Neural Networks 63 (2015) 117–132

Contents lists available at ScienceDirect

Neural Networks

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

Approximate kernel competitive learning

Jian-Sheng Wu [a,b], Wei-Shi Zheng [a,c,*], Jian-Huang Lai [a,d]

[a] School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510006, China
[b] SYSU-CMU Shunde International Joint Research Institute, Shunde, China
[c] Guangdong Province Key Laboratory of Computational Science, Guangzhou 510275, China
[d] Guangdong Province Key Laboratory of Information Security, China

HIGHLIGHTS

- Kernel competitive learning (KCL) cannot be applied in large scale data problem.
- Propose a projection based approximate KCL method for large scale data problem.
- Provide theoretical analysis on why the approximation modelling would work for KCL.
- A pseudo-parallelled approximate computation framework for large scale KCL is developed.
- Experimentally show the effectiveness and efficiency of the proposals.

ARTICLE INFO

Article history:
Received 18 April 2014
Received in revised form 23 September 2014
Accepted 14 November 2014
Available online 27 November 2014

Keywords:
Kernel clustering
Kernel competitive learning
Large scale computation

ABSTRACT

Kernel competitive learning has been successfully used to achieve robust clustering. However, kernel competitive learning (KCL) is not scalable for large scale data processing, because (1) it has to calculate and store the full kernel matrix that is too large to be calculated and kept in the memory and (2) it cannot be computed in parallel. In this paper we develop a framework of approximate kernel competitive learning for processing large scale dataset. The proposed framework consists of two parts. First, it derives an approximate kernel competitive learning (AKCL), which learns kernel competitive learning in a subspace via sampling. We provide solid theoretical analysis on why the proposed approximation modelling would work for kernel competitive learning, and furthermore, we show that the computational complexity of AKCL is largely reduced. Second, we propose a pseudo-parallelled approximate kernel competitive learning (PAKCL) based on a set-based kernel competitive learning strategy, which overcomes the obstacle of using parallel programming in kernel competitive learning and significantly accelerates the approximate kernel competitive learning for large scale clustering. The empirical evaluation on publicly available datasets shows that the proposed AKCL and PAKCL can perform comparably as KCL, with a large reduction on computational cost. Also, the proposed methods achieve more effective clustering performance in terms of clustering precision against related approximate clustering approaches.

1. Introduction

Clustering, as an important kind of unsupervised learning approach, plays an important role in discovering the structure of data and exploratory in nature (Jain, 2010). Up to now, there are lots of clustering methods developed for various problems in a wide range of applications, e.g., engineering, computer science, life and medical science, earth science, social science and economics (Xu & Wunsch, 2005). Typical clustering methods are such as k-means (MacQueen, 1967), hierarchical clustering, kernel k-means (Schölkopf, Smola, & Müller, 1998), and spectral clustering (Ng, Jordan, & Weiss, 2001).

Due to the effectiveness on grouping data, in the past decades, competitive learning has received a lot of attention and has been widely applied in data clustering (Banerjee & Ghosh, 2004; Cottrell, Hammer, Hasenfuß, & Villmann, 2006; Fort, Letremy, & Cottrell, 2002; Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Martinetz, Berkovich, & Schulten, 1993; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Schleif, Zhu, & Hammer, 2013; Vesanto & Alhoniemi, 2000; Wang, Lai, & Zhu, 2010, 2012; Xu, Krzyzak, & Oja, 1993). Compared to the traditional iterative clustering algorithms,

* Corresponding author at: School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510006, China. Tel.: +86 020 84110175.
E-mail addresses: jiansheng4211@gmail.com (J.-S. Wu), wszheng@ieee.org (W.-S. Zheng), stsljh@mail.sysu.edu.cn (J.-H. Lai).

http://dx.doi.org/10.1016/j.neunet.2014.11.003
0893-6080/

such as k-means and kernel k-means, competitive learning has the advantages to avoid being trapped in a local minimum resulted by non-optimal initialisations (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Schleif et al., 2013; Wang et al., 2010) and has the ability to avoid learning extremely small clusters or even empty clusters (Banerjee & Ghosh, 2004) due to the adoption of the online update rule and winner update rule.

Most of the developed competitive learning approaches (Ahalt, Krishnamurthy, Chen, & Melton, 1990; Cottrell et al., 2006; Desieno, 1988; Fort et al., 2002; Kohonen, 1990; Martinetz et al., 1993; Vesanto & Alhoniemi, 2000; Xu et al., 1993) are based on the assumption that the clusters can be linearly separated in the data space. To overcome this shortage, recently, kernel competitive learning (KCL) (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Wang et al., 2010) and graph based multi-prototype competitive learning (Wang, Lai et al., 2012) have been developed to handle with the nonlinearly separable datasets. By mapping data points from the data space into a much higher or even infinite dimensional space, called the Reproduced Kernel Hilbert Space (RKHS), induced by a kernel function that is usually implicitly defined, as kernel k-means, kernel competitive learning would be likely to find linearly separated hyperplane in the RKHS space, which can yield arbitrary clustering shapes in the original data space (Xu & Wunsch, 2005). After initialising using a graph-based method, Wang, Lai et al. (2012) performs a multi-prototype competitive learning to refine the clustering and identify clusters of an arbitrary shape.

However, the large scale computational complexity and space complexity challenge the kernel competitive learning and the graph based multi-prototype competitive learning on dealing with large scale datasets. It is because nowadays, there are huge of data (e.g. world wide webpages, digital images and video surveillance data) created every day due to the development of computer science and information techniques. For example, the amount of digital data created and replicated from now to the year 2020 will reach at least 35 trillion gigabytes (Digital Universe Study, 2010).

To address the high computational complexity and space complexity problems, works in Schleif et al. (2013); Schleif, Zhu, Gisbrecht, and Hammer (2012) approximated the kernel competitive learning by avoiding using the full kernel matrix. By directly applying the Nystroḿ method (Williams & Seeger, 2001) to approximate the kernel matrix when calculating the distances between data points and cluster prototypes, Schleif et al. (2012) reduce the computational complexity and space complexity from $O(\tau cn^2)$ and $O(n^2)$ to $O(nm^2 + m^3 + 2\tau cnm)$ and $O(mn)$, respectively, where τ is the number of iterations and m is the number of sampled data points. By using the core set points of each cluster, rather than using the whole data points of each cluster, to update the cluster prototypes after the reassignment of cluster labels, Schleif et al. (2013) can avoid calculating the whole kernel matrix and the quadratic computation when calculating the distances between data points and the cluster prototypes. However, it has to calculate the sub kernel matrix when computing the distances between data points and prototypes and costs $O(nd/\epsilon^2)$ (Bădoiu & Indvk, 2002) (or $O(1/\epsilon^8)$ when applying the probabilistic speedup method Tsang, Kwok, & Cheung, 2005), where ϵ is appropriately set to be 10^{-6} (Tsang et al., 2005), to calculate a core set for each cluster in each iterative step, which prevents its application to high dimensional clustering problem and clustering problem with a large number of clusters. For example, it almost costs 3 days to cluster the Caltech 101 data points if ϵ is set to 0.1 and much more time is used if ϵ is set smaller. Moreover, as stated in Fort et al. (2002), due to the batch update of the cluster prototypes after the reassignment of cluster labels for data points, both of them are depending on the initialisation and will generate imbalanced clusters. Note that imbalanced clustering will likely yield extremely small clusters or empty clusters (Bradley, Bennett, & Demiriz, 2000; Kashima, Ide, Kato, & Sugiyama, 2009). This can be a considerable concern for clustering problems with a large number of clusters and a high dimensionality (Bradley et al., 2000). Although the sizes of clusters may differ differently, more balanced clustering (or say less imbalanced clustering) is preferred in several real-life applications, such as large retail chains, and marketing campaign (Banerjee & Ghosh, 2004). In addition, balanced clustering is helpful to alleviate the sensitivity to clustering initialisation and avoid outlier clusters (Banerjee & Ghosh, 2004) as well.

In the clustering literature, there are also related works on large scale clustering. These works include: (1) k-means algorithm (Huang, 1998; Nistér & Stewénius, 2006; Ordonez & Omiecinski, 2004; Philbin, Chum, Isard, Sivic, & Zisserman, 2007; Wang, Wang, Ke, Zeng, & Li, 2012), (2) data sampling and summarisation techniques (Guha, Rastogi, & Shim, 1998; Kaufman & Rousseeuw, 2005; Ng & Han, 2002; Zhang, Ramakrishnan, & Livny, 1996), (3) distributed models (Chen, Song, Bai, Lin, & Chang, 2011; Cordeiro et al., 2011; Ene, Im, & Moseley, 2011) and incremental clusterings (Lühr & Lazarescu, 2009; Zhang, Liu, & Wang, 2007), (4) random sampling for approximating some specific non-linear kernels (Chitta, Jin, & Jain, 2012; Kar & Karnick, 2012; Pham & Pagh, 2013; Rahimi & Recht, 2007), and (5) data sampling for kernel k-means (Chitta, Jin, Havens, & Jain, 2011) and spectral clustering (Fowlkes, Belongie, Chung, & Malik, 2004; Williams & Seeger, 2001). However, these works have their weaknesses in one of the following points: (1) relying on the assumption about linearly separable model (Cordeiro et al., 2011; Ene et al., 2011; Guha et al., 1998; Huang, 1998; Kaufman & Rousseeuw, 2005; Ng & Han, 2002; Nistér & Stewénius, 2006; Ordonez & Omiecinski, 2004; Philbin et al., 2007; Wang, Wang et al., 2012; Zhang et al., 1996), (2) not scalable for nonlinear extension (Nistér & Stewénius, 2006; Philbin et al., 2007; Wang, Wang et al., 2012), and (3) incurring several limitations, such as sensitive to prototype initialisation, trapped in local minimum and yielding imbalanced partition (Chitta et al., 2011, 2012; Kar & Karnick, 2012; Pham & Pagh, 2013; Rahimi & Recht, 2007). Although kernel competitive learning can help alleviate the above limitations (1) and (3), it is indeed necessary to develop a scalable approach for kernel competitive learning for processing large scale data.

In this work, in order to make kernel competitive learning tractable for large scale data, we propose an approximate kernel competitive learning in this paper for pursuing robust approximate kernel competitive learning for clustering large scale data in a nonlinear way. In order to accelerate the large scale learning, we further propose a pseudo-parallelled approximate kernel competitive learning framework based on a set-based kernel competitive learning strategy.

1.1. Contributions

Motivated by the approximation idea used in approximate kernel k-means that endows low time and space complexity for clustering, in this paper, we wish to combine kernel competitive learning and the approximation idea together so as to develop a large scale clustering method. However, it does not mean it is straightforward to apply the approximation idea used in approximate kernel k-means to kernel competitive learning, because we will analyse that a direct use would contradict the constraint used in the approximation idea that the prototypes must be bounded in a sampled subspace, which is necessary to avoid the calculation and storing of the full kernel matrix. In order to solve this problem, we introduce the projection strategy into the approximation framework and combine it with kernel competitive learning so as to develop a novel large scale kernel competitive learning

method. We prove that the introduced projection step makes the approximation strategy work correctly along with kernel competitive learning.

Our second contribution is to accelerate the proposed approximate kernel competitive learning. We develop a pseudo-parallelled approximate computation framework based on a set-based update strategy. Different from the batch-based update kernel competitive learning (Cottrell et al., 2006; Fort et al., 2002; Schleif et al., 2013, 2012), which parallelises the competitive learning by updating the cluster prototypes after assigning all data points to their closest clusters, the set-based update strategy is to update the prototypes based on a subset of data points, so that parallelled computing can be derived in each subset to speed up the clustering, meanwhile retaining the co-interaction between update of prototypes over the subsets.

Although it is also an intuitive approach to directly apply the Nystrom̈ method to approximate the kernel matrix in the kernel competitive learning approaches (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Wang et al., 2010) as in Schleif et al. (2012). However, it cannot solve the large scale computational complexity because the update for the winner cluster prototype involving $O(n)$ computations for each coming data point in each iteration, so that the total complexity for updating the prototypes still be $O(\tau n^2)$. While it is possible to apply other kernel approximation methods in Chitta et al. (2012), Kar and Karnick (2012), Pham and Pagh (2013) and Rahimi and Recht (2007), these approximation works are kernel dependent and cannot be generalised to other kernels.

In summary, the contributions of the proposals are

(1) An approximate kernel competitive learning method is proposed for making kernel competitive learning applicable to large scale data;
(2) A pseudo-parallelled approximate computation framework for large scale kernel competitive learning method based on a set-based learning strategy is developed.

The experimental results on five real-world datasets from medium scale to large scale demonstrate the effectiveness of the proposed methods as compared to related approaches. We show that our approximate kernel competitive learning would not sacrifice the performance of kernel competitive learning (KCL), while the computational complexity is significantly reduced.

2. Preliminary

To make our paper self-contained, we first introduce a recently developed kernel competitive learning algorithm (Wang et al., 2010) and the approximate kernel k-means for large scale problem.

2.1. Kernel competitive learning

Kernel competitive learning (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Wang et al., 2010) is a generalisation of the linear competitive learning algorithm (Ahalt et al., 1990; Kohonen, 1990; Martinetz et al., 1993). Through mapping data points via a nonlinear mapping induced by a given kernel function into the feature space, kernel competitive learning performs linear competitive learning in the feature space. It is able to group data into arbitrary geometrical shapes in the data space.

To be detailed, let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be a dataset consisting of n data points, where $\mathbf{x}_i \in \Re^d$. Given the dataset $\mathcal{X}$, kernel competitive learning maps these data points into a feature space induced by a given kernel function $\phi(\cdot)$ and spans the corresponding feature space $\mathcal{H}$ using the image set $\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$. In general, the kernel function $\phi(\cdot)$ is implicitly defined by a mercer kernel κ (Schölkopf et al., 1998) such that $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$, where $\langle\cdot,\cdot\rangle$ denotes the inner product between two vectors. Then kernel competitive learning assigns these data points into one of the c clusters $\{\pi_1, \pi_2, \ldots, \pi_c\}$ where

$$\begin{aligned} &\pi_i \neq \emptyset, \quad i = 1, \ldots, c, \\ &\pi_i \cap \pi_j = \emptyset, \quad i, j = 1, \ldots, c,\ i \neq j, \\ &\bigcup_{i=1,\ldots,c} \pi_i = \mathcal{X}. \end{aligned} \tag{1}$$

We briefly describe the kernel competitive learning (Wang et al., 2010) as follows. Let n_k and f_k denote the cumulative winning times and winning frequency of the cluster π_k, respectively. For performing kernel competitive learning, the algorithm randomly picks a data point $\mathbf{x}_i$ in the t-th iteration and then performs the following three steps,

1. **STEP 1**: Select the closest prototype $\mathbf{m}_k$ as the winner prototype by

$$k_i = \arg\min_{k=1,\ldots,c} f_k \|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2, \tag{2}$$

   where $\|\cdot\|$ represents the L_2 norm of the vector.
2. **STEP 2**: Update the winner prototype $\mathbf{m}_{k_i}$ by

$$\mathbf{m}_{k_i} \leftarrow \mathbf{m}_{k_i} + \eta_t(\phi(\mathbf{x}_i) - \mathbf{m}_{k_i}), \tag{3}$$

   where η_t is the learning rate and should decrease monotonically.
3. **STEP 3**: Update the winning frequency by

$$\begin{aligned} &n_{k_i} \leftarrow n_{k_i} + 1, \\ &f_k = n_k / \sum_{l=1}^{c} n_l, \quad k = 1, \ldots, c. \end{aligned} \tag{4}$$

The procedure repeats the above steps until the number of iterations reaches a pre-specified number $t_{\max}$ or all the prototypes converge, which can be characterised by the convergence criterion

$$e = \sum_{k=1}^{c} \|\mathbf{m}_k^{(t)} - \mathbf{m}_k^{(t-1)}\|^2 \leq \epsilon, \tag{5}$$

where $\mathbf{m}_k^{(t)}$ is the k-th prototype in the iteration t.

Like kernel k-means (Schölkopf et al., 1998), kernel competitive learning also suffers from large scale computational complexity and space complexity problems. Its computational complexity and space complexity are $O(\tau c n^2)$ and $O(n^2)$, respectively.

2.2. Approximate kernel k-means

Recently, an approximate kernel k-means was proposed (Chitta et al., 2011) to overcome the large scale computational complexity and space complexity problems in kernel k-means. It constrains the cluster prototypes in a sampled subspace such that the number of vectors used to represent the cluster prototypes decreases greatly. It first randomly samples a subset of m ($m \ll n$) data points from $\mathcal{X}$. Let such a subset be denoted as $\hat{\mathcal{X}} = \{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_m\}$. Then the subspace $\mathcal{H}_b$ is constructed as $\mathcal{H}_b = \text{span}\{\phi(\hat{\mathbf{x}}_1), \ldots, \phi(\hat{\mathbf{x}}_m)\}$. Given such a subspace $\mathcal{H}_b$, approximate kernel k-means (Chitta et al., 2011) groups the dataset $\mathcal{X}$ into c clusters by bounding the cluster prototypes in $\mathcal{H}_b$ as

$$\min_{\{\mathbf{m}_k \in \mathcal{H}_b\}_{k=1}^{c}} \sum_{k=1}^{c} \sum_{i=1}^{n} U_{i,k} \|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2, \tag{6}$$

where $U_{i,k}$ is the (i, k)-entry of the cluster membership indicator matrix U defined by

$$U_{i,k} = \begin{cases} 1 : \text{if } \mathbf{x}_i \in \pi_k, \\ 0 : \text{otherwise.} \end{cases} \tag{7}$$

When the cluster membership matrix U is given, the optimal cluster prototypes learned by Eq. (6) are given by

$$\mathbf{m}_k = \sum_{i=1}^{m} \alpha_{i,k} \phi(\hat{\mathbf{x}}_i), \quad k = 1, \ldots, c, \tag{8}$$

where $\alpha_{i,k}$ is the (i, k)-entry of the coefficient matrix α, defined as

$$\alpha = \hat{K}^{-1} K_B^T \bar{U}. \tag{9}$$

Here, $\bar{U}$ denotes the L_1 normalised membership matrix whose entry $\bar{U}_{i,k}$ is defined as $\bar{U}_{i,k} = U_{i,k}/|\pi_k|$, K_B is the similarity matrix whose (i, j)-entry is defined as $(K_B)_{i,j} = \kappa(\mathbf{x}_i, \hat{\mathbf{x}}_j)$, and $\hat{K}$ is a block of K_B with the entry $\hat{K}_{i,j} = \kappa(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$. As a result, the distance from the image $\phi(\mathbf{x}_i)$ to the cluster prototype $\mathbf{m}_k$ can be computed efficiently by

$$K_{i,i} - 2\sum_{j=1}^{m} \alpha_{j,k} (K_B)_{i,j} + \sum_{j=1}^{m}\sum_{l=1}^{m} \alpha_{j,k}\alpha_{l,k} \hat{K}_{j,l}. \tag{10}$$

Hence, the time and the space complexities are $O(nm^2 + m^3 + \tau cnm)$ and $O(nm)$, respectively. Therefore, by constraining the cluster prototypes in a sampled subspace, the approximate kernel k-means algorithm extends the kernel k-means to solve the large scale clustering problem with much lower computational and space complexity.

3. Proposed kernel competitive learning for large scale learning

In this section, we propose our modelling for approximate kernel competitive learning based on the typical competitive learning framework, in which the winner cluster prototype is updated by the on-line update rule, i.e., the winner cluster prototype is updated immediately after the coming data point is assigned to its closest cluster. Section 3.1 explains why directly applying competitive learning to approximate kernel k-means does not work. Sections 3.2 and 3.3 present the framework and theoretical analysis first, Section 3.4 presents a detailed computation procedure for the theoretical analysis in Section 3.3 and the computational complexity is also provided in Section 3.5 finally.

3.1. Can competitive learning be applied to approximate kernel k-means?

An intuitive approach to extend kernel competitive learning for large scale learning may be to apply competitive learning to approximate kernel k-means. However, it can be verified that the prototype will be out of the subspace $\mathcal{H}_b$ when $\phi(\mathbf{x}_i)$ is not in $\mathcal{H}_b$, due to its update for the winner prototype using Eq. (3) (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Wang et al., 2010). So it will contradict to the constraint in approximate kernel k-means that the prototypes must be bounded in the spanned subspace $\mathcal{H}_b$. To illustrate, suppose $\mathbf{x}_i$ is the current data point and $\phi(\mathbf{x}_i)$ is not in $\mathcal{H}_b$. As shown in Fig. 1, if we apply Eq. (3) to update the winner prototype $\mathbf{m}_{k_i}$ and $\mathbf{z}'$ is the updated winner prototype, then $\mathbf{z}'$ will be out of $\mathcal{H}_b$, so $\mathbf{z}'$ cannot be represented by the sampled data points. As a result, it has to calculate and keep the full kernel matrix to compute the similarities between images and clusters. Consequently, its computational complexity will increase as the same with kernel k-means, which is $O(\tau cn^2)$. Thus, using Eq. (3) to update the winner prototype will not work in approximate kernel k-means. In the following sections, we will propose two scalable approximate kernel competitive learning methods.

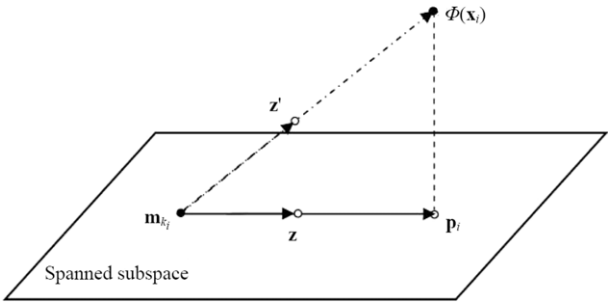


Fig. 1. Illustration of the direction on which the winner prototype moves. The dash dot arrow line is the winner prototype update direction using the update strategy in KCL and the solid arrow line is the update direction using our projection strategy.

3.2. The model

We aim to introduce the approximation idea into kernel competitive learning for tackling large scale data clustering in a robust way. Let $\hat{\Phi} = [\phi(\hat{\mathbf{x}}_1), \ldots, \phi(\hat{\mathbf{x}}_m)]$ be the matrix consisting of $\{\phi(\hat{\mathbf{x}}_i)\}_{i=1}^m$ as the columns and $(K_B)_{i,:}$ be the ith row of the matrix K_B. Then the approximate kernel competitive learning model consists of the following four steps:

1. **STEP 0**: Compute the projection vector of $\phi(\mathbf{x}_i)$ in $\mathcal{H}_b$:
$$\mathbf{p}_i = \hat{\Phi}\hat{K}^{-1}((K_B)_{i,:})^T. \tag{11}$$
2. **STEP 1**: Select the closest prototype as the winner prototype by
$$k_i = \arg\min_{k=1,\ldots,c} f_k \|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2. \tag{12}$$
3. **STEP 2**: Update the winner prototype $\mathbf{m}_{k_i}$ by
$$\mathbf{m}_{k_i} \leftarrow \mathbf{m}_{k_i} + \eta_t(\mathbf{p}_i - \mathbf{m}_{k_i}), \tag{13}$$
4. **STEP 3**: Update the winning frequency by
$$n_{k_i} \leftarrow n_{k_i} + 1,$$
$$f_k = n_k / \sum_{l=1}^{c} n_l, \quad k = 1, \ldots, c. \tag{14}$$

Steps 1–3 will repeat until the stopping criterion is reached.

Although compared to the computational framework of kernel competitive learning (KCL) we only add the STEP 0 and modify STEP 2 in order to introduce the projection strategy step into KCL, such a change is significantly important to make the approximation idea used in KCL and make it applied to much larger scale dataset as we show in the experiment. Although the changes are not too much, the theoretical guarantee behind is not trivial. In the following, we will theoretically analyse the correctness of this extension.

3.3. Theoretical analysis

A main characteristic of the developed approximate kernel competitive learning (Section 3.2) is to compute the projection of $\phi(\mathbf{x}_i)$ at Step 0 and use it to update the winner prototype $\mathbf{m}_{k_i}$ at Step 2. Albeit not a big change of kernel competitive learning, we prove that such a modification is significantly necessary and important to make the approximation idea work along with kernel competitive learning by the following theorems.

Theorem 1. *The optimal cluster prototypes in Eq. (6) are actually in the projection subspace. That is, the k-th cluster prototype can be represented using the mean of the projections of $\phi(\mathbf{x}_i)$, where $\mathbf{x}_i \in \pi_k$, i.e.*

$$\mathbf{m}_k = \frac{\sum_{i=1}^{n} U_{i,k}\mathbf{p}_i}{|\pi_k|}, \quad k = 1, \ldots, c, \tag{15}$$

where $\mathbf{p}_i$ is the projection of $\phi(\mathbf{x}_i)$ in $\mathcal{H}_b$ and it is defined by Eq. (11).

To prove this theorem, we introduce the following lemma first.

Lemma 1. *The ith column of the matrix $\mathbf{P} = \hat{\Phi}\hat{K}^{-1}K_B^T$ equals the projection vector of $\phi(\mathbf{x}_i)$ in $\mathcal{H}_b$.*

Proof. Denote $\phi(\mathbf{x}_i)$ by $\mathbf{b}_i$ and suppose $\bar{\mathbf{b}}_i$ is the orthogonal projection vector of $\mathbf{b}_i$ in $\mathcal{H}_b$. Then $\mathbf{b}_i - \bar{\mathbf{b}}_i$ is perpendicular to $\mathcal{H}_b$, so that in the orthogonal complement of $\mathcal{H}_b$ and the system of matrix equation

$$\hat{\Phi}\mathbf{y} = \bar{\mathbf{b}}_i \tag{16}$$

is consistent, and has at least one solution. Suppose $\bar{\mathbf{y}}$ is one of the solutions to this system. Then we have

$$\hat{\Phi}^T(\mathbf{b}_i - \bar{\mathbf{b}}_i) = \hat{\Phi}^T(\mathbf{b}_i - \hat{\Phi}\bar{\mathbf{y}}) = 0 \tag{17}$$

according to the theorem in Lay (2002). Hence,

$$\hat{\Phi}^T\hat{\Phi}\bar{\mathbf{y}} = \hat{\Phi}^T\phi(\mathbf{x}_i). \tag{18}$$

Then, the projection vector of $\phi(\mathbf{x}_i)$ in $\mathcal{H}_b$ is given by

$$\bar{\mathbf{b}}_i = \hat{\Phi}\bar{\mathbf{y}} = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\phi(\mathbf{x}_i) = \hat{\Phi}\hat{K}^{-1}((K_B)_{i,:})^T, \tag{19}$$

according to Eq. (16) and $\hat{\Phi}^T\hat{\Phi} = \hat{K}$. It is easy to verify that $\bar{\mathbf{b}}_i$ is the ith column of the matrix $\hat{\Phi}\hat{K}^{-1}K_B^T$. □

Based on the above Lemma, we can now prove Theorem 1 as follows:

Proof. Given the cluster membership matrix U, the optimal prototypes in Eq. (6) are given by Eq. (8). We can rewrite the k-th prototype in the matrix–vector product form as

$$\mathbf{m}_k = \hat{\Phi}\alpha_{:,k}, \tag{20}$$

where $\alpha_{:,k}$ is the k-th column of the matrix α. Since $\alpha = \hat{K}^{-1}K_B^T\bar{U}$, hence, $\alpha_{:,k} = \hat{K}^{-1}K_B^T\bar{\mathbf{u}}_k$, where $\bar{\mathbf{u}}_k$ is the k-th column vector of $\bar{U}$. Therefore, we have

$$\mathbf{m}_k = \hat{\Phi}\hat{K}^{-1}K_B^T\bar{\mathbf{u}}_k = \mathbf{P}\bar{\mathbf{u}}_k = \frac{\sum_{i=1}^{n} U_{i,k}\mathbf{p}_i}{|\pi_k|}. \tag{21}$$

Thus, we can express the cluster prototypes by using the projection vectors. □

Theorem 2. *Suppose $\mathbf{m}_{k_i}$ is the winner cluster prototype and $\mathbf{m}_{k_i} \in \mathcal{H}_b$, then the winner prototype $\mathbf{z}$ updated using Eq. (13) is the projection of $\mathbf{z}'$ updated using Eq. (3) onto $\mathcal{H}_b$, i.e.,*

$$\mathbf{z} = \Pi\mathbf{z}', \tag{22}$$

where $\Pi = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T$ is the projection matrix.

Proof. Suppose $\mathbf{x}_i$ is the picked data point and $\mathbf{m}_{k_i}$ is the winner cluster prototype. Denote $\bar{\mathbf{z}}$ as the projection of $\mathbf{z}'$ onto $\mathcal{H}_b$. Then, by projecting $\mathbf{z}'$ onto $\mathcal{H}_b$ we will have

$$\bar{\mathbf{z}} = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\mathbf{z}'. \tag{23}$$

Substitute Eq. (3) for $\mathbf{z}'$ as $\mathbf{z}' = \mathbf{m}_{k_i} + \eta_t(\phi(\mathbf{x}_i) - \mathbf{m}_{k_i})$, then we have

$$\bar{\mathbf{z}} = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T((1-\eta_t)\mathbf{m}_{k_i} + \eta_t\phi(\mathbf{x}_i)). \tag{24}$$

As $\mathbf{m}_{k_i}$ is in $\mathcal{H}_b$, then it can be represented as the linear combination of $\phi(\hat{\mathbf{x}}_1), \ldots, \phi(\hat{\mathbf{x}}_m)$. Suppose $\mathbf{a}$ is the corresponding coefficient vector, i.e., $\mathbf{m}_{k_i} = \hat{\Phi}\mathbf{a}$, then

$$\begin{aligned}\bar{\mathbf{z}} &= (1-\eta_t)\hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\hat{\Phi}\mathbf{a} + \eta_t\hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\phi(\mathbf{x}_i)\\ &= \mathbf{m}_{k_i} + \eta_t(\mathbf{p}_i - \mathbf{m}_{k_i}).\end{aligned} \tag{25}$$

The second equation is derived because $\hat{\Phi} = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\hat{\Phi}$ and $\mathbf{p}_i = \hat{\Phi}\hat{K}^{-1}\hat{\Phi}^T\phi(\mathbf{x}_i)$. Therefore, $\mathbf{z}$ is the projection of $\mathbf{z}'$ onto $\mathcal{H}_b$. □

Therefore, according to Theorems 1 and 2, it is feasible to update $\mathbