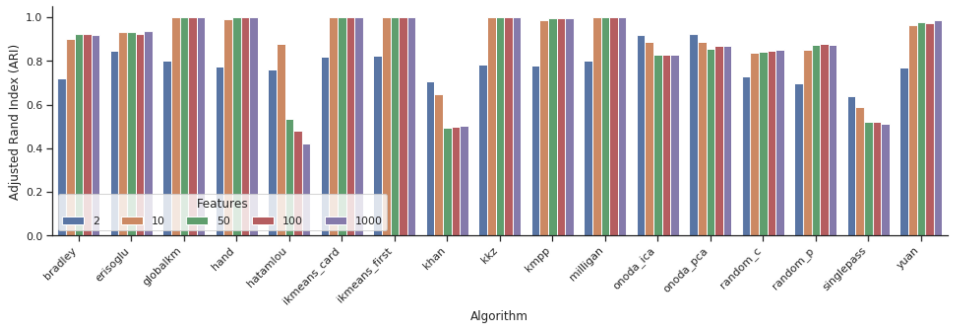

Figure 4.2: Mean ARI over all synthetic data sets broken down by number of features.

Figure 4.3 shows the average ARI on the same data sets but now broken down by the within-cluster standard deviation. A higher within-cluster standard deviation means clusters are more sparse, leading to a higher chance of

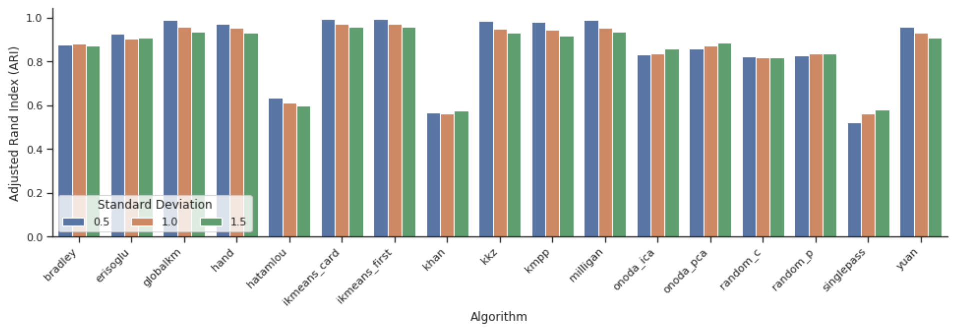

Figure 4.3: Mean ARI over all synthetic data sets broken down by within-cluster standard deviation.

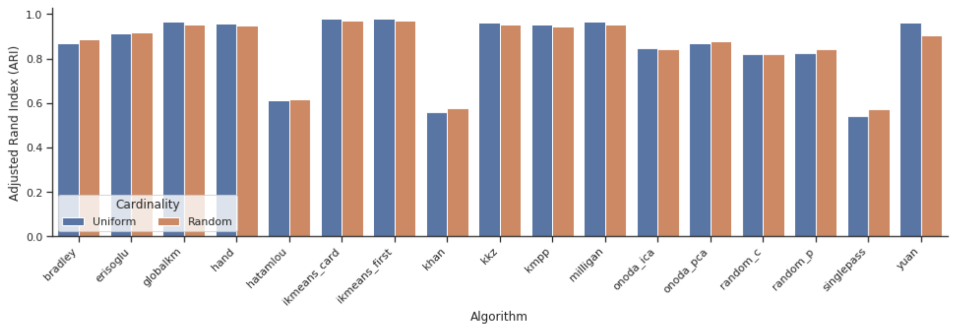

Figure 4.4: Mean ARI over all synthetic data sets broken down by cardinality.

overlap. The highest ARI for a within-cluster standard deviation of 0.5 was given by globalkm, ikmeans_card, ikmeans_first, and milligan—followed very closely by hand, kkz, kmpp, and yuan. With a higher standard deviation we can see a small lead by ikmeans_card and ikmeans_first.

Figure 4.4 shows the average ARI on the same data sets but broken down by cardinality of clusters. A uniform cardinality means that each cluster had the same number of data points. A random cardinality allows clusters to have a uniformly random number of data points, subject to a minimum of 30 data points per cluster. Surprisingly some algorithms had very similar (and good) results for both experiments. Again, we can see a small lead by ikmeans_card and ikmeans_first.

4.2 ANOVA effect sizes on synthetic data sets

This section presents a further statistical analysis of the cluster recovery performance of the initialisation algorithms, as measured by ARI. These are the results of a series of one-way ANOVA tests on the results presented in Section 4.1. These tests are used to measure the sensitivity of each algorithm to each of the data set characteristics shown in Table 3.1.

The results are presented using the $\hat{\eta}^2$ measure of effect size. The values range from between 0 and 1, where larger values of $\hat{\eta}^2$ indicate that an algorithm's cluster recovery performance was more affected by each specific data set characteristic.

Name	Clusters	Features	Std. deviation	Cardinality
bradley	0.1938	0.2803	0.0002	0.0029
erisoglu	0.2798	0.0569	0.0036	0.0002
globalkm	0.0277	0.4256	0.0344	0.0020
hand	0.0440	0.4847	0.0156	0.0015
hatamlou	0.5082	0.2621	0.0015	0.0001
ikm_card	0.0289	0.3965	0.0232	0.0025
ikm_first	0.0284	0.3845	0.0233	0.0025
khan	0.6466	0.0723	0.0002	0.0008
kkz	0.0317	0.4596	0.0293	0.0013
kmpp	0.0402	0.4451	0.0396	0.0014
milligan	0.0281	0.4269	0.0340	0.0019
onoda_ica	0.4507	0.0320	0.0047	0.0005
onoda_pca	0.3021	0.0126	0.0057	0.0010
random_c	0.3492	0.0792	0.0004	0.0000
random_p	0.3521	0.1802	0.0008	0.0018
singlepass	0.7836	0.0197	0.0051	0.0017
yuan	0.0460	0.3608	0.0223	0.0435

Table 4.2: ANOVA $\hat{\eta}^2$ scores for factor levels by initialisation

4.3 *k*-means iterations on synthetic data sets

In this section we discuss the impact of each initialisation algorithm on the number of iterations k-means takes to converge, with a lower number of iterations being desirable.

We note that for globalkm and hand, our experiments show that k-means took a single iteration to converge, which may seem like an incredibly good result. However, these algorithms already incorporate k-means (without changes) within them, and run it multiple times.

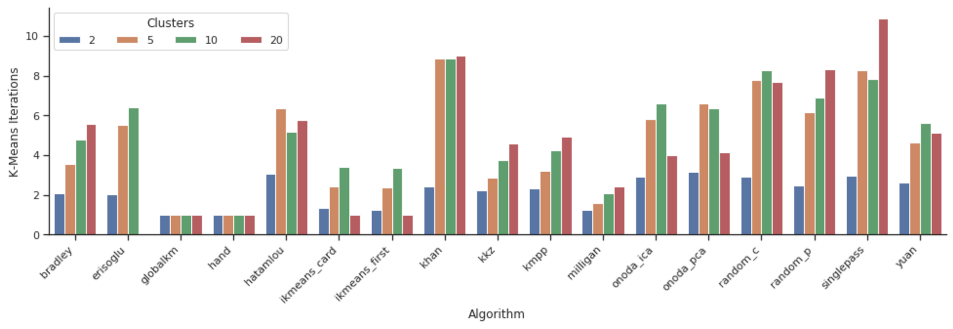

Figure 4.5: Mean k-means iterations taken to converge over all synthetic data sets broken down by number of clusters.

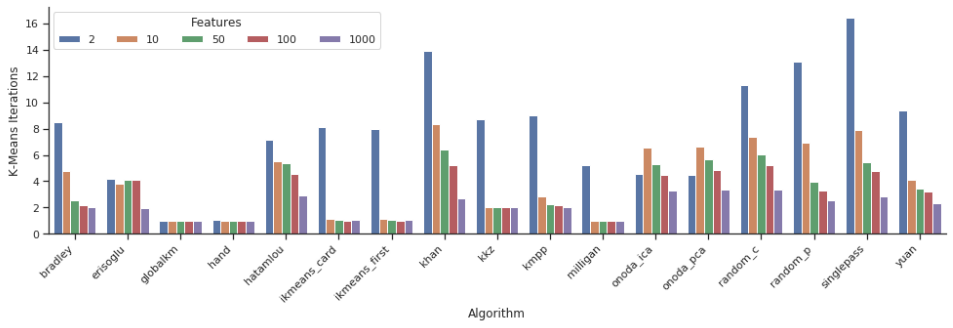

Figure 4.6: Mean k-means iterations taken to converge over all synthetic data sets broken down by number of features.

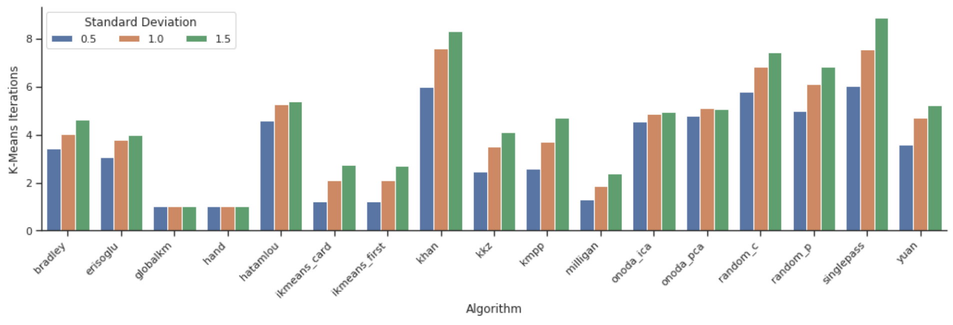

Figure 4.7: Mean k-means iterations taken to converge over all synthetic data sets broken down by within-cluster standard deviation.

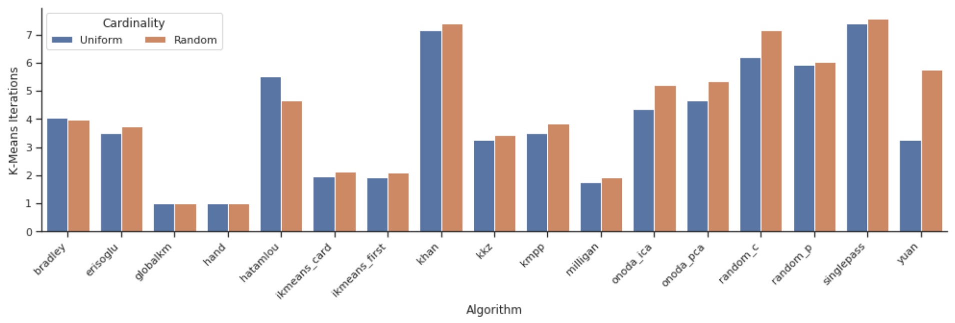

Figure 4.8: Mean k-means iterations taken to converge over all synthetic data sets broken down by cardinality.

Figure 4.5 shows the average number of iterations k-means took to converge when initialised by each algorithm, broken down by the number of clusters in each data set. The best results are given by ikmeans_card, ikmeans_first, and milligan. It is interesting to see these algorithms lead to similar performance when the number of clusters is two. When the number of clusters is five or ten milligan outperforms ikmeans_card and ikmeans_first. However, when the number of clusters is 20 ikmeans_card and ikmeans_first take the lead again. This may be because although very good, milligan is based on hierarchical clustering. This latter, unlike k-means, does not revisit assignments of data points to clusters. The assignment of a data point to a cluster is final (with the obvious exception where clusters are merged), and future clusters are identified based on previous clusters. Figure 4.6 shows that the major difference between these algorithms happens when the number of features is two. Figures 4.7 and 4.8 show somewhat similar performance between these.

4.4 Results on real-world data sets

The results on real-world data sets are certainly more mixed and difficult to read. This is expected as we do not have the ability to control the characteristics of each data set. Having said that, we were still somewhat surprised to see that, measured using ARI, all algorithms performed about the same in terms of cluster recovery on average (see Figure 4.9).

In our experiments, we ran each non-deterministic algorithm 50 times. Hence, we were able to calculate the standard deviation of an algorithm when applied to a particular data set. Figure 4.9 also shows the average of these standard deviations, which were not particularly high. Additionally, we show the proportion of incorrect number of clusters, this being the number of times an initialisation algorithm returned a clustering S such that $|S| \neq K$. In our experiments, we always make the assumption that K is known, so we ignored such cases in our other results.

Figure 4.10 shows the average number of iterations k-means took to converge when initialised by each of the algorithms we experiment with. It was interesting to see hatamlou and erisoglu among the best performers, given that they did not perform well in our other experiments. We should probably mention the good result obtained by onoda_pca. This figure shows a small advantage of ikmeans_card over ikmeans_first, as does Figure 4.9.

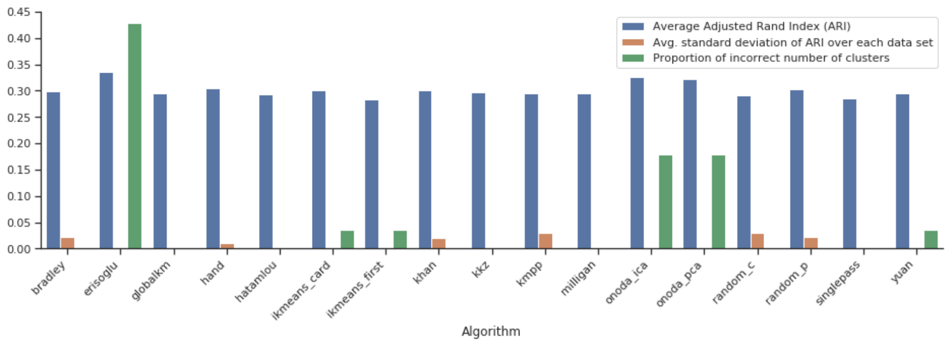

Figure 4.9: Mean ARI over all real-world data sets, standard deviation of ARI over each real-world data set, and the proportion of incorrect number of clusters found by each algorithm over all real-world data sets.

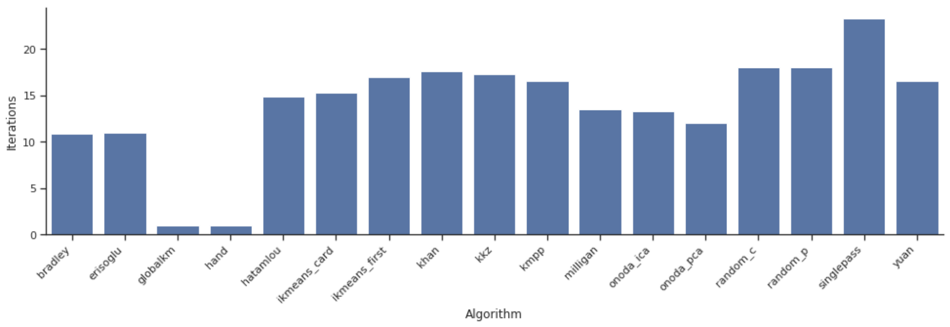

Figure 4.10: Mean k-means iterations taken to converge over all real-world data sets.

4.5 Recommendations for practical applications

In this section we introduce a small number of brief example use cases exploring how the results presented above may potentially be used to inform the selection of an appropriate k-means initialisation algorithm. Each example involves a hypothetical researcher in a different field, in each case making use of k-means clustering in their work.

The use cases presented are not exhaustive, nor are they intended to be. Rather, we hope they provide a little insight into some ways in which the outcomes of the research presented in this work might be of value to the wider research community, and the thought processes that may be involved in making use of our results.

4.5.1 Vector quantisation

Vector quantisation is a data compression technique based on clustering. Essentially, a clustering process is applied to the data set to be compressed, and the resulting, compressed data set consists of two elements: the final cluster centroids and the list of cluster indices which then represent each data point [4]. From these artefacts, a data set can be reconstructed, or in other words decompressed. Clearly, the technique is "lossy", in that information is lost in the compression process and can never be recovered. Thus, vector quantisation is applicable where the importance of reducing the size of the data is considered to outweigh the disadvantages of the information loss.

