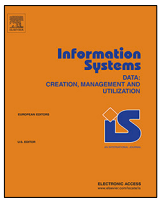
Information Systems 100 (2021) 101787

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Reuse-centric *k*-means configuration

Lijun Zhang [a], Hui Guan [a,*], Yufei Ding [b], Xipeng Shen [c], Hamid Krim [c]

[a] University of Massachusetts Amherst, Amherst, MA, 01002, United States of America
[b] University of California, Santa Barbara, CA, 93106, United States of America
[c] North Carolina State University, Raleigh, NC, 27606, United States of America

ARTICLE INFO

Article history:
Received 17 September 2019
Received in revised form 22 December 2020
Accepted 11 April 2021
Available online 16 April 2021
Recommended by Peixiang Zhao

Keywords:
K-means
Algorithm configuration
Computation reuse

ABSTRACT

K-means configuration is to find a configuration of *k*-means (e.g., the number of clusters, feature sets) that maximize some objectives. It is a time-consuming process due to the iterative nature of *k*-means. This paper proposes *reuse-centric k-means configuration* to accelerate *k*-means configuration. It is based on the observation that the explorations of different configurations share lots of common or similar computations. Effectively reusing the computations from prior trials of different configurations could largely shorten the configuration time. To materialize the idea, the paper presents a set of novel techniques, including *reuse-based filtering*, *center reuse*, and a two-phase design to capitalize on the reuse opportunities on three levels: validation, number of clusters, and feature sets. Experiments on *k*-means–based data classification tasks show that *reuse-centric k-means configuration* can speed up a heuristic search-based configuration process by a factor of 5.8, and a uniform search-based attainment of classification error surfaces by a factor of 9.1. The paper meanwhile provides some important insights on how to effectively apply the acceleration techniques to tap into a full potential.

1. Introduction

As one of the most popular data mining algorithms [1], *k*-means clustering has been used in many applications. Its uses go far beyond simple clustering. An important use of *k*-means, for instance, is for model-free classifier construction. After clustering some training data in the feature space, the method may use the labels of each cluster to classify any new data item falling into that cluster [2]. Another example is to use *k*-means for feature learning by using the centroids of the clusters to produce features [3].

The effectiveness of *k*-means in applications depends on many factors, such as the features used for clustering and the resulting number of clusters. As a result, algorithm configuration is essential for *k*-means–based data mining [4,5]. On the other hand, as an iterative algorithm, *k*-means is very time-consuming to run on large datasets. The configuration of *k*-means for a dataset requires many runs of *k*-means in various settings. The time-consuming nature of *k*-means and the required repeated runs of *k*-means in its configuration make *k*-means–based data mining a time-consuming process, a problem being continuously exacerbated by the rapid growth of data in this era.

There are some general methods proposed for speeding up the configuration process of algorithms [6]. They have mostly focused on how to reduce the number of trial configurations. How to accelerate the examination of the remaining configurations through historical information reuse is a complementary direction that has not received sufficient explorations. And how to effectively accomplish it for *k*-means is yet a largely unexplored problem.

This paper presents a systematic exploration in that direction. It introduces the concept of *reuse-centric k-means configuration*, which promotes information reuse across the explorations of different configurations of *k*-means. The motivating observation is that the explorations of different configurations of *k*-means share lots of common and similar computations. Effectively reusing the computations could largely cut the configuration time with little or no effect on the quality of the final results.

To materialize the idea, this paper strives to answer three main research questions:

- What historical information is essentially useful for *k*-means configuration?
- How to efficiently reuse the information to maximize the reuse benefits?
- Whether and how much the reuse-based optimizations affect the final results?

Specifically, we have designed two techniques, called *reuse-based filtering* and *center reuse*, to promote computation reuse across trials of different configurations.

* Corresponding author.
E-mail addresses: lijunzhang@cs.umass.edu (L. Zhang), huiguan@cs.umass.edu (H. Guan), yufeiding@cs.ucsb.edu (Y. Ding), xshen5@ncsu.edu (X. Shen), ahk@ncsu.edu (H. Krim).

https://doi.org/10.1016/j.is.2021.101787
0306-4379/

Reuse-based filtering takes advantage of the clusters and the distance between a point and its nearest center unveiled in a previous trial of *k*-means. Through the reuse, it is able to leverage triangle inequality to avoid some distance calculations—that is, using computationally efficient lower bounds of the distances between a point and potential centers to filter out some centers that are unlikely to be the nearest to a point, and avoid calculating the distances to those centers. (Section 3.3)

Center reuse is to use the clustering results of some earlier trials to initialize cluster centers for some later trials on different configurations. The reuse helps make the later trials converge faster and hence saves configuration time. (Section 3.4)

For both types of reuse, we have explored the opportunities in multiple levels: across validations, across *k*, and across feature sets. Besides their effectiveness in drastically cutting configuration time, an appealing property of these techniques is their simplicity. They are designed to be simple to implement and deploy to ensure their applicability in general data mining applications.

In addition to the two techniques, we have also explored the use of a *two-phase design* to speed up the configuration process when a full error surface is needed for meeting various desired trade-offs among multiple quality metrics (e.g., different weights of the classification errors over the classification time). (Section 3.5)

We evaluate the efficiency and effectiveness of these techniques by way of both the configuration speed and quality of the final results, in both sequential and parallel settings. Our results show that these techniques can work together in synergy, speeding up a heuristic search-based configuration process by up to 5.8X. When they are used to speed up the attainment of the error surface of *k*-means–based classifiers, they shorten the process by a factor of 9.1. All the optimization techniques we propose cause no change to the final *k*-means results except for the *center reuse* technique. We conduct a focused study on its influence, which concludes that the caused disparity is negligible (less than 3%). We further provide some sensitivity study to reveal how the optimization techniques perform in various settings, and point out some important insights – such as, the directions of configuration explorations – on how to deploy them to tap into a full potential. (Section 4)

Overall, this work makes the following major contributions:

- It provides the first systematic study on how historical information reuse may help speed up the *k*-means configuration process.
- It proposes a set of novel techniques to effectively promote information reuse across explorations of different *k*-means configurations.
- Through sensitivity studies, it reports the performance of the techniques in various settings, and sheds some important insights on the suitable ways to deploy these techniques.
- It demonstrates large (5–9X) speed benefits from these techniques, and confirms only little disparity they may cause to the quality of *k*-means results.

2. Related work

Our work falls in the category of algorithm configuration. Algorithm configuration or tuning is to find the configuration of a given algorithm that maximizes some performance metric. As a combinatorial problem, algorithm configuration space is often enormous, and the tuning is notoriously time-consuming.

Many studies have attempted to help shorten the process. Most of these prior methods fall into three categories [6]: (1) using *racing procedures* to help eliminate candidate configurations that are significantly outperformed by other configurations [7–9], (2) using *stochastic local search (SLS)* methods to intelligently search the configuration space [10–12], (3) using *sequential model-based optimization* methods to build models to help quickly identify promising configurations [13,14]. These methods mainly aim at reducing the number of configurations needed to try to find appropriate configurations. In this work, we tackle the *k*-means configuration problem from the angle of computation reuse that is complementary to previous methods.

Reuse-centric optimization, especially center reuse, at a high level, shares a spirit with that of transfer learning, which stores the knowledge gained in solving the source task and apply it to other problems with similar properties. Both concepts are motivated by the fact that knowledge learned previously can help solve new problems faster or with better solutions [15]. This work materializes the high-level concept by answering some open questions on what knowledge is beneficial to reuse for *k*-means configuration, and how to reuse the knowledge effectively. The set of novel techniques it proposes are designed to leverage the specific properties of *k*-means configurations to address those open challenges.

3. Proposed techniques

We describe, in this section, our proposed techniques. Before then, we first discuss the factors and objectives necessary to consider in *k*-means configuration.

3.1. Overview of k-means configuration

Understanding the usage of *k*-means in real applications helps understand the purpose and objectives of *k*-means configuration.

Even though *k*-means is a clustering tool, it is often used as a module for a purpose beyond simple clustering. In *k*-means–based data classification, for example, through *k*-means, training data are grouped into clusters, which are then used for classifying test data: The cluster centers are used as compact representations of the data, and each center has an associated class label decided by its data-point members. The classification of a testing data point is then made to the class of its closest cluster center.

Fig. 1a outlines a general structure of applications that use *k*-means. Data are first projected onto some feature space. *k*-means clustering subsequently runs on the projected data to form some clusters, which are then used by the application for some follow-up purposes (e.g., classification).

K-means configuration is a process of finding the configuration (e.g., the number of clusters *k* and feature sets) that can maximize certain objectives. The objectives are often aimed at maximizing the quality of the ultimate results of the application (e.g., classification accuracy); some internal metrics of clustering (e.g., within and across cluster distances) could be relevant but are usually secondary to the application-level objectives. Cross-validation (e.g., on data classification) is often used in the process to help assess the quality of a configuration.

Fig. 1a also illustrates two important factors of *k*-means to configure and to impact the applications in some way. The first is the set of features to extract or select from the raw data, and the second is *k*, the number of clusters to form. Although the configuration involves only two factors, even with the fast Yinyang *k*-means algorithm [19], on a dataset of modest size, the configuration is still computationally intensive (days) when exploring all combinations of *k* and feature sets. There are some other factors that could also be worth tuning, such as the definitions of distance, the ways to do feature extraction. However, the two factors (*k* and feature sets) have the largest numbers of variants

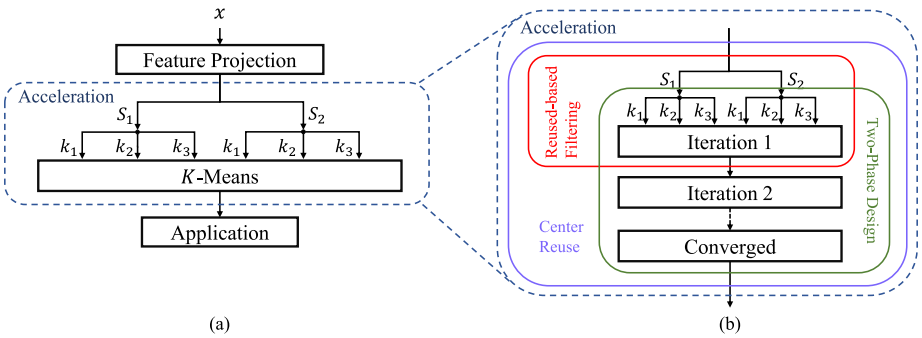


Fig. 1. Overview of *k*-means–based applications and where our three acceleration techniques are applied. (a) A general structure of *k*-means–based applications with *k*-means configuration. The dash-lined box represents the most time-consuming *k*-means clustering step our acceleration techniques apply on. (b) The details of the acceleration techniques including reused-based filtering, center reuse, and two-phase design. The dash-lined boxes illustrate the scopes each technique works on.

Table 1
Data statistics.

Dataset	Size(B)	n	#attributes	#classes	#one-hot dimensions	#PCA dimensions	*dstep*	*kstep*
Gamma	1.2e6	1.9e4	10	2	10	8	1	10
Sensorless	4.4e6	5.8e4	49	11	48	10	1	10
Credit [16]	3.0e6	3.0e4	24	2	23	13	1	10
Gassensor [17]	1.7e6	1.4e4	11	2	128	16	1	10
Miniboone	3.4e7	1.3e5	50	2	50	34	2	20
Adult	2.0e7	4.5e4	14	2	104	59	2	10
Connect	3.1e7	6.8e4	42	2	126	61	2	10
Activity [18]	1.1e7	1.0e5	561	6	561	157	8	10
Census	2.0e8	1.4e5	68	2	388	186	8	20

and hence dominate the configuration space. Our discussion in this paper focuses on them; thanks to the combinatorial nature of the space, the speedups attained for their configurations directly translate to the overall speedups of the whole configuration process despite the presence of other secondary factors one may wish to tune.

Our acceleration techniques pertain to the most time-consuming *k*-means clustering step, circled with a dash-lined rectangle in Fig. 1a.

3.2. Overview of the acceleration techniques

Our acceleration techniques consist of three stages: *reuse-based filtering*, *center reuse*, and a *two-phase design*. The first two materialize the idea of *reuse-centric k-means configuration*, which saves computations in the configuration process through promoting the reuse of computation results from the trials of some earlier configurations. The last technique uses a two-phase design to first quickly get an estimated surface of classification errors, and then uses it to help focus the explorations on valuable configurations. The first two are generally applicable for all *k*-means–based data mining tasks, while the last one is especially useful when a detailed relation between configurations and the final results of the application (e.g., classification accuracy) is needed.

The techniques work at different aspects of the problem and can function in synergy. The dash-lined boxes in Fig. 1b illustrate the scopes they each work on.

Reuse-based filtering reuses the clusters obtained in the trial of one configuration (with feature set S and k value) to speed up the first iteration of *k*-means in a later trial of some other configuration. It concentrates on the first iteration of *k*-means because in modern *k*-means (e.g., Yinyang *k*-means [19]), the later iterations are already highly optimized, and each takes a much shorter time than the first iteration does. For instance, in our experiments with nine datasets of different sizes and dimensions (listed in Table 1), the first iteration of Yinyang *k*-means takes 10%–40% of the entire *k*-means time.

Center reuse sets good initial centers for *k*-means by leveraging the centers from earlier trials. It works across all three levels: across the iterations in feature selection, iterations of k value exploration, and cross-validations. It significantly helps shorten the time for *k*-means to converge in the algorithm configuration.

The two-phase design aims at reducing the number of configurations to explore for each set of features. It hence contributes to the computational savings within, rather than across the explorations of a given set of feature.

Reuse-based filtering and the *two-phase design* do not alter the clustering results. *Center reuse* could lead to clustering results different from the ones attained by using random centers. However, later in Section 4.4, we will show that the influence causes negligible impact on the results of algorithm configurations.

We next explain each of the three techniques in detail.

3.3. Reuse-based filtering

K-means is time consuming primarily because of its calculations of the distances from data points to potential cluster centers. In the standard *k*-means, each iteration needs to compute $n \times k$ distances (n is the number of points, k is the number of cluster centers), from every data point to every cluster center in order to identify which cluster center is the closest to the data point. Modern *k*-means algorithms (e.g., Yinyang *k*-means [19]) successfully avoid many distance calculations in later iterations of a *k*-means, but they all still need the $n \times k$ distance calculations in the first iteration of *k*-means. In our experiments, we observe that the first iteration weights up to 40% of the entire *k*-means time. We call it the *first iteration problem*. Algorithm configuration of *k*-means needs many runs of *k*-means; every one of them suffers from the *first iteration problem*.

To alleviate the issue, we propose *reuse-based filtering*. It is based on the well-known geometric property of *Triangle Inequality (TI)*.

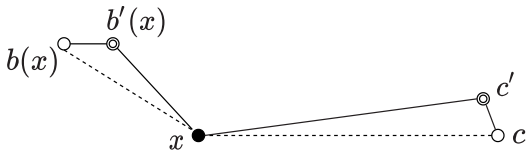


Fig. 2. Example of *Triangle Inequality*. Circles and double circles represent centers in current and previous iteration respectively, and $b(x)$ is the so-far nearest center of point x.

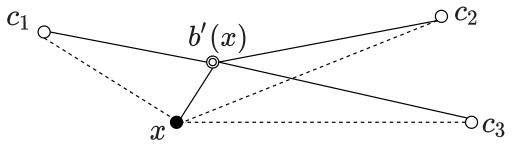


Fig. 3. Illustration of how the configuration with $k = k_1$ can help save distance computation in the first iteration of another configuration with $k = k_2$. $b'(x)$ is the closest center of point x when $k = k_1$; c_1, c_2, c_3 are the initial centers and $b(x)$ is the so-far nearest center of point x when $k = k_2$.

3.3.1. TI and its prior use for k-means

We provide the formal definition of TI and landmark as follows.

Theorem 3.1. *Triangle Inequality (TI): Let q, p, L be three points in a metric space (e.g. Euclidean space) and $d(x, y)$ be the distance between the any two points x, y in the space. Triangle Inequality states that $d(q, p) \leq d(q, L) + d(p, L)$. Point L is called a landmark.*

TI has been used by previous work [19–22] to avoid unnecessary distance calculations in k-means, except for its first iteration. The basic idea in those works is to use the cluster centers in the previous iteration as landmarks to help quickly attain the lower bounds and upper bounds between each data point and the new centers in the current iteration. We take Fig. 2 as an example. Circles and double circles represent centers in current and previous iteration respectively, and $b(x)$ is the so-far nearest center of a point x. If the lower bound $lb(x, c)$ between x and a center c is even larger than the upper bound $ub(x, b(x))$ between x and its so-far nearest center $b(x)$ in this current iteration, there is no need to calculate $d(x, c)$. According to TI, we know that,

$$ub(x, b(x)) = d(x, b'(x)) + d(b(x), b'(x)) \tag{3.1}$$

$$lb(x, c) = |d(x, c') - d(c, c')| \tag{3.2}$$

So if $lb(x, c) > ub(x, b(x))$, there is no need to calculate $d(x, c)$ and k-means should be able to skip the center c directly, since the lower bound of the distance between x and x is even greater than the upper bound of the distance between x and $b(x)$, which makes c cannot be closer to x than $b(x)$. Notice that this idea has not been applied to the first iteration of k-means because there is no previous iteration that it can leverage.

3.3.2. Basic idea of reuse-based filtering

Our *reuse-based filtering* is inspired by the prior use of TI in k-means. Its basic approach is to leverage the results from the exploration of an earlier configuration to help produce the lower/upper bounds of distances for the exploration of later configurations, whereby, TI can then be applied to identify and avoid the unnecessary distance computations.

The nature of algorithm configuration poses several special challenges for materializing the idea that do not appear in the prior use of TI for accelerating a single run of k-means.

- In different iterations of a single k-means, distances are all based on the same set of data features, and the number of cluster centers is also identical. But in algorithm configuration, these factors could all differ in the exploration of different configurations. That causes complexities in how to reuse distances, and how to effectively define landmarks.
- How to ensure that the acceleration to the first iteration does not interfere with the acceleration of the later iterations of k-means. When modern k-means algorithms apply TI to later iterations to avoid unnecessary distance calculations, they leverage the $n \times k$ distances from the first iteration to help attain some tight distance bounds for TI to work effectively [19–22]. If *reuse-based filtering* avoids computing many of the distances in the first iteration, it could pose risks for the acceleration of the later iterations to work properly.

We next explain how our design of *reuse-based filtering*, and how the design addresses the two special concerns.

3.3.3. Detailed design of reuse-based filtering

We explain the design of reuse-based filtering in two levels: across k and across feature sets.

Reuse across k. This reuse happens among the configurations that share the same set of features, with different k values. Suppose that the reuse is from one configuration with $k = k_1$ to another with $k = k_2$. Compared to the previous usage of triangle inequality to eliminate unnecessary distance computations, as shown in Fig. 2, we cannot build that one-to-one previous-center relationship between two configurations with different k. Instead, we could use the closest center $b'(x)$ for each point x in the configuration with $k = k_1$ as the landmark for all the initial centers in the configuration with $k = k_2$. Fig. 3 provides the illustration. At the beginning, the so-far nearest center $b(x)$ of point x when $k = k_2$ can be set as any initial center in the current configuration. Here we set it as c_1 in Fig. 3. Then the upper bound $ub(x, b(x))$ and the lower bound $lb(x, b(x))$ between x and $b(x)$ are just the exact distance $d(x, b(x))$.

$$b(x) = c_1, ub(x, b(x)) = lb(x, b(x)) = d(x, b(x)) \tag{3.3}$$

So for each possible center $c = c_2, c_3$, we can calculate the corresponding upper bound $ub(x, c)$ and lower bound $lb(x, c)$ according to TI.

$$ub(x, c) = d(b'(x), c) + d(x, b'(x)) \tag{3.4}$$

$$lb(x, c) = |d(b'(x), c) - d(x, b'(x))| \tag{3.5}$$

If $lb(x, c) > ub(x, (b(x)))$, the center c can be skipped as the previous example in Fig. 2. If $lb(x, b(x)) > ub(x, c)$, the nearest center $b(x)$ of point x should be set as c since the center c must be closer than the current nearest center $b(x)$. If none of the two cases are satisfied, the distance $d(x, c)$ between the center c and the point x needs to be calculated to compare with $d(x, b(x))$ to decide which one is closer.

Note that the distance from x to $b'(x)$ can be directly reused from the trial with $k = k_1$, the extra distance computations we need to carry out are those from the centers in $k = k_1$ to the initial centers in $k = k_2$. In total, there are $k_1 \times k_2$ distance computations and a small part of distance computations from points to centers, which is far less than the cost of distance computations from every point to every center (i.e. $n \times k_2$), where n is the total number of points. Similarly to the previous usage of triangle inequality [19,22], this optimization does not change the final cluster results, as distance computations to some center c will be eliminated only when c cannot be the closest center to the point x.

Reuse across feature sets. This reuse happens among the configurations that have the same number of clusters k, but

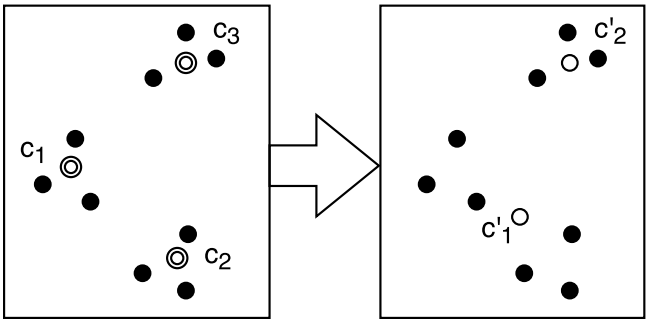


Fig. 4. Illustration of center reuse across k. The two graphs represent the k-means on two configurations with k equaling three (left) and two (right) respectively. The double circles in the left graph show the three centers attained in the exploration of earlier configuration. These centers are grouped to get two group centers c'_1, c'_2, which are then used as the initial centers (marked as circles in the right picture) for exploring the latter configuration.

use different sets of features. As our optimization is based on triangle inequality, which requires the distance to be defined in the same metric space, we need to be conservative about distance reuses across different feature sets. Before we give the detailed explanation about how *reuse across feature sets* is applied, we first introduce a theorem on distances defined in two different, but highly related, metric spaces.

Theorem 3.2. *For any two pairs of points (x^{F_1}, c^{F_1}) and (x^{F_2}, c^{F_2}), where x^{F_1} and c^{F_1} are in feature space F_1 with the feature set S_1, while x^{F_2} and c^{F_2} are in feature space F_2 with the feature set S_2. If x^{F_2} and c^{F_2} have only a subset dimensions of x^{F_1} and c^{F_1} respectively, i.e., $S_2 \subset S_1$, then the distance between x^{F_2} and c^{F_2} must be no larger that between x^{F_1} and c^{F_1} in any p-norm space. That is to say, $d(x^{F_2}, c^{F_2}) \leq d(x^{F_1}, c^{F_1})$.*

The theorem directly follows the distance definition in any p-norm space, in that the distance function monotonically increases with the number of dimensions.

For distance reuse across feature sets, a distance computed in feature space F_1 can be used for bound computations in feature space F_2 without affecting the final clustering result as long as S_2 is a subset of S_1. In particular, for each center c'^{F_1} obtained in feature space F_1, we remove those dimensions that are not used in F_2 to build a corresponding center c'^{F_2} in feature space F_2.

Fig. 3 can also serve as an illustration of how distance reuse can help eliminate unnecessary distance computations to some center c across feature sets, where $b'(x)$ is computed from the closest center of point x in feature space F_1, while c_1, c_2 and c_3 are the initial centers in feature space F_2. Similarly, the so-far nearest center $b(x)$ of point x in feature space F_2 can be set as any initial center, such as c_1, in the current configuration at the beginning, and the upper bound $ub(x, b(x))$ and the lower bound $lb(x, b(x))$ between x and $b(x)$ are just the exact distance $d(x, b(x))$. Then for other possible centers $c = c_2, c_3$, their upper bound $ub(x, c)$ and lower bound $lb(x, c)$ can be computed based on TI. Compared to the reuse across k, our two bound computations directly use $d(x^{F_1}, b'^{F_1}(x))$ calculated in feature space F_1, which themselves are the upper bounds of $d(x, b'(x))$ in feature space F_2 since $S_2 \subset S_1$.

$$ub(x, c) = d(b'(x), c) + d(x, b'(x)) \leq d(b'(x), c) + d(x^{F_1}, b'^{F_1}(x)) \tag{3.6}$$

$$lb(x, c) = d(b'(x), c) - d(x, b'(x)) \geq d(b'(x), c) - d(x^{F_1}, b'^{F_1}(x)) \tag{3.7}$$

The following logical judgment is the same as what we did in the reuse across k. These bound computations are a simple extension of the traditional triangle inequality, and as a consequence, our method will remove the distance computation to some center c, only if it is impossible to be the closest center to the point x.

Further, our accelerations based on both reuse across k and reuse across feature space can easily be combined with previous accelerations focusing on the later iterations of k-means [19,22]. Instead of using the exact distance results for bound computation, we can easily replace the exact distance with corresponding bounds obtained in our optimized first iteration. Although our optimization may theoretically affect the efficiency of the optimization applied in the later iteration of k-means, our empirical experience shows that these two accelerations are mostly orthogonal to each other. (Details in Section 4.5.3.)

3.4. Center reuse

The second technique we propose for accelerating configuration of k-means is called *center reuse*. The idea is simple. As is well known, the convergence speed of k-means is highly sensitive to the quality of the initial centers. Some initial centers can make k-means converge in much fewer iterations than others. The basic idea of *center reuse* is to use the cluster centers attained in the exploration of some earlier configuration as the initial centers for the explorations of later configurations.

Center reuse is based on the following hypothesis:

Hypothesis 3.3. In algorithmic configuration, effectively using centers from an earlier run of k-means to initialize later runs of k-means could shorten the convergence process while causing negligible effects on the result of the algorithm configuration.

Specifically, we consider *center reuse* in three scenarios, corresponding to the different levels of the explorations of k-means configurations shown in Fig. 1b.

Reuse across validations. This reuse is among the different folds in cross validations in the exploration of a certain configuration. As aforementioned, recall that when exploring a given k and a set of input features, cross-validation is often used to examine the quality of the final results when that configuration is used. Take k-means–based classification as an example: cross-validation computes the errors of the classifier produced through k-means in that configuration. A V-fold cross validation builds V classifiers with each on a slightly different training dataset. The *center reuse* at this level is to use the cluster centers attained during the training of the first of the V classifiers as the initial centers for the k-means in the constructions of the other $V - 1$ classifiers.

Reuse across k. This reuse happens among the configurations that share the same set of features, but different k values. Because of the difference in k, the centers may not be directly reusable. Our empirical investigation shows that the problem can be handled through a simple design. Suppose that the reuse is from one configuration with $k = k_1$ to another with $k = k_2$. If $k_2 > k_1$, in addition to using the centers attained in exploring the earlier configuration, we add $k_2 - k_1$ randomly generated centers as needed. If $k_2 < k_1$, we cluster the k_1 centers into k_2 groups and then take the group centers as the initial centers for the exploration of the latter configuration. Fig. 4 illustrates this case.

Reuse across feature sets. This reuse happens among the configurations that use different sets of features. Suppose that the reuse is from configuration C_1 with feature set S_1 to configuration C_2 with feature set S_2. The differences in the feature sets make direct center reuse difficult. Our design is to reuse the values of overlapped features between S_1 and S_2, and generate the values of the other features of S_2 (if there are any). Our experiments show that the generation can be as simple as using the mean value of each feature.

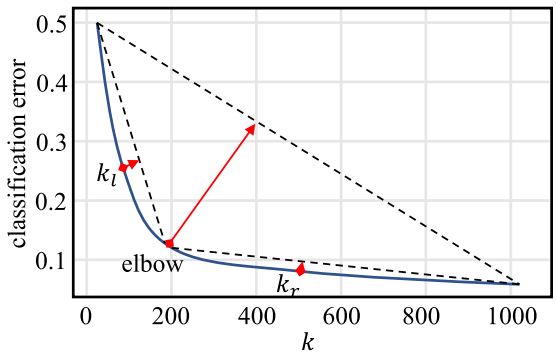


Fig. 5. An example of an error curve and the illustration of curve segmentation.

When two configurations differ in both k and feature sets, *center reuse* first applies the *reuse across feature sets* to handle the dimension differences in the features, and then applies the *reuse across k* to set the initial cluster centers.

Our empirical investigation shows that *center reuse* is beneficial for accelerating k-means configuration – that is, Hypothesis 3.3 holds – when the following principles are followed:

- Reuse cross validations should be always applied. Even though the training datasets of different validations differ, the datasets come from the same source and share the same distributions. Their cluster centers are hence similar. The reuse can always shorten the convergence process significantly.
- Reuse across k can be applied to either direction: from a smaller k to a larger one or from a larger k to a smaller one. In the case of increasing k values, as the extra centers are added randomly, the complexity is the same compared with the random initialization. In the case of decreasing k values, we group centers to get initial centers using k-means with only one iteration. The overhead is also negligible. Even if the two k values differ a lot, center reuse-based initialization degrades to random initialization and will not bring extra costs. Specifically, in our experiments, we notice that center-reuse always give substantial benefits even when the two k values differ up to 25% of the maximum k value (e.g. the difference is 200 and the maximum k is 1000). Thus, reuse across k can always be applied.
- Reuse across feature sets are applied to a different set of features. Since we randomly generate the values for dimensions that are not overlapped between the two feature sets, reuse across feature sets degrades to random initialization in the worse case (i.e., two features sets are completely different). If PCA-based step-wise feature selection is used, there is always overlap between two features sets. Thus reuse across feature sets can always be applied. We observed that reuse across feature sets can give substantial benefits even when the two features sets differ in 15% of the maximum dimension (e.g. the difference is 8 and the maximum dimension is 60).

Section 4 will provide the details of our empirical experiments on the effectiveness of *center reuse* in saving computations, and on the effects it has on the quality of k-means configuration.

3.5. Two-phase design

The two techniques presented so far form the basis for our reuse-centric k-means configuration. In this part, we introduce another complementary technique named *two-phase design*.

This technique is particularly useful when the error surface of the target application is desired. For k-means–based classification, for instance, the error surface indicates how the classification error changes when k and feature sets change. The surface is composed of a set of error curves, with each corresponding to one set of input features. The black curve in Fig. 5 illustrates such an error curve when a particular feature set is considered while k changes.

The error surface is useful when the criterion for the best configuration varies. For instance, k-means–based classifier tends to give a higher classification accuracy when k gets larger. But at the same time, the classification time also gets longer. In some situations, a user may want different trade-offs between the accuracy and the time in different scenarios. Having the error surface in hand can help meet the needs without rerunning the algorithm configuration every time the desired trade-off changes.

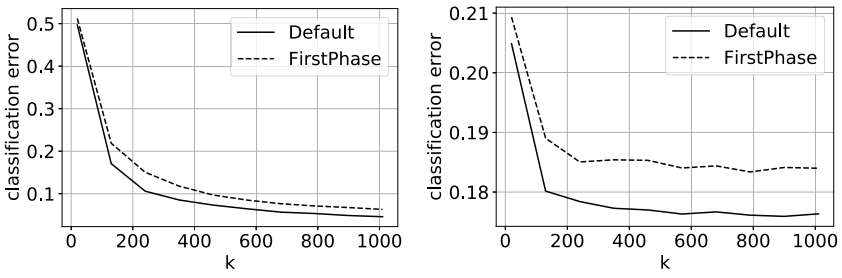
The *two-phase design* is based on the observation that some points on an error surface more critically affect the accuracy of the error surface than some other points do. For instance, on the curve shown in Fig. 5, the parts outside the elbow area are close to straight lines, and are hence easy to approximate through curve interpolation on only several sampling points, but the elbow area is more subtle in shape, and would require more sampling points to get a reasonable interpolation result. Therefore, how to identify elbow point to improve the accuracy of error curves is a key point here. Meanwhile, the elbow point can also be selected as the desired configuration for its relatively low classification error and running time reflected by k.

The idea of the *two-phase design* is to first quickly obtain an estimated shape of the error surface, based on which, it then conducts a focused exploration of the configurations (e.g., those fall into the elbow area) that are most important for the accuracy of the final error surface. The two phases in the design happen during the exploration of each given set of input features. To ease the understanding, we explain the technique and how it works by drawing on k-means–based classification as an example use of k-means.

The first phase in the design, specifically, tries to quickly get the approximate classification errors at a small number of sampled k values. It employs both the *reuse-based filtering* and *center reuse* for speed. At the same time, it adopts two approximation methods. The first is to use only the first fold of cross-validation; the second is to replace k-means clustering with only a one-step clustering. The one-step clustering assigns points to clusters based on their distances to the centers produced from our *center reuse* scheme; no center updates or point reassignments are done. The rationale of our first phase approximation method comes from the statistical similarity across different folds of a dataset, and the reasonable quality of the cluster centers produced from the *center reuse*.

These two approximation methods could incur some deviation from the exact classification errors, but as we observed, their Pearson correlation coefficient with the errors from Yinyang K-mean are always higher than 0.95 for all datasets we tested. Fig. 6 shows approximated classification error curves from the first phase for two example datasets with ten k values sampled from the range [20, 1010]. "Default" here refers to the default k-means with random initialization while "FirstPhase" refers to our first phase approximation method. Even though the errors from the first phase are higher than the errors from standard k-means, the trends of the error curves are well approximated.

Based on that shape of curve, the second phase identifies the critical sections (i.e., the elbow section) of the curve and selects some important configurations to conduct more focused and detailed explorations to subsequently get the precise accuracy at those points. Through interpolation across those points,



(a) sensorless (b) adult

Fig. 6. Approximated classification error curves from the first phase for two datasets.

it finally obtains the error curve. Compared to uniform sampling, this two-phase design allows better error curves to be attained with detailed explorations of fewer configurations.

Two notes are worth making about the second phase. The first is how to identify the critical sections. We call it also *curve segmentation*. Given the range of k, $[k_{min}, k_{max}]$, the way we segment the curve and do non-uniform sampling is as follows:

1. Find the elbow point on the curve. k_{elbow} is the corresponding k value;
2. For the two sub-ranges $[k_{min}, k_{elbow}]$ and $[k_{elbow}, k_{max}]$, find the elbow point for each partial curve in the range. The corresponding k values are k_l and k_r;
3. Then the range is split into three parts: $[k_{min}, k_l]$, $[k_l, k_r]$, and $[k_r, k_{max}]$. Different stepsizes can be used when exploring the three sub-ranges.

Let s_1, s_2 and s_3 be the stepsize for sampling the range $[k_{min}, k_l]$, $[k_l, k_r]$, and $[k_r, k_{max}]$ respectively. We set the step sizes of the sampling as follows:

$$s_2 = \frac{m_2(k_l - k_{min}) + m_1 m_2(k_r - k_l) + m_1(k_{max} - k_r)}{m_1 m_2(n_k - 1)}, \tag{3.8}$$

$$s_1 = m_1 s_2, \tag{3.9}$$

$$s_3 = m_2 s_2, \tag{3.10}$$

where n_k is the total number of k values to sample, while m_1 and m_2 are parameters determining the degree of discrimination in sampling to the different segments. When $m_1 = m_2 = 1$, we have $s_1 = s_2 = s_3 = (k_{max} - k_{min})/(n_k - 1)$, which is equivalent to do uniform sampling. When $m_1 > 1$, we have $s_1 > s_2$, which means the range of $[k_{min}, k_l]$ will have larger step size of sampling than the range of $[k_l, k_r]$. Otherwise, the range of $[k_{min}, k_l]$ will have smaller step size of sampling than the range of $[k_l, k_r]$. The same thing happens to m_2 as well. Since the error curve usually has a typical shape from steep to gentle as shown in Fig. 5, which suggests to use smaller step size in the range of $[k_{min}, k_l]$ and larger step size in the range of $[k_r, k_{max}]$, m_1 should be smaller than 1 while m_2 should be greater than 1. Finally we set them to 0.5 and 2 respectively through experiments.

The second note is about elbow point detection. While the notation of elbow points is well-known, there are no broadly accepted definitions. Various techniques [23,24] have been proposed to detect the knee point of a curve. We adopt a lightweight approach similar to [24]. The idea is to draw a line from the first to the last point of the curve, and then find the data point that is farthest away from that line. Fig. 5 illustrates the process of detecting knee points of an error curve and shows the boundaries k_l and k_r of the curve segments obtained through the method.

4. Evaluations

We conduct a series of experiments to evaluate the proposed techniques. Specifically, we focus on the following questions:

- Q1: How much computation can the optimizations save? How much can they speed up k-means configuration, in both sequential and parallel settings?
- Q2: Do the optimizations degrade the configuration results?
- Q3: How are the benefits affected by dataset attributes and problem settings (e.g., k, data dimensions, landmarks)?
- Q4: How to deploy the optimizations (e.g., reuse from smaller k or from larger k, smaller feature sets or larger feature sets) to maximize the benefits?

This section answers the first question by reporting the overall computation savings and speedups brought by the techniques in Sections 4.2 and 4.3 , answers the second question in Section 4.4, answers the third and fourth ones through some sensitivity studies in Section 4.5.

4.1. Methodology

Our experiments use k-means–based classification as a concrete usage scenario of k-means configurations. It is worth noting that the techniques are general, applicable to other usage of k-means.

All the experiments run on an HPE Apollo 2000 server with two Intel Haswell CPUs (14 cores/CPU, 2.3–3.3 GHz) and 128 GB RAM. We use nine large, real-world datasets taken from the UCI machine learning repository [25]. The statistics of the datasets including data size (size), the number of instances (n), the number of attributes (#attributes) and the number of classes (#classes) are listed in Table 1. Note that the attributes here are raw data, so we need to apply one-hot encoding on them firstly. The range of one-hot feature dimensions is shown in the "#one-hot dimensions" column. Then we used PCA as the feature projection method to extract features from one-hot features and adopted the step-wise feature selection method to select feature sets. We retained a maximum number of components that cumulatively explain 99% of variation. The minimum number of dimensions is two. The overall range of feature dimensions to consider in the configuration explorations is shown in the "#PCA dimensions" column. "*dstep*" and "*kstep*" columns show the default step size to increase or decrease the dimension and the number of clusters respectively.

Yinyang k-means [19] is used in the baseline implementations of the algorithm configuration to minimize time. Yinyang k-means is one of the state-of-the-art algorithms proposed by Ding and others recently. The algorithm filters unnecessary distance calculations by using continuously maintained lower bounds on the distances of each point to the cluster centers as well as an upper bound to the cluster center to which it was assigned. Even though the bounds yield a significant speedup compared to the standard k-means (over 9X on average) [19], the algorithm needs to compute one full iteration of k-means to initialize bounds in the beginning, which requires the computation of all distances to the centers.

Table 2
Speedup (X) on baseline *K*-means.

Dataset	Time(s)[a]	Reuse-based filtering	Center reuse	
			Across validations	Across k and feature sets
Gamma	2808.1	3.09	3.58	2.05
Sensorless	10730.6	3.22	3.40	1.73
Credit	5713.5	3.30	4.05	2.04
Gassensor	1901.9	4.37	4.39	2.33
Miniboone	56636.8	1.56	3.73	1.83
Adult	9904.4	4.30	5.80	2.18
Connect	23621.7	1.54	4.42	1.63
Activity	6569.6	1.11	2.76	1.91
Census	79872.7	2.30	4.14	1.68

[a]Time(s) refers to *k*-means clustering time in seconds for all 200 configurations without our optimizations.

K-means++ [26] is a commonly used initialization method for better approximation to the optimal *k*-means solution (i.e., minimizing the within-cluster sum of squares). However, our experiment shows that for configuring *k*-means–based data classification, it does not outperform random-based center initialization in either the speed or the quality of the produced classifier. Random initialization is hence used in our baseline.

Euclidean distance is used in all the experiments as selections of various distance metrics are not a focus of this work. Our acceleration techniques apply to various metric spaces, except that cross-feature reuse-based filtering requires p-norm spaces as stated in Theorem 3.2.

4.2. Speedups on heuristic search

This part reports the speedups brought to the *k*-means configuration process by our *reuse-based filtering* and *center reuse*. Algorithm configurations typically employ some heuristic search algorithms to explore the configuration space. Our optimizations are largely orthogonal to what search algorithms are used. Our experiments use stochastic hill climbing. Hill climbing is an iterative algorithm that starts with an arbitrary solution and then attempts to find a better solution by changing the solution in some way. If the change produces a better solution, then it is taken as the new solution. Otherwise, a new change is examined. The process is repeated until no further improvements can be found or some stopping criterion is met. Stochastic hill climbing makes the change at random.

The tuning objective is set to consider both the classification accuracy and the response time of the built classifier. Specifically, it is to find the smallest k (hence giving the fastest classification response) that can achieve a classification accuracy over a given threshold (e.g., 90% given by users). The stopping criterion is that the maximum number of configurations (200) is tested. The baseline method repeats the following process until it meets the stop criterion:

1. Choose the number of dimensions d and the number of clusters k from the search space randomly;
2. Run *k*-means–based classification with the specified configuration and get the average classification error through 10-fold cross-validation;
3. If the average classification error reaches a predefined error threshold (1 - accuracy threshold) and the k value is smaller than that in the current best configuration, then take it as the current best solution.

Our computation reuse techniques accelerate the second step. Each randomly generated configuration could have different d and/or k than what a previous trial uses. We design the following reuse strategy to select a historic trial for computation reuse:

- If there are previous trials of *k*-means that use the same d as the current one, then we reuse the distances from the trial with the largest k for reuse-based filtering and the cluster centers from the trial with the nearest k for center reuse.
- Otherwise, we reuse the distances from the trial whose d is larger but has the least difference than the current one for reuse-based filtering and the cluster centers from the trial with the least difference on d (can be smaller or larger) for center reuse.

To enable the above reuse strategy, the cluster centers, the data points distribution, and the distance bounds from every data point to its cluster center from previous trials of *k*-means in the first fold of cross-validation have to be stored. Also, the point-to-center distances from previous trials of *k*-means needs to be updated whenever a configuration with a larger k and the same d is tested. In experiments, our method tests from the largest d and the largest k and apply reuse to smaller d values and k values, so it is necessary to store only one setting of previous result. The concrete storage space needed depends on the data structure used. In particular, a floating-point list is used for store cluster center coordinates, and a list of point index and point-to-center distance pairs is for point distribution and distance bounds in our experiments.

The speedups (baseline time/running time) from each technique are listed in Table 2. For reuse-based filtering, we observe that it has only minor effects on the speeds of the later iterations of Yinyang *k*-means (Details in Section 4.5.3). So we report the speedup of reuse-based filtering for the first iteration. The speedup comes from the distance computation saved by TI-based filtering. Center reuse affects the entire *k*-means clustering and thus has the dominant influence on the overall speedup. It confirms that our simple design for center reuse across k and features sets works well. The next subsection reports the overall effects when all the techniques are used together in attaining error surfaces.

4.3. Speedup on the attainment of error surfaces

Our second experiment studies the benefits of our techniques on the attainment of classification error surfaces. As Section 3.5 has mentioned, error surfaces capture the relations between configurations and the errors of the corresponding classifiers. They could be helpful when the criterion for the best configuration varies. In this experiment, we apply all three techniques that Section 3 proposes to accelerate the attainment of the error surfaces.

Uniform search of the configuration space is a simple but frequently used method to attain error surfaces. It is used in the baseline implementation. Uniform search evaluates every combination of all the d values and several uniformly sampled k values.

Table 3
Speedup on the attainment of error surfaces.

Dataset	Speedup (X) by computation reuse ($\#k = 32$)					Percentage of k saving			Overall speedup (X)		
	Reuse-based filtering		Center reuse			Two-phase design			$\#k = 8$	$\#k = 16$	$\#k = 32$
	Across k	Across feature sets	Across validations	Across k	Across feature sets	$\#k = 8$	$\#k = 16$	$\#k = 32$			
Gamma	3.42	3.12	4.81	2.78	1.44	11.51%	29.52%	16.81%	4.57	6.30	5.70
Sensorless	3.61	3.80	4.50	2.37	1.59	21.41%	31.00%	24.21%	5.34	6.73	6.94
Credit	3.65	3.70	5.31	2.81	1.82	6.69%	23.82%	15.09%	4.98	6.64	6.77
Gassensor	4.31	4.34	5.75	3.32	3.19	26.54%	38.77%	11.95%	6.54	8.59	6.74
Miniboone	1.91	1.97	4.56	2.57	2.02	3.85%	49.24%	48.98%	4.50	8.24	9.17
Adult	4.65	5.05	6.43	3.86	3.88	12.99%	24.64%	22.71%	6.76	8.27	9.07
Connect	1.59	1.74	4.13	2.91	2.28	11.19%	13.88%	5.89%	4.58	5.07	4.98
Activity	1.21	1.23	2.77	1.84	2.07	15.25%	17.36%	13.79%	3.02	3.49	3.34
Census	2.71	2.78	4.89	2.63	2.61	16.57%	33.12%	28.91%	5.34	7.32	7.64

Table 4
Speedup in parallel settings.

Dataset	#threads	Speedup (X)			
		$\#k = 5$	$\#k = 10$	$\#k = 20$	$\#k = 30$
Gamma	2	3.66	4.06	4.53	4.60
	4	3.59	4.13	4.35	4.52
	8	3.26	3.70	4.13	4.19
	16	2.91	3.18	3.23	3.29
Sensorless	2	3.85	4.23	4.33	4.58
	4	2.23	3.16	4.08	4.58
	8	1.69	3.72	3.84	4.02
	16	1.40	3.46	3.84	3.67
Credit	2	3.73	4.01	4.77	5.12
	4	3.96	4.49	4.89	5.08
	8	3.71	4.40	4.60	4.88
	16	3.69	4.02	4.30	4.75
Gassensor	2	4.32	4.87	5.26	5.42
	4	4.27	4.69	5.02	5.63
	8	3.69	4.47	4.66	4.90
	16	2.55	3.95	4.06	4.75
Miniboone	2	3.92	4.12	4.38	4.57
	4	3.86	4.14	4.16	4.27
	8	3.70	3.87	4.22	4.22
	16	3.97	4.29	4.54	4.71
Adult	2	5.65	6.01	6.45	6.68
	4	5.48	5.32	5.98	6.67
	8	4.87	5.64	6.06	6.31
	16	2.41	3.06	5.77	5.94
Connect	2	4.01	4.11	4.34	4.40
	4	4.01	4.24	4.39	4.51
	8	3.78	4.00	4.31	4.34
	16	3.68	3.68	4.15	3.13
Activity	2	2.37	2.45	2.58	2.64
	4	2.32	2.42	2.51	2.58
	8	2.15	2.24	2.39	2.42
	16	2.15	2.27	2.44	2.50

Our proposed method is to use two-phase design to reduce the number of k to be evaluated for building a classification error curve. Computation reuse techniques are applied to save the clustering time for each sampled configuration. Since the configurations for uniform search to be tested are already known, our method starts with the largest d and the largest k and apply reuse to smaller d values and k values.

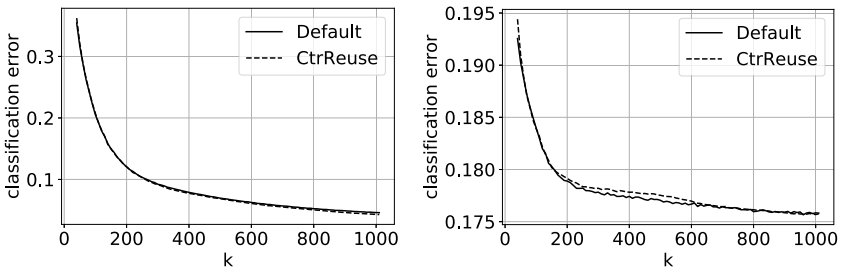
We first apply all the three techniques to speed up sequential uniform search, and then apply them to parallel uniform search on the 28-core parallel machine.

4.3.1. Sequential

In this setting, only one thread is used for the configuration process. The range of dimensions to search is listed in Table 1 and the range of k values is [20, 1010]. The number of sampled k in uniform search are 8, 16, 32.

The speedups from all three techniques are listed in Table 3. For reuse-based filtering and center reuse, the speedups, namely the ratio between baseline time and our running time, are reported, while for two-phase design, the percentage of k saving is reported. It can be seen that the running time of k-means is decreased largely with reuse-based filtering and center reuse, and we could reduce the number of k to be evaluated for recovering the error curve without affecting the benefits from the computation reuse with our two-phase design. As a consequence, the overall speedup with all three techniques is up to 9.17X.

The overall speedup on the dataset *activity* is not as high as on the other datasets. Specifically, the dominated acceleration factor, center reuse across validations, produces a smaller speedup compared with that on the other datasets. In contrast, the results from the dataset *census*, which has the same large dimensions but about 14 times larger data size, shows much larger speedups for reuse-based filtering and center reuse across



(a) sensorless (b) adult

Fig. 7. Classification error curves for two datasets.

all three levels. This is because when a dataset is relatively small but has a large size of feature sets, training sets are likely to follow different distributions and thus the centers resulting from one fold of training set is not as good for another fold of training set. Another special phenomenon is the percentage of k saving on the dataset *miniboone*. It can be seen that when the number of k used by the uniform sampling changes, the percentage of k saving by using two-phase design increases from 3.8% to around 49%. This situation occurs because the error curve of *miniboone* has a relatively large linear area, namely the curve segment of $[k_{min}, k_l]$ and $[k_r, k_{max}]$, which needs only a small number of k to approximate the error curve obtained by the uniform sampling in our two-phase design. Therefore the percentage of k saving becomes significant when the uniform search needs more k to construct the error curve.

4.3.2. Parallel

The parallel results are interesting to examine because our techniques, especially the computation reuses, bring data dependence to the exploration of different configurations: for a configuration to reuse results from another, it has to wait for the results to be produced. They hence could hamper the parallel search.

Table 4 presents results of our algorithms in parallel settings when the two computation reuse techniques are applied. In order to run our algorithms in parallel, some dependencies incurred by the computation reuse have to be removed. When scheduling the task to each thread, we use the following strategy to break dependencies: if the number of threads supported is no larger than the number of feature sets to be evaluated, then only remove dependencies caused by reuse across feature sets. Each thread examines a subset of feature sets and the entire sampled k values. The larger the dimension is, the longer the k-means clustering time is. To balance the workload among the threads, we assign each thread feature sets in an alternating manner. For example, if the dimensions are from two to five and the number of threads is two, then the first thread runs dimensions two and four while the second thread runs dimensions three and five. When the thread supported is larger than the number of feature sets, some dependencies caused by reuse across k are also removed.

As shown in Table 4, we have good speedups with various numbers of threads and numbers of sampled k. The larger the number of sampled k is, the larger the speedup is. This is because we have a larger ratio of reusable distance computation for a larger set of sampled k.

4.4. Quality influence of center reuse

Recall that among all the three optimizations we introduce, only center reuse might affect the quality of the resulting classifiers due to the sensitivity of k-means on initial centers. In this part, we provide the details of our empirical measurements of the effects of center reuse on the quality of both classification and clustering results.

Since k-means is sensitive to initialization, we perform 100 runs of k-means–based classification with different random seeds for each configuration. The classification error is averaged over the 100 runs. The metrics we use to measure the discrepancy between the error curve from the baseline and that from our center-reuse based technique are Mean Absolute Error (MAE) and Mean Percent Error (MPE).

Given a list of values $[a_1, a_2, \ldots, a_n]$ and its approximation $[\tilde{a_1}, \ldots, \tilde{a_n}]$, MAE and MPE are defined as follows:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |a_i - \tilde{a}_i|}{n} \tag{4.1}$$

$$\text{MPE} = 100\% \times \frac{\sum_{i=1}^{n} |(a_i - \tilde{a}_i)/a_i|}{n} \tag{4.2}$$

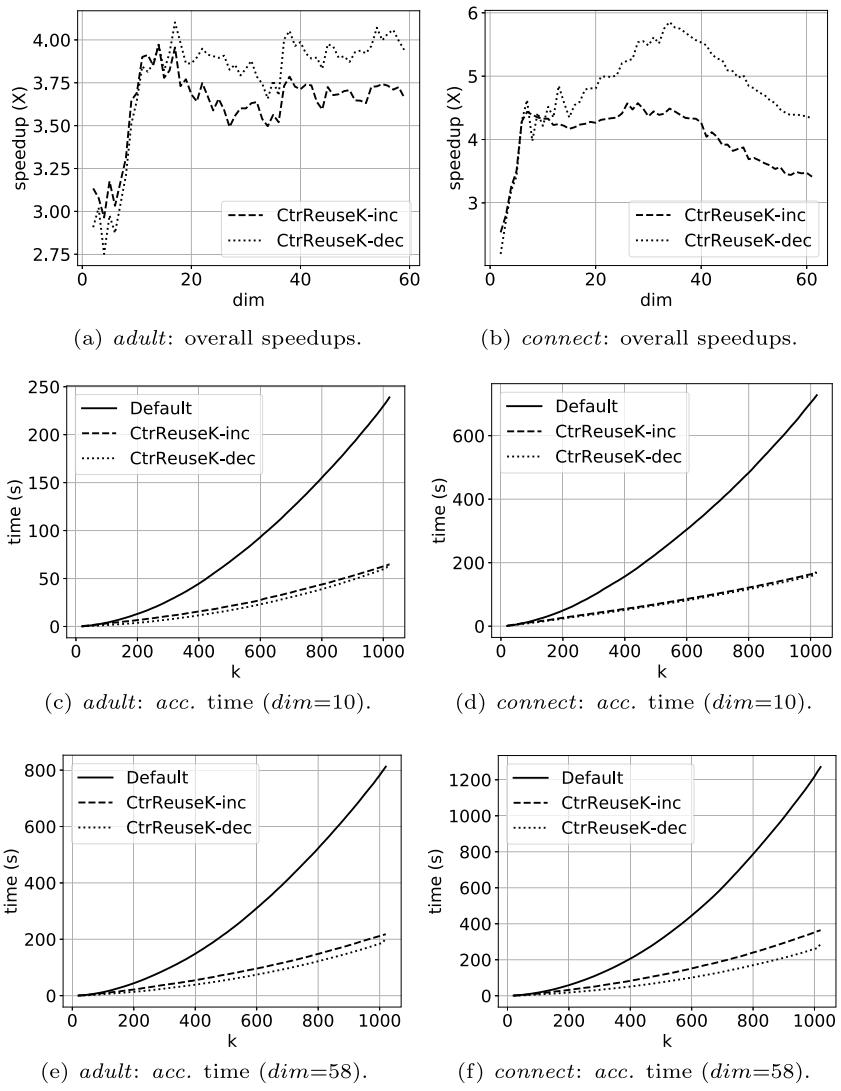
For the datasets in Table 1, the range of MAE is from $2.67E-04$ to $1.75E-03$ and the range of MPE is from 0.15% to 2.63%. Fig. 7 shows the classification error curves from two datasets. "Default" here refers to the Yinyang k-means with random initialization while "CtrReuse" refers to the Yinyang k-means with center-reuse initialization. Even though center reuse yields a different error curve, the MAE is lower than 0.002 and the MPE is lower than 3%, indicating the little influence of center reuse on classification quality.

We also validated the minor influence of center reuse on clustering quality through traditional internal metrics including Davies–Bouldin index [27], Dunn index [28], and Silhouette coefficient [29]. Since the objective of k-means is to minimize the within-cluster sum of squares (WCSS), we also included WCSS as a metric. We used Pearson correlation coefficient (PCC) to measure the correlation of k-metric value curves with center-reuse initialization and those with random initialization. For the datasets in Table 1, the PCCs are higher than 0.969 for all the indexes and higher than 0.9997 for WCSS. The MPE for all the indexes and WCSS are mostly within 5%. The results indicate that center reuse also has negligible influence on clustering quality.

4.5. Sensitivity analysis and insights

Experiments on heuristic search and the attainment of error surfaces have shown the efficacy of our acceleration techniques. To better take advantage of reuse-based optimization, we provide some insights by answering the following questions:

- For center reuse across k, should we reuse centers resulting from a smaller k or a larger k?
- For center reuse across feature sets, should we reuse centers resulting from a smaller feature set or a larger feature set?
- For reuse-based filtering, how does the number of landmarks affect the speedup? How does the optimization affect later iterations of k-means? How is the speedup related with k and the size of feature sets?



(a) *adult*: overall speedups. (b) *connect*: overall speedups.

(c) *adult*: *acc.* time (*dim*=10). (d) *connect*: *acc.* time (*dim*=10).

(e) *adult*: *acc.* time (*dim*=58). (f) *connect*: *acc.* time (*dim*=58).

Fig. 8. Center reuse across *k* with inc/dec *k* on datasets *adult* and *connect*. *acc.* is short for accumulated.

Table 5
Speedups (X) of center reuse across *k* with different *kstep*.

kstep	10	20	50	100	200	500
Sensorless	4.11(3.96)	3.08	1.88	1.71	1.72	1.13
Miniboone	4.31(4.06)	3.22	2.20	1.68	1.32	1.15
Adult	4.26(3.85)	3.10	2.13	2.06	1.46	1.23
Connect	4.95(4.20)	3.49	2.26	1.93	1.56	1.16

We next answer the questions in detail. We report the detailed measurements with four representatives of all the datasets.

4.5.1. Insights for center reuse across k

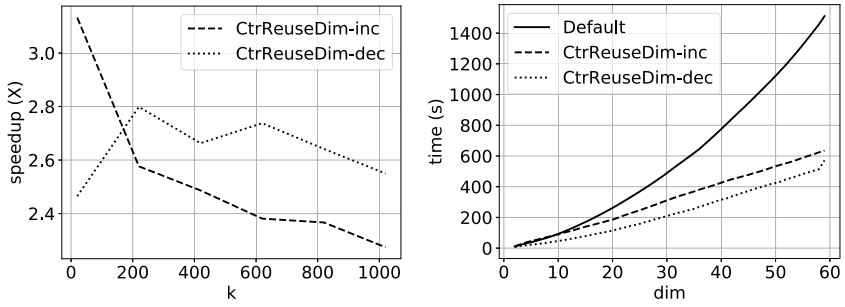
To compare the speedup of reusing centers from a smaller *k* with that from a larger *k*, we compare two methods: CtrReuseK-inc, which uses *k* centers to initialize $k + kstep$ centers by randomly adding *kstep* centers, and CtrReuseK-dec, which uses *k* centers to initialize $k - kstep$ centers by merging centers through one iteration of *k*-means. The range of *k* values is from 20 to 1020 and *kstep* is 200. The range of *d* values is from 2 to 59 and *dstep* is 1. The baseline method is the default method for *k*-means configuration.

Experiment results show that **center reuse by merging centers from a larger *k* generally gives larger speedups than that by randomly adding centers from a smaller *k*.** Figs. 8(a) and 8(b) show the detailed results on datasets *adult* and *connect*: CtrReuseK-dec gives much larger speedups than CtrReuseK-inc especially when the dimensions are larger than eight. It is worth noting that because CtrReuseK-dec starts from the largest *k*, it has a longer startup time. However, its larger speedups on other *k* values lead to much shorter overall configuration time. It is confirmed by the accumulated time of the configuration process as Figs. 8(c)–8(f) show with dimensions 10 and 58.

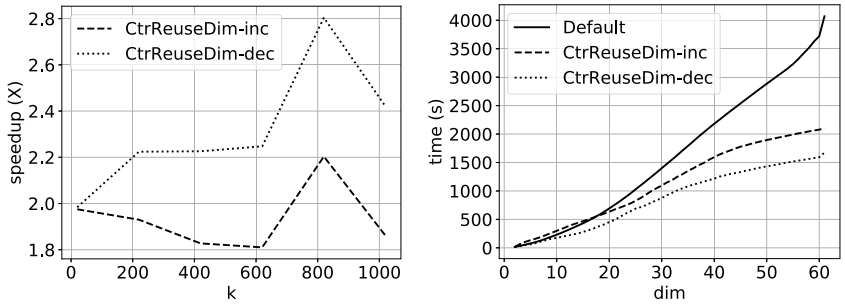
Table 5 shows the speedup of center reuse across *k* with different step size. Because decreasing *k* gives better speedups, we listed only speedups from the method CenterReuseK-dec except for the column '10', where the speedups from CenterReuseK-inc are listed in parentheses. According to the table, **the larger the *kstep*, the smaller the speedup. When the *kstep* is less than 500, which means the change is less than 50% of the maximum *k* value, center reuse gives significant speedups.**

4.5.2. Insights for center reuse across feature sets

We conduct similar experiments to examine the influence of reuse directions across feature sets. We use CtrReuseDim-inc and CtrReuseDim-dec for the increasing and decreasing directions, and *dstep* for the step size. CtrReuseDim-inc fills extra dimensions with the mean value of each dimension, and CtrReuseDim-dec

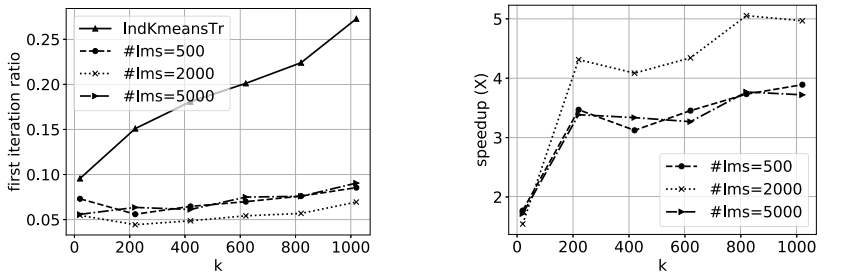


(a) *adult*: overall speedups. (b) *adult*: accumulated time.

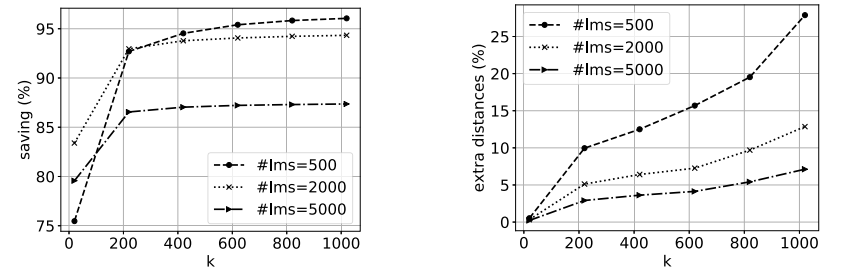


(c) *connect*: overall speedups. (d) *connect*: accumulated time.

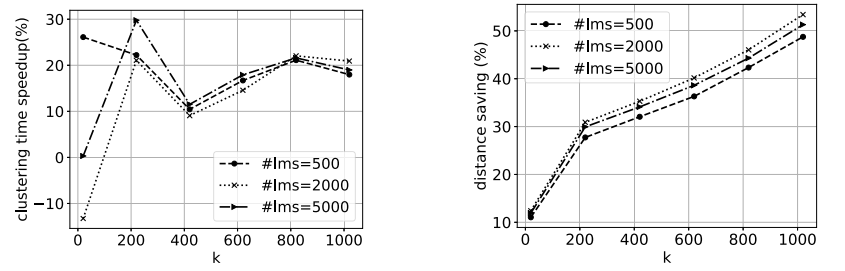
Fig. 9. Center reuse across feature sets with inc/dec d on datasets *adult* and *connect*.



(a) First iteration ratio over k. (b) First iteration speedups.

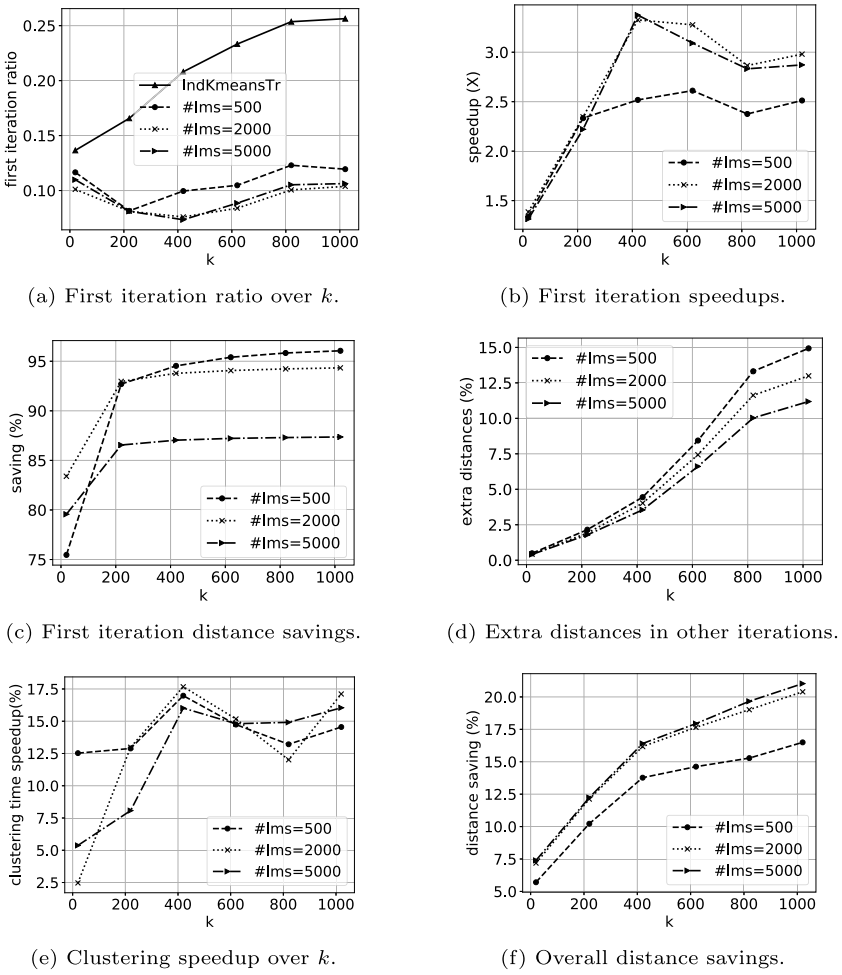


(c) First iteration distance savings. (d) Extra distances in other iterations.



(e) Clustering speedup over k. (f) Overall distance savings.

Fig. 10. Reuse-based filtering performance on different k and different numbers of landmarks (#lms) on *adult* (dim=11).



(a) First iteration ratio over k. (b) First iteration speedups.

(c) First iteration distance savings. (d) Extra distances in other iterations.

(e) Clustering speedup over k. (f) Overall distance savings.

Fig. 11. Reuse-based filtering performance on different k and different numbers of landmarks (#lms) on *adult* (dim=59).

Table 6
Speedups (X) of center reuse across feature sets with different *dstep*.

dstep	1	2	4	8	16	32
Sensorless	1.82(1.26)	1.14	1.03	1.0	–	–
Adult	2.62(2.36)	2.10	1.67	1.31	1.22	1.07
Connect	2.43(1.94)	2.15	2.06	1.27	1.08	1.01
Activity	4.19(3.32)	3.36	2.70	2.20	1.62	1.28

just removes the extra dimensions. The range of k values is from 20 to 1020 and *kstep* is 200. The range of d values is from 2 to 59 and *dstep* is 1. The baseline method is the default method.

Experiment results show that **center reuse by removing dimensions from centers with a larger d generally performs better than center reuse by adding dimensions to centers a smaller k.** Fig. 9 shows the detailed results on datasets *adult* and *connect*.

Table 6 shows the speedup of center reuse across feature sets with different step size. Since decreasing d gives similar or better speedups, we listed only speedups from the method CenterReuseDim-dec except for the column '1', where the speedups from CenterReuseDim-inc are listed in parentheses. According to the table, **the larger the *dstep*, the smaller the speedup. When the *dstep* is less than 50% of the maximum size of the feature set, the speedups from center reuse are significant.**

4.5.3. Insights for reuse-based filtering

This part investigates the influence of the number of landmarks for reuse-based filtering, and the impact of the filtering on later iterations of k-means.

The number of landmarks determines the number of distance pruned through reuse-based filtering. As the number of landmarks is larger, the lower bounds of distance from each point to cluster centers are tighter and thus more exact distance calculations could be pruned; however, the overhead of calculating the lower bounds also becomes significant.

Table 7 shows the speedups and corresponding distance savings for the first iteration of k-means with reuse-based filtering across k. According to the results, **decent speedups manifest when the number of landmarks is around $\sqrt{n}$.** The observation

Table 7
Speedups (X) and distance savings for the first iteration of K-means with reuse-based filtering across k.

#landmarks	200	500	1000	2000	3000	4000	5000
Sensorless	2.77(89.7%)	3.07(92.5%)	**3.20(93.4%)**	3.14(92.9%)	3.06(91.5%)	2.95(90.1%)	2.71(85.3%)
Miniboone	1.26(50.5%)	1.29(54.8%)	1.37(57.6%)	1.39(60.0%)	1.41(61.1%)	1.47(61.8%)	**1.47(62.1%)**
Adult	2.52(74.9%)	2.96(81.6%)	3.47(84.9%)	**3.62(86.1%)**	3.58(85.4%)	3.45(84.0%)	3.29(82.4%)
Activity	**1.05(16.7%)**	1.03(16.7%)	1.01(15.3%)	0.97(12.3%)	0.95(9.9%)	0.94(7.4%)	0.93(5.7%)

is consistent with previous studies on fast knn with triangle inequality [30–32].

As mentioned in Section 3.3, reuse-based filtering helps with the first iteration of k-means, but could possibly degrade the efficiency of the later iterations of k-means. The overall results in Sections 4.2 and 4.3 have already shown that the overall effects are positive, leading to large overall speedups. Figs. 10 and 11 show the detailed results of *adult* on three numbers of landmarks (#*lms*) to shed some insights in further depth when $d = 11$ and $d = 59$. Figs. 10(d) and 11(d) report that the looser bounds due to the use of reuse-based filtering in the first iteration cause about 0%–20% extra distance calculations in later iterations. However, the large savings in the first iteration (reflected by 75%–95% first iteration distance savings in Figs. 10(c) and 11(c), up to 5.1X speedups in Fig. 10(b) and up to 3.4X speedups in Fig. 11(b)) still lead to 10%–60% overall distance savings as Figs. 10(f) and 11(f) reports, and significant clustering speedups as Figs. 10(e) and 11(e) shows. A Similar positive results are observed on other d values.

5. Conclusion

In this work, we introduced the concept of *reuse-centric k-means configurations* to promote information reuse across the explorations of different configurations of k-means. It was shown that our computation-reuse promotion techniques, *reuse-based filtering* and *center reuse*, could largely cut the configuration time of k-means–based data classification. We also introduced a two-phase design, which when working in synergy with the other two techniques, reduces the uniform search-based attainment of classification error surfaces by a factor of 9. In addition, through a series of sensitivity study and in-depth analysis, we provided some important insights on how to tap into the full potential of the techniques.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This material is based upon work supported by DOE Early Career Award, United States (DE-SC0013700), the National Science Foundation, United States (NSF) under Grant No. CCF-1455404, CCF-1525609, CNS-1717425, CCF-1703487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or NSF.

References

[1] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, S.Y. Philip, et al., Top 10 algorithms in data mining, Knowl. Inf. Syst. 14 (1) (2008) 1–37.
[2] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning, Vol. 1, in: Springer Series in Statistics, Springer, Berlin, 2001.
[3] A. Coates, A.Y. Ng, Learning feature representations with k-means, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 561–580.
[4] A. Kalogeratos, A. Likas, Dip-means: an incremental clustering method for estimating the number of clusters, in: Advances in Neural Information Processing Systems, 2012, pp. 2393–2401.
[5] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (Feb) (2012) 281–305.
[6] H.H. Hoos, in: Y. Hamadi, E. Monfroy, F. Saubion (Eds.), Automated Algorithm Configuration and Parameter Tuning, Vol. 3, Springer Berlin Heidelberg, 2012.
[7] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: An overview, in: Experimental Methods for the Analysis of Optimization Algorithms, Springer, 2010, pp. 311–336.
[8] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the F-race algorithm: Sampling design and iterative refinement, in: International Workshop on Hybrid Metaheuristics, Springer, 2007, pp. 108–122.
[9] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, Morgan Kaufmann Publishers Inc., 2002, pp. 11–18.
[10] M. López-Ibánez, L. Paquete, T. Stützle, Exploratory analysis of stochastic local search algorithms in biobjective optimization, in: Experimental Methods for the Analysis of Optimization Algorithms, Springer, 2010, pp. 209–222.
[11] F. Hutter, H.H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: AAAI, Vol. 7, 2007, pp. 1152–1157.
[12] H.H. Hoos, T. Stützle, Stochastic Local Search: Foundations and Applications, Elsevier, 2004.
[13] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.
[14] F. Hutter, H.H. Hoos, K. Leyton-Brown, K.P. Murphy, An experimental investigation of model-based parameter optimisation: SPO and beyond, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, 2009, pp. 271–278.
[15] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[16] I.-C. Yeh, C.-h. Lien, The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients, Expert Syst. Appl. 36 (2) (2009) 2473–2480.
[17] R. Huerta, T. Mosqueiro, J. Fonollosa, N.F. Rulkov, I. Rodriguez-Lujan, Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring, Chemometr. Intell. Lab. Syst. 157 (2016) 169–176.
[18] D. Anguita, A. Ghio, L. Oneto, X. Parra, J.L. Reyes-Ortiz, A public domain dataset for human activity recognition using smartphones, in: ESANN, 2013.
[19] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, T. Mytkowicz, Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup, in: Proceedings of the 32nd International Conference on Machine Learning, ICML-15, 2015, pp. 579–587.
[20] J. Drake, G. Hamerly, Accelerated k-means with adaptive distance bounds, in: 5th NIPS Workshop on Optimization for Machine Learning, 2012, pp. 42–53.
[21] G. Hamerly, Making k-means even faster, in: Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, 2010, pp. 130–140.
[22] C. Elkan, Using the triangle inequality to accelerate k-means, in: ICML, Vol. 3, 2003, pp. 147–153.
[23] Q. Zhao, V. Hautamaki, P. Fränti, Knee point detection in BIC for detecting the number of clusters, in: International Conference on Advanced Concepts for Intelligent Vision Systems, Springer, 2008, pp. 664–673.
[24] V. Satopaa, J. Albrecht, D. Irwin, B. Raghavan, Finding a "kneedle" in a haystack: Detecting knee points in system behavior, in: Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on, IEEE, 2011, pp. 166–171.
[25] A. Asuncion, D. Newman, UCI machine learning repository, 2007.
[26] D. Arthur, S. Vassilvitskii, k-means++: The advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.
[27] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. (2) (1979) 224–227.
[28] J.C. Dunn, Well-separated clusters and optimal fuzzy partitions, J. Cybern. 4 (1) (1974) 95–104.
[29] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65.
[30] X. Wang, A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality, in: Neural Networks (IJCNN), the 2011 International Joint Conference on, IEEE, 2011, pp. 1293–1299.
[31] W. Lu, Y. Shen, S. Chen, B.C. Ooi, Efficient processing of k nearest neighbor joins using mapreduce, Proc. VLDB Endowment 5 (10) (2012) 1016–1027.
[32] Y. Ding, X. Shen, M. Musuvathi, T. Mytkowicz, Top: A framework for enabling algorithmic optimizations for distance-related problems, Proc. VLDB Endowment 8 (10) (2015) 1046–1057.