This technique is particularly suitable, and will tend to be used most often, in situations where the data set consists of many points which are highly similar to each other. In other words, where points within each cluster have a lower distance from their centroid. This is because in such cases the quantity of information lost in compression is reduced. Therefore, a researcher might select a k-means initialisation algorithm providing good cluster recovery performance on data sets with low within-cluster standard deviation. From Figure 4.3 we see that Intelligent k-means (ikm_card and ikm_first) consistently shows excellent performance (in other words, mean ARI scores close to or at 1.0) where within-cluster standard deviation is low, as do Global k-means (gkm), Hand & Krzanowski (hand), KKZ, k-means++ (kmpp), Milligan, and Yuan.

Of course, data compression in general is most likely to be desirable in cases where the data set is very large. A researcher might therefore find it preferable to select an initialisation which results in a low number of k-means iterations being required for convergence, thus reducing the time taken to cluster the data. From Figure 4.7 we see that, of the algorithms identified above, Global k-means and Hand & Krzanowski show excellent performance by this criterion, typically requiring a single k-means iteration for convergence. This is very closely followed by Intelligent k-means and Milligan.

Where data sets are large, and all other factors being equal, it seems reasonable for a researcher to prefer a deterministic k-means initialisation algorithm over a non-deterministic one, as the former requires only a single run, whereas the latter would require numerous restarts, with the researcher having to make the arbitrary decision of how many restarts are sufficient. For this reason the researcher may choose not to select Hand & Krzanowski.

The computational performance of algorithms is generally beyond the scope of this work. However, where data sets are large enough to justify compression in the first place, it is impossible to ignore the fact that the sheer speed of an algorithm is likely to be of great importance to the researcher. Global k-means is known to be slow[1], and in fact computationally prohibitive for large data sets, requiring $N(K-1)$ runs of k-means on the entire data set [39]. Indeed, even its original authors explicitly acknowledge the computational performance of Global k-means to be problematic [70]. For this reason, it is not possible to recommend that the researcher select Global k-means.

From the above analysis we feel it is reasonable to conclude that, for the researcher using k-means as the clustering algorithm for vector quantisation, both Intelligent k-means and Milligan remain viable and in fact very strong recommendations as the choice of k-means initialisation algorithm.

4.5.2 Regression analysis

Regression analysis is a group of statistical methods used to determine the mathematical relationship between a dependent variable and one or more independent variables. As such, a regression model can be used to predict the effects of changes in the independent variables upon the dependent variable.

[1]Our informal observations do appear to confirm this. We direct interested readers to Appendix C.2 ("Computational performance of algorithms"), in particular Figure C.6

Since regression has quadratic time complexity of $O(N^2)$ (where N is the number of data points), it is computationally prohibitive for large data sets. One approach to managing this problem involves data clustering: once the data set has been clustered, the regression analysis may be performed upon the resulting centroids [4].

In general, the accuracy of the regression analysis will be improved by a greater number of centroids and a higher accuracy of clustering [4]. A researcher using k-means as the clustering algorithm for regression analysis will therefore wish to select an initialisation algorithm which displays strong cluster recovery performance where the number of clusters K is high.

For the synthetic data sets used in our experiments, the number of clusters ranges from 2 to 20, and we acknowledge that this maximum may in some cases be significantly less than required in the case of regression analysis. That being said, we hope that our results provide some insight into how well the cluster recovery performance of each algorithm scales as K increases[2].

From the results shown in Figure 4.1, we can immediately disregard certain initialisation algorithms for this use case. As noted in Section 2.2.11, the Erisoglu, Calis & Sakallioglu algorithm (erisoglu) was unable to successfully cluster the data in any experimental run where $K = 20$. Further, the ARI values returned by Hatamlou, Khan and Single Pass Seed Selection (singlepass) degrade markedly as K increases.

By contrast, both versions of Intelligent k-means actually show better cluster recovery performance at $K = 20$ than at lower values, although as discussed in Sections 2.2.9 and 4.1, the results in those cases may not be truly representative. In many experimental runs of Intelligent k-means we were not able to produce results (see Table C.1), and so the performance of this algorithm at $K = 20$ remains unknown.

The remaining algorithms showing relatively strong cluster recovery performance for higher values of K are Global k-means, Hand & Krzanowski, KKZ, k-means++ (kmpp), Milligan and Yuan, so we might consider these to be a shortlist for recommendation.

As established above, we are concerned here only with situations whereby

[2]We note in passing that our hypothetical researcher would no doubt be capable of using linear regression to predict more accurately how performance would scale to larger values of K, the independent variable in this case.

the data sets upon which regression analysis is to be performed are large enough to require the use of clustering in the first place. Thus, we must now consider the implications of the selection of a specific k-means initialisation with regard to the timescales involved.

In the vector quantisation example discussed in Section 4.5.1, it was agreed that Global k-means is computationally prohibitive for large data sets, and so it is impossible to recommend its use here. Similarly, we acknowledged that for performance reasons, assuming all other factors to be equal, a deterministic algorithm is considered to be preferable to a non-deterministic one, and so the researcher may choose not to employ k-means++.

Whilst Hand & Krzanowski is non-deterministic, we must now consider the number of iterations required for k-means itself to converge. From Figure 4.5 we can see that, of the four algorithms remaining under consideration, Hand & Krzanowski (hand) consistently requires a single k-means iteration to converge in all cases and, whilst this is yet to be demonstrated in our experimental results, it does not seem unreasonable to extrapolate this behaviour to higher values of K. Milligan shows the second best performance here, although the number of iterations required can be seen to increase as K increases. This is also the case for KKZ and Yuan, but their performance is visibly worse than for Milligan.

Based on the above analysis, Hand & Krzanowski appears to be the most suitable choice of algorithm for the regression analysis use case. However, our informal timing statistics (see Figure C.7) show that the speed of Hand & Krzanowski does decrease as K increases, whereas Milligan seems to show relatively consistent performance.

If we imagine the timing statistics to be credible, it seems impossible to choose between Hand & Krzanowski and Milligan based on the information available. In such situations we would recommend that the researcher consider both algorithms and perform experimental runs of each against samples of their real data, so that a choice of k-means initialisation algorithm may be made empirically, based on their specialised findings.

4.6 Summary

This chapter presented the results of our experiments in two ways. Firstly, we showed the cluster recovery performance of each algorithm broken down by the various characteristics of the data sets used in the experiments. Secondly, we explored how a hypothetical researcher might make use of the results in order to inform the choice of a k-means initialisation algorithm to be used in their own work. It is hoped that this shows to some extent the potential value of this research.

In the following chapter we turn to the practical details of the software developed in order to perform our experiments, which is a key deliverable of this project, and introduce the three new open source software libraries which resulted from this work.

5 Software Libraries

As previously mentioned, a key deliverable for this project is a suite of software libraries comprising implementations of all the algorithms covered in our experiments, along with a small framework to run these algorithms against the many data sets used, and various utilities for data generation and processing. Our approach ensures that the experiments described in Section 3 are reproducible. We hope that the development work involved in this project may be of value to subsequent researchers and developers, and so make all the software freely available and open source, and welcome contributions from the wider community.

The software in question comprises three libraries, the primary one of these being named pykmeans. This includes the implementations of the k-means initialisation algorithms covered in Section 2.2, along with the supporting framework used to run the experiments and capture the output. The pykmeans library is discussed in Section 5.1.

We make available two further open source libraries which emerged from the pykmeans project. These are named pycleandata (see Section 5.2) and pygendata (Section 5.3), both of which were used extensively in performing our experiments. The former downloads and pre-processes real-world data sets, including those described and discussed in Section 3.1.2, while the latter was used in the generation of our 6,000 synthetic data sets which are described and discussed in Section 3.1.1.

The remainder of this section is intended to provide a description of the three libraries, including a brief overview of how they may be used and the structure of the output generated by each.

5.1 pykmeans

This is the primary software library used to perform the experiments described in this work. It is implemented in Python 3, and comprises: an example implementation of the k-means algorithm itself[1]; implementations in Python 3 of the k-means initialisation algorithms covered in Section 2.2; and the supporting framework code used to run the experiments and capture the output.

The framework makes use of parallel processing where this is available, using Python's `concurrent.futures.ProcessPoolExecutor` class, and captures and saves output from each experimental run, handling errors gracefully where possible.

While many of the initialisations are full Python implementations based on the algorithms published in the original literature, several make use of built-in scikit-learn functions where these are available, for example the Ward hierarchical algorithm used in Milligan (Section 2.2.3). In such cases we implement a lightweight wrapper around the existing functionality, thereby providing a consistent interface across all algorithms.

5.1.1 File overview

In this section we present a brief overview of the file structure of the pykmeans library. This is not intended to be comprehensive, rather to provide a brief introduction to what is included in the library.

- `runner.py`: loads data sets from disk, executes the experimental runs in parallel, and saves the resulting output to disk
- `kmeans.py`: the example implementation of k-means itself
- `dataset.py` and `cluster.py`: Python classes used throughout pykmeans
- `initialisations/`: the implementations of the k-means initialisation algorithms
- `datasets/`: storage for data sets, including several example data sets for testing purposes, both real-world and synthetic
- `metrics/`: implementations of and wrappers for cluster validity indices (CVIs)

[1]We must clarify that our implementation of k-means is included only for reference purposes and is not used in the experiments, since scikit-learn provides a well-tested and optimised implementation. However, certain of its functions—in particular `distance_table()`—are used by some of the initialisation algorithms.

- `preprocessors/`: implementations of and wrappers for data preprocessing methods, such as normalisation
- `tests/`: the unit tests for the entire library
- `_output/`: an empty directory for storing the output from experimental runs, including recovered cluster labels
- `verify.sh` a simple shell utility to monitor progress of experimental runs

5.1.2 Usage

All operations for pykmeans—and indeed pygendata and pycleandata—are triggered at the command line. To invoke an experimental run, the command is as follows:

```
$ python3 runner.py <algorithm> <data_dir> <restarts>
```

The parameters which must be supplied to `runner.py` as above are:

1. `algorithm`: the identifier for the initialisation algorithm to be run, each of which can be found in Table 4.2
2. `data_dir`: the relative path to the directory containing the data sets for the experimental run
3. `restarts`: the number of restarts to be performed per data set, which will typically be 1 for deterministic initialisation algorithms and more for non-deterministic algorithms

Within the specified `<data_dir>` directory, each individual data set must be located with its own subdirectory, which in turn must contain the files `data.csv` and `labels.csv`. The former of these must contain the data set itself, and the latter the target class labels (which are used to calculate adjusted Rand index (ARI) as per Section 3.2). This layout is consistent with the data sets output by both pygendata and pycleandata.

Where available, it is recommended to run `runner.py` within the `nohup` wrapper, since execution time for the experimental runs of some algorithms can extend to a matter of hours when applied to thousands of data sets.

The pykmeans library includes a reasonably thorough suite of unit tests, which can be invoked using the command:

```
$ python3 -m unittest discover
```

Of course, unit test coverage can always be extended, and we acknowledge this as one of our items for future work under Section 5.1.5.

5.1.3 Output

All output from the experimental run is placed under `_output/`. Within that, a subdirectory is created, with its name identifying the algorithm, specifically using the string provided as the `<identifier>` parameter above. This in turn contains a subdirectory named exactly as the `<data_dir>` parameter. Within the latter subdirectory a further directory is created for each data set processed. This will contain files of two types, following the naming conventions `labels-*.csv` and `output-*.csv`. One of each of these files is created for each individual restart of the algorithm, and the wildcard is replaced by a counter identifying each restart.

The "labels" files contain a comma-separated list of the class labels retrieved for the data set, while the "output" files contain all the information needed to identify the specific restart, the algorithm and the data set, along with the calculated ARI and number of k-means iterations required to converge. It is these exact files which are concatenated and used to generate the charts shown in Section 4.

5.1.4 Requirements

The pykmeans library has a small number of dependencies on widely-available Python-based libraries:

- Python 3
- scikit-learn >= 0.20
- SciPy >= 1.3.1
- NumPy >= 1.15.4 [87]

5.1.5 Future work

It is anticipated that the pykmeans library should be an ongoing project, and as such there is plentiful scope for further development. We note in particular:

- There are numerous proposed k-means initialisations in the literature which are not yet included in the library due to practical limitations. We anticipate further algorithms to be added in time, and welcome contributions from the wider community.
- Whilst the requirement of additional parameters for k-means initialisation algorithms may be considered undesirable, we acknowledge that they are a fact of life. In our experiments we consistently use fixed values as recommended by original authors, however it may be helpful to subsequent researchers to provide a more flexible mechanism by which to vary these.
- Whilst a reasonably complete set of unit tests is included, we acknowledge that test coverage for any project can always be improved.

The pykmeans library is open source under the terms of the MIT License, and is available via GitHub at:
`https://github.com/simonharris/pykmeans`.

5.2 pycleandata

This is the tool used to automate the download and cleaning of the real-world data sets used in these experiments, as detailed in Section 3.1.2. Whilst pycleandata was not strictly one of the original deliverables specified for this work, we make it available as an open source project in the hope that it may be of use to further researchers or developers.

The tool downloads data sets—typically from the UCI Machine Learning Repository, though this is not mandated—and automates several common data cleaning tasks. Such tasks are configured by the user and examples include: the dropping of any data columns which are not required, such as an index column; removal of data points with missing data; and the isolation of class labels.

We acknowledge that certain data sets require further ad hoc pre-processing steps such as decompression, concatenation of files, or file format conversions which are—at least currently—beyond the scope of pycleandata. In such cases we performed the work manually and make available the resulting pycleandata-compatible data files by hosting them in the project's GitHub account.

The library is written in Python and makes use of the pandas library [88] for data manipulation.

5.2.1 File overview

This section is not an exhaustive listing of every file comprising the pycleandata library, rather it is intended to provide a brief overview of the codebase. Some key files include:

- `cleandata.py`: the main Python file which coordinates the data cleaning process
- `dataset.py`: a Python class encapsulating the properties and behaviour of an abstract data set, such as the data itself, the class labels, and the functionality for downloading and saving to disk
- `data.yml`: the main YAML configuration file
- `_cache/`: local cache for downloaded data files
- `offline_data/`: the directory in which any manually pre-processed data files are hosted

5.2.2 Usage

The details of the data sets to be processed by pycleandata must be specified in the `data.yml` file. The library comes pre-configured for the 28 data sets used in these experiments. This section will not detail each configuration directive in detail as this is the responsibility of the library's own documentation. However, by way of examples, such directives include: the ability to specify which column separator is used in the data; which, if any columns are to be dropped; whether the data set contains an index column or header row; and the characters or strings which signify missing data.

Each data set must be assigned a unique key as an identifier, as this is used throughout the process, and identifies the resulting processed data set which is stored locally.

The command to execute the downloading and processing of all configured data sets is rather simple:

```
$ python3 cleandata.py
```

To download and process a single data set, a further optional argument exists. This must be the data set key as it appears in `data.yml`, as discussed above. For example, to retrieve and process the well-known Iris data set:

```
$ python3 cleandata.py iris
```

For users with access to GNU `make`, a `Makefile` is also provided, containing targets i) the default target `go`, to run `workhorse.py` for all data sets as above; ii) `cacheclean`; to remove locally-cached data sets from `_cache/`; and iii) `dataclean` to remove all locally-stored processed data sets. These last targets instinctively feel dangerous, however a fundamental design decision behind pycleandata is that its operation should be both repeatable and reasonably fast. In this way, restarting the entire process is not typically an onerous matter.

5.2.3 Output

All cleaned data is saved into a directory named `cd_data/`, which will be created if it is not present. Within `cd_data/` a subdirectory is created for each data set processed, each of which in turn contains two files:

- `data.csv`: the data set itself
- `labels.csv`: the class labels of the data points

The tool also creates a report file name `cd_report.csv`, with details of the data sets processed in the latest run. This is, incidentally, the file used to populate the fields of Table 3.2.

5.2.4 Requirements

The pycleandata library has dependencies on a small number of widely-available Python libraries:

- Python 3
- NumPy >= 1.15.4
- pandas >= 0.23
- PyYAML >= 3.13

5.2.5 Future work

As with the pykmeans library, pycleandata is deliberately an ongoing project, and there is scope for further development. Features and improvements which may be considered include:

- the facility to maintain more than one configuration file, and perhaps specify this at the command line
- greater flexibility in terms of the normalisation of data, as this is currently either "on" or "off", and at the time of writing only the approach specified in (B.1) is supported

The pycleandata library is open source under the terms of the MIT License, and is available via GitHub at:
`https://github.com/simonharris/pycleandata`.

5.3 pygendata

This is the tool used to generate the synthetic data sets, as described in Section 3.1.1. It therefore has configuration options covering each of the parameters detailed in Table 3.1. The library is written in Python and makes use of scikit-learn's `make_blobs()` function to generate data sets containing Gaussian clusters. Where parallel processing is available, this is implemented using Python's `concurrent.futures.ProcessPoolExecutor` class.

5.3.1 File overview

The structure of the codebase for pygendata is minimal, and consists primarily of:

- `generate.py`: the main data generation script
- `output/`: empty directory for generated data sets

5.3.2 Usage

All configuration options are currently defined and documented within `generate.py`, although it is intended that the configuration directives will be moved to external configuration files in future versions. Once the configuration options are set, data can be generated using the following command:

```
$ python generate.py
```

For users with access to GNU `make`, a `Makefile` is also provided, containing targets i) `data`, to run `generate.py` as above; and ii) `clean`; to remove generated data sets from `output/`. The latter should probably be used with care.

5.3.3 Output

Under `output/`, each generated data set is placed inside its own directory, with a naming convention based on its configuration. So for a data set named `2_10_1000_r_0.5_004`, in order:

- number of clusters = 2
- number of features = 10
- number of samples = 1000
- cardinality (**u**niform or **r**andom) = random
- within-cluster standard deviation = 0.5
- index, in other words a counter, as we can generate multiple data sets for each configuration = 4

For manageability, generated data sets are grouped into subdirectories based on the number of clusters, that is to say the current value from iterating `OPTS_K`. Each dataset subdirectory then contains:

- `data.csv`: the data set itself
- `labels.csv`: the class labels of the data points

We mention in passing that the structure of data sets generated by pygendata is entirely consistent with the structure of data sets output by pycleandata (Section 5.2). Contents of `output/` are protected by a `.gitignore` file as it is not anticipated that users will commit them to this project on purpose.

5.3.4 Requirements

- Python 3
- scikit-learn >= 0.20
- NumPy >= 1.15.4

5.3.5 Future work

As with all other libraries presented here, pygendata is an ongoing project, and there is certainly scope for further development work. Features that may be considered desirable include:

- the ability to run from separate configuration files, for example YAML
- support for more flexible normalisation, for example pluggable normalisation strategies which can be specified during the configuration process

The pygendata library is open source under the terms of the MIT License, and is available via GitHub at:
`https://github.com/simonharris/pygendata`.

6 Conclusion and Future Work

In this work we considered the highly popular k-means data clustering algorithm and acknowledged that it is not without its shortcomings. We primarily concerned ourselves with the sensitivity of k-means to its initialisation in the form of a set of initial centroids, and its resulting tendency to find local optima rather than global. Many algorithms have been proposed with the intention of achieving better cluster recovery by providing k-means with an improved initialisation.

We performed an extensive side-by-side study comparing the performance of 17 such algorithms by using them to initialise the k-means clustering of 6,000 synthetic data sets with varying characteristics, and 28 real-world data sets.

We measured the cluster recovery of each algorithm by calculating the widely-used adjusted Rand index (ARI) between each clustering produced by k-means on each data set, and the respective set of correct labels. The use of this particular measure means that the results we present are corrected-for-chance. We also investigated the number of iterations k-means itself required to converge when intialised with the centroids generated by each of the algorithms we experimented with.

Our experiments showed that it is not possible to declare one single algorithm to outperform all others in all cases, as algorithms often fared better or worse when compared using datasets displaying different characteristics. For example, whilst in general algorithms showed a decline in ARI when recovering higher a number of clusters, some showed markedly more consistent performance than others. We also found that certain algorithms simply were not suitable for certain data set configurations, for example sometimes finding the incorrect number of clusters, or being unable to even run in cases where the number of features was smaller than the number of clusters.

We also introduced a suite of novel software libraries which we hope may be of interest and use to subsequent researchers and developers in fields related to this work.

With regard to possible future work, it was not practicable to implement and analyse every published k-means algorithm, and so future studies might choose to implement and incorporate further algorithms.

Appendices

A Conventions

Throughout this work we attempt to adhere to consistent naming and notational conventions for key concepts related to k-means, clustering, and data in general. These are listed here for reference.

- The number of clusters is denoted by K. It is upper case because it is constant.
- "Row" indices are denoted by i, while column indices are v.
- Regarding data, X denotes the whole data set. Thus x_i denotes an individual data point while x_{iv} denotes an individual "cell".
- Subscript t is used to denote "another one", for example. x_t for "some other data point".
- "Rows" are termed *data points*, while columns are termed *features*.
- The number of features in the data set supplied to an initialisation algorithm is V, while the number of data points is N. These are again upper case since they are constant.
- A set of clusters is $S = \{S_1, S_2, ..., S_K\}$, and the individual clusters are upper case because clusters themselves are sets.
- We use the term "centroids" for cluster centroids/prototypes, and "initial centroids" for the output of initialisation strategies, prior to k-means itself being run.
- A set of centroids is $C = \{c_1, c_2, ..., c_K\}$.
- The Euclidean (or other) distance measure is $d(...)$ as per [79] and (1.1).

B Data Normalisation

All of our data sets, both synthetic and real, were range normalised using

$$x_{iv} = \frac{x_{iv} - \bar{x}_v}{max(x_v) - min(x_v)}, \tag{B.1}$$

where $\bar{x}_v$ is the mean of feature v, given by $N^{-1} \sum_{x_i \in X} x_{iv}$. This approach provides us with data sets whereby all features have a mean of 0 and a range of 1. To help visualise this, Table B.1 shows a small food nutrient data set taken from Hartigan's 1975 presentation of k-means [89, Chapter 4], while Table B.2 shows the same data once normalised as per (B.1), rounded to six decimal places.

Food	Energy (cal)	Protein (g)	Calcium (mg)
Beef, braised	11	29	1
Hamburger	8	30	1
Beef, roast	13	21	1
Beef, steak	12	27	1
Beef, canned	6	31	2
Chicken, broiled	4	29	1
Chicken, canned	5	36	1
Beef, heart	5	37	2

Table B.1: The Hartigan food nutrient data set prior to normalisation

Food	Energy (cal)	Protein (g)	Calcium (mg)
Beef, braised	0.333333	-0.0625	-0.25
Hamburger	0.0	0.0	-0.25
Beef, roast	0.555556	-0.5625	-0.25
Beef, steak	0.444444	-0.1875	-0.25
Beef, canned	-0.222222	0.0625	0.75
Chicken, broiled	-0.444444	-0.0625	-0.25
Chicken, canned	-0.333333	0.375	-0.25
Beef, heart	-0.333333	0.4375	0.75

Table B.2: The Hartigan food nutrient data set after normalisation

We have opted for (B.1) rather than the popular z-score [90] normalisation procedure because the latter is biased towards unimodal distributions. To explain this, if the hypothetical features v_1 and v_2 are unimodal and multimodal respectively, the standard deviation of v_2 will be higher than that of v_1. Hence, the z-scores for v_2 will be lower than those of v_1. However, in clustering we are more interested in the information most likely present in the multiple modes of v_2 rather than that contained in the single mode of v_1.

C Supplementary Results Data

C.1 Results by inertia

In this section we present the results of the experiments using an additional cluster validity index (CVI), named inertia or the within-cluster sum-of-squares criterion.

By contrast with the adjusted Rand index (ARI) described in Section 3.2, inertia is an internal CVI, in that it requires no a priori information—specifically cluster labels in this case—to determine cluster validity.

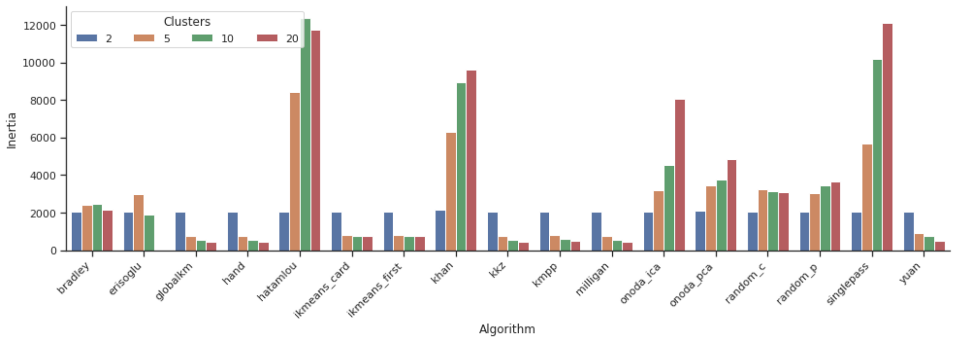

Figure C.1: Mean inertia over all synthetic data sets broken down by number of clusters.

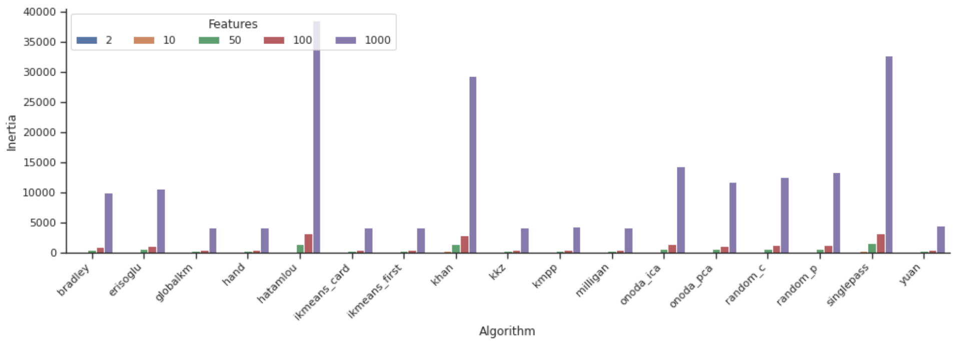

Figure C.2: Mean inertia over all synthetic data sets broken down by number of features.

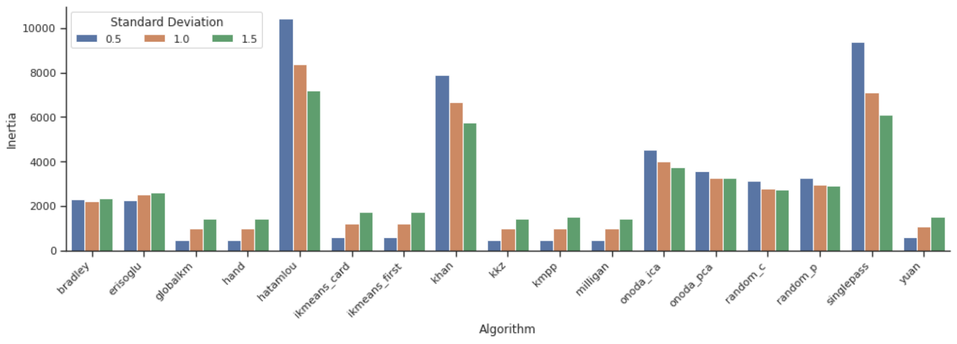

Figure C.3: Mean inertia over all synthetic data sets broken down by within-cluster standard deviation.

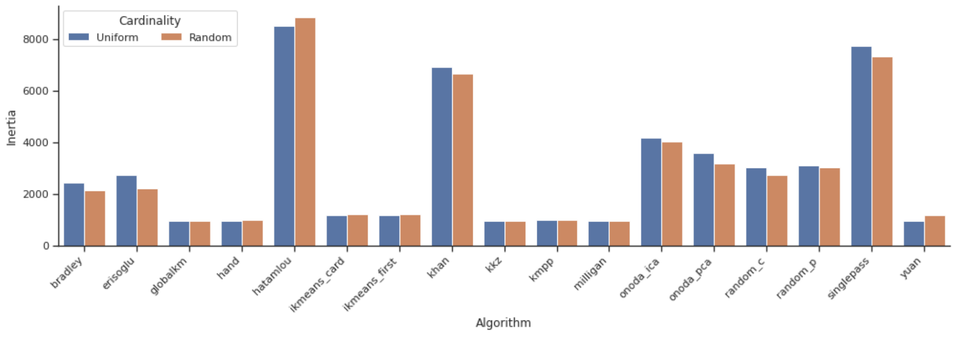

Figure C.4: Mean inertia over all synthetic data sets broken down by cardinality.

This is a non-normalised index. The minimum inertia is 0 and there is no maximum. Since Euclidean distance becomes inflated the greater the number of features, inertia can increase vastly the higher the dimensionality of the data. This can clearly be seen from Figure C.2, where the subtleties at lower dimensionality are almost completely lost. In comparing algorithms by other data set characteristics, we see a general correlation with the ARI-based results presented in Section 4.1. Thus, we reproduce these results here purely for interest and completeness, not least because k-means itself seeks to minimise this criterion.

C.2 Computational performance of algorithms

Whilst this topic is stated to be beyond the scope of this work, it seems inevitable that questions will be asked about the computational performance, or more specifically speed, of the k-means initialisation algorithms included in the experiments.

Since we did record timing data for each experimental run, we include it here for the interested reader, although we do so with certain caveats. Firstly, the numbers presented are entirely unscientific, since measurements were not taken under what would be considered controlled laboratory conditions: the experiments were run on a server farm on which many other processes may be running at any one time. Inevitably, the timings captured will be affected by the relative load on the servers at the time of running.

Secondly, this is not a like-for-like comparison between the speed of each algorithm: some implementations are very thin wrappers over library functions supplied by scikit-learn, which can be assumed to be highly optimised and thoroughly tested. Other implementations are hand-coded, interpreted Python 3 programs written by a single developer to project deadlines, with computational performance not being a high priority.

All timings are the output of Python 3's `time.perf_counter()` function[1], and are measured in fractional seconds.

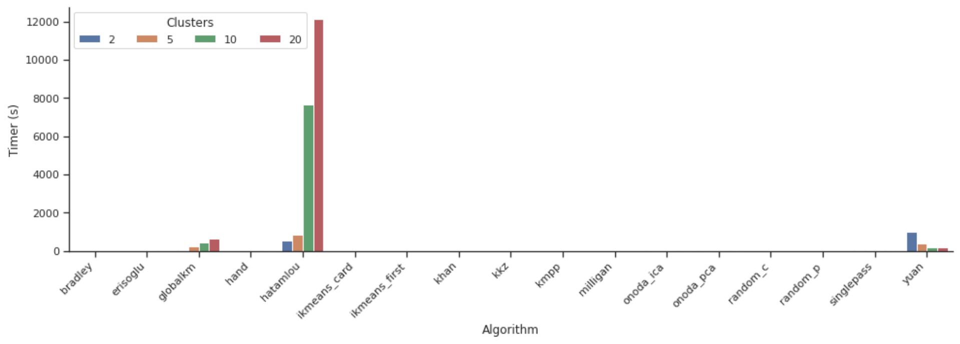

Figure C.5: Mean execution time per algorithm.

It is rather clear from Figure C.5 that the implementation of Hatamlou suffers from significant performance issues. This is not unexpected, partly because it is indeed a relatively involved algorithm, but moreover because we acknowledge it to be a naive implementation, involving many nested programmatic loops. Improvement of the implementation's speed is certainly to be considered future work, perhaps involving vectorisation of as many operations as possible.

[1]https://docs.python.org/3/library/time.html#time.perf_counter

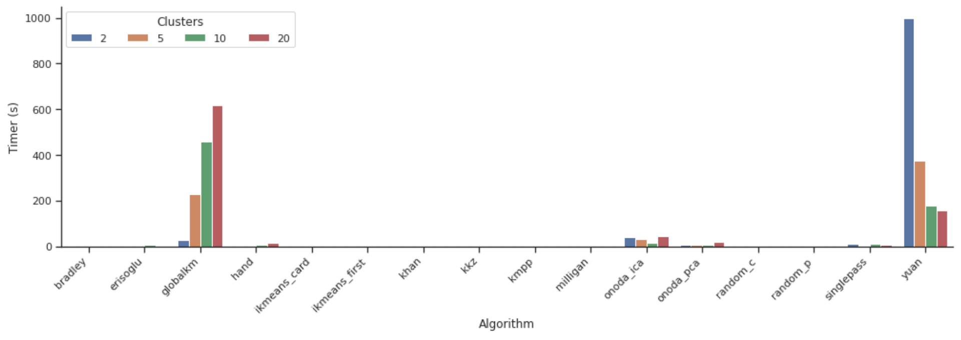

Figure C.6: Mean execution time per algorithm excluding Hatamlou.

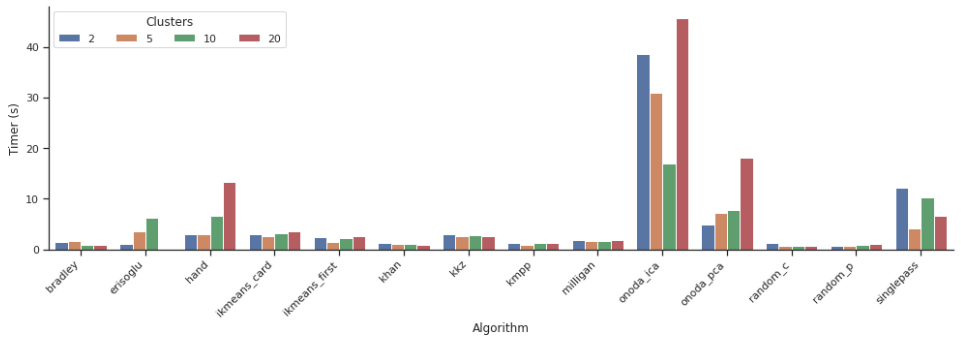

Figure C.7: Mean execution time per algorithm excluding Hatamlou, Global k-means and Yuan.

With Hatamlou removed (Figure C.6) a little more detail becomes visible. Here we clearly see the performance issues which are known to be characteristic of Global k-means [39, 70].

Removing Global k-means and Yuan allows a great deal more detail to be seen regarding the faster algorithms. This is shown in Figure C.7.

C.3 Unsuccessful experimental runs

We were not able successfully to find results for every experimental run: for a variety of reasons, not all algorithms were able to converge for every data set. The explanations for this are given in the algorithms' respective descriptions in Section 2.2. Table C.1 shows the number of experimental runs which were successfully completed for all algorithms. Where the number of clusters $K = 2$, all experiments successfully completed, so these cases are omitted from the table.

Algorithm	$K = 5$	$K = 10$	$K = 20$	Real-world
bradley	1500/1500	1500/1500	1500/1500	28/28
erisoglu	**1220/1500**	**26/1500**	**0/1500**	**16/28**
globalkm	1500/1500	1500/1500	1500/1500	28/28
hand	1500/1500	1500/1500	1500/1500	28/28
hatamlou	1500/1500	1500/1500	1500/1500	28/28
ikmeans_card	**1377/1500**	**1092/1500**	**895/1500**	**27/28**
ikmeans_first	**1377/1500**	**1092/1500**	**895/1500**	**27/28**
khan	1500/1500	1500/1500	1500/1500	28/28
kkz	1500/1500	1500/1500	1500/1500	28/28
kmpp	1500/1500	1500/1500	1500/1500	28/28
milligan	1500/1500	1500/1500	1500/1500	28/28
random_c	1500/1500	1500/1500	1500/1500	28/28
random_p	1500/1500	1500/1500	1500/1500	28/28
singlepass	**1499/1500**	**1499/1500**	1500/1500	28/28
onoda-ica	**1200/1500**	**1200/1500**	**900/1500**	**23/28**
onoda-pca	**1200/1500**	**1200/1500**	**900/1500**	**23/28**
yuan	1500/1500	1500/1500	1500/1500	28/28

Table C.1: Number of successful experimental runs per algorithm

We note that generally speaking, the number of unsuccessful experimental runs tends to increase as the number of clusters in the data set increases.

Bibliography

[1] B. Mirkin, *Clustering for data mining: a data recovery approach.* Chapman and Hall/CRC, 2005.

[2] D. Steinley, “K-means clustering: a half-century synthesis,” *British Journal of Mathematical and Statistical Psychology*, vol. 59, no. 1, pp. 1–34, 2006.

[3] F. Khan, “An initial seed selection algorithm for k-means clustering of georeferenced data to improve replicability of cluster assignments for mapping application,” *Applied Soft Computing*, vol. 12, no. 11, pp. 3698–3700, 2012.

[4] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining.* Pearson, 2018.

[5] C. Murthy and N. Chowdhury, “In search of optimal clusters using genetic algorithms,” *Pattern Recognition Letters*, vol. 17, no. 8, pp. 825–832, 1996.

[6] R. Xu and D. C. Wunsch, “Survey of clustering algorithms,” 2005.

[7] A. K. Jain, “Data clustering: 50 years beyond k-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[8] R. C. de Amorim, “A survey on feature weighting based k-means algorithms,” *Journal of Classification*, vol. 33, no. 2, pp. 210–242, 2016.

[9] J. C. Bezdek, R. Ehrlich, and W. Full, “Fcm: The fuzzy c-means clustering algorithm,” *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[10] F. Murtagh and P. Contreras, “Algorithms for hierarchical clustering: an overview,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[11] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, “Density-based clustering,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[12] C. Hennig, “What are the true clusters?,” *Pattern Recognition Letters*, vol. 64, pp. 53–62, 2015.

[13] U. Von Luxburg, R. C. Williamson, and I. Guyon, “Clustering: Science or art?,” in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 65–79, 2012.

[14] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[15] E. W. Forgy, “Cluster analysis of multivariate data: efficiency versus interpretability of classifications,” *Biometrics*, vol. 21, pp. 768–769, 1965.

[16] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[17] R Core Team, *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2020.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] A. Hatamlou, “In search of optimal centroids on data clustering using a binary search algorithm,” *Pattern Recognition Letters*, vol. 33, no. 13, pp. 1756–1760, 2012.

[20] D. Arthur and S. Vassilvitskii, “How slow is the k-means method?,” in *Symposium on Computational Geometry*, vol. 6, pp. 1–10, 2006.

[21] P. Fränti and S. Sieranoja, “K-means properties on six clustering benchmark datasets,” *Applied Intelligence*, vol. 48, no. 12, pp. 4743–4759, 2018.

[22] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Transactions on pattern analysis and machine intelligence*, no. 1, pp. 81–87, 1984.

[23] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[24] D. Steinley, "Local optima in k-means clustering: what you don't know may hurt you.," *Psychological Methods*, vol. 8, no. 3, p. 294, 2003.

[25] D. Steinley and M. J. Brusco, "Initializing k-means batch clustering: A critical evaluation of several techniques," *Journal of Classification*, vol. 24, no. 1, pp. 99–121, 2007.

[26] D. J. Hand and W. J. Krzanowski, "Optimising k-means clustering results with standard software packages," *Computational Statistics & Data Analysis*, vol. 49, no. 4, pp. 969–973, 2005.

[27] G. P. Babu and M. N. Murty, "A near-optimal initial seed value selection in k-means means algorithm using a genetic algorithm," *Pattern Recognition Letters*, vol. 14, no. 10, pp. 763–769, 1993.

[28] S. J. Redmond and C. Heneghan, "A method for initialising the k-means clustering algorithm using kd-trees," *Pattern Recognition Letters*, vol. 28, no. 8, pp. 965–973, 2007.

[29] J. M. Peña, J. A. Lozano, and P. Larranaga, "An empirical comparison of four initialization methods for the k-means algorithm," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027–1040, 1999.

[30] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[31] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[32] J. He, M. Lan, C.-L. Tan, S.-Y. Sung, and H.-B. Low, "Initialization of cluster refinement algorithms: A review and comparative study," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 1, pp. 297–302, IEEE, 2004.

[33] J. He, A.-H. Tan, and C.-L. Tan, "Art-c: A neural architecture for self-organization under constraints," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, vol. 3, pp. 2550–2555, IEEE, 2002.

[34] J. T. Tou and R. C. Gonzalez, "Pattern recognition principles," 1974.

[35] J. He, A.-H. Tan, C.-L. Tan, and S.-Y. Sung, "On quantitative evaluation of clustering systems," in *Clustering and information retrieval*, pp. 105–133, Springer, 2004.

[36] D. Steinley and R. Henson, "Oclus: an analytic method for generating clusters with known overlap," *Journal of Classification*, vol. 22, no. 2, pp. 221–250, 2005.

[37] M. J. Brusco, "Clustering binary data in the presence of masking variables," *Psychological Methods*, vol. 9, no. 4, p. 510, 2004.

[38] D. Steinley, "Profiling local optima in k-means clustering: developing a diagnostic technique," *Psychological Methods*, vol. 11, no. 2, p. 178, 2006.

[39] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the k-means clustering algorithm," *Expert Systems with Applications*, vol. 40, no. 1, pp. 200–210, 2013.

[40] T. Su and J. G. Dy, "In search of deterministic methods for initializing k-means and Gaussian mixture clustering," *Intelligent Data Analysis*, vol. 11, no. 4, pp. 319–338, 2007.

[41] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[42] S. van Dongen, "Performance criteria for graph clustering and Markov cluster experiments," 2000.

[43] M. Meilă, "Comparing clusterings—an information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873–895, 2007.

[44] S. Zahra, M. A. Ghazanfar, A. Khalid, M. A. Azam, U. Naeem, and A. Prugel-Bennett, "Novel centroid selection approaches for kmeans-

clustering based recommender systems," *Information Sciences*, vol. 320, pp. 156–189, 2015.

[45] M. do Carmo Nicoletti and A. F. de Oliveira, "Empirical evaluation of five algorithms for the initialization phase of the k-means algorithm," *International Journal of Hybrid Intelligent Systems*, no. Preprint, pp. 1–19, 2019.

[46] A. F. de Oliveira and M. do Carmo Nicoletti, "Favoring the k-means algorithm with initialization methods," in *International Conference on Intelligent Systems Design and Applications*, pp. 21–31, Springer, 2018.

[47] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for k-means clustering," *Pattern Recognition Letters*, vol. 25, no. 11, pp. 1293–1302, 2004.

[48] H. Chernoff, "The use of faces to represent points in k-dimensional space graphically," *Journal of the American Statistical Association*, vol. 68, no. 342, pp. 361–368, 1973.

[49] A. Maedeh and K. Suresh, "Design of efficient k-means clustering algorithm with improved initial centroids," *MR International Journal of Engineering and Technology*, vol. 5, no. 1, pp. 33–38, 2013.

[50] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 4–es, 2007.

[51] P. Fränti and S. Sieranoja, "How much can k-means be improved by using better initialization and repeats?," *Pattern Recognition*, vol. 93, pp. 95–112, 2019.

[52] M. M.-T. Chiang and B. Mirkin, "Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads," *Journal of Classification*, vol. 27, no. 1, pp. 3–40, 2010.

[53] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series c (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[54] M. Astrahan, “Speech analysis by clustering, or the hyperphome method,” *Stanford Artificial Intelligence Project Memorandum AIM-124*, 1970.

[55] M. B. Al-Daoud, “A new algorithm for cluster initialization,” in *WEC’05: The Second World Enformatika Conference*, 2005.

[56] S. Sieranoja and P. Fränti, “Random projection for k-means clustering,” in *International Conference on Artificial Intelligence and Soft Computing*, pp. 680–689, Springer, 2018.

[57] P. Fränti, T. Kaukoranta, and O. Nevalainen, “On the splitting method for vq codebook generation,” *Optical Engineering*, vol. 36, no. 11, pp. 3043–3051, 1997.

[58] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: A new data clustering algorithm and its applications,” *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

[59] V. Faber, “Clustering and the continuous k-means algorithm,” *Los Alamos Science*, vol. 22, no. 138144.21, p. 67, 1994.

[60] G. W. Milligan, “An examination of the effect of six types of error perturbation on fifteen clustering algorithms,” *Psychometrika*, vol. 45, no. 3, pp. 325–342, 1980.

[61] J. H. Ward Jr, “Hierarchical grouping to optimize an objective function,” *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[62] G. W. Milligan and P. D. Isaac, “The validation of four ultrametric clustering algorithms,” *Pattern Recognition*, vol. 12, no. 2, pp. 41–50, 1980.

[63] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, “A new initialization technique for generalized Lloyd iteration,” *IEEE Signal processing letters*, vol. 1, no. 10, pp. 144–146, 1994.

[64] R Core Team, *Initialization of cluster prototypes using KKZ algorithm*, 2020 (accessed July 16, 2020). `https://www.rdocumentation.org/packages/inaparc/versions/1.1.0/topics/kkz`.

[65] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[66] K. K. Pavan, A. A. Rao, A. D. Rao, and G. Sridhar, "Robust seed selection algorithm for k-means type algorithms," *International Journal of Computer Science & Information Technology*, vol. 3, no. 5, p. 147, 2011.

[67] "The usc-sipi image database," 1977.

[68] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on communications*, vol. 28, no. 1, pp. 84–95, 1980.

[69] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering," in *ICML*, vol. 98, pp. 91–99, Citeseer, 1998.

[70] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[71] B. D. Ripley, *Pattern recognition and neural networks*. Cambridge University Press, 2007.

[72] P. Hansen, E. Ngai, B. K. Cheung, and N. Mladenovic, "Analysis of global k-means, an incremental heuristic for minimum sum-of-squares clustering," *Journal of Classification*, vol. 22, no. 2, pp. 287–310, 2005.

[73] F. Yuan, Z.-H. Meng, H.-X. Zhang, and C.-R. Dong, "A new algorithm to get the initial centroids," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*, vol. 2, pp. 1191–1193, IEEE, 2004.

[74] R. C. De Amorim and B. Mirkin, "Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering," *Pattern Recognition*, vol. 45, no. 3, pp. 1061–1075, 2012.

[75] M. Erisoglu, N. Calis, and S. Sakallioglu, "A new algorithm for initial cluster centers in k-means algorithm," *Pattern Recognition Letters*, vol. 32, no. 14, pp. 1701–1705, 2011.

[76] S. Deelers and S. Auwatanamongkol, "Enhancing k-means algorithm with initial cluster centers derived from data partitioning along the data axis with the highest variance," *International Journal of Computer Science*, vol. 2, no. 4, pp. 247–252, 2007.

[77] T. Niknam and B. Amiri, "An efficient hybrid approach based on PSO, ACO and k-means for cluster analysis," *Applied Soft Computing*, vol. 10, no. 1, pp. 183–197, 2010.

[78] T. Onoda, M. Sakai, and S. Yamada, "Careful seeding method based on independent components analysis for k-means clustering," *Journal of Emerging Technologies in Web Intelligence*, vol. 4, no. 1, pp. 51–59, 2012.

[79] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243–256, 2013.

[80] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[81] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz, "Internal versus external cluster validation indexes," *International Journal of Computers and Communications*, vol. 5, no. 1, pp. 27–34, 2011.

[82] G. W. Milligan and M. C. Cooper, "A study of the comparability of external criteria for hierarchical cluster analysis," *Multivariate Behavioral Research*, vol. 21, no. 4, pp. 441–458, 1986.

[83] D. Steinley, "Properties of the Hubert-Arabie adjusted Rand index," *Psychological Methods*, vol. 9, no. 3, p. 386, 2004.

[84] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423, 2001.

[85] R. C. De Amorim and C. Hennig, "Recovering the number of clusters in data sets with noise features using feature rescaling factors," *Information Sciences*, vol. 324, pp. 126–145, 2015.

[86] The University of Essex, *High Performance Computing Cluster*, 2020 (accessed October 28, 2020). `https://www.essex.ac.uk/student/it-services/high-performance-computing-(hpc)`.

[87] T. E. Oliphant, *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.

[88] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[89] J. A. Hartigan, *Clustering algorithms.* John Wiley & Sons, Inc., 1975.

[90] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern Recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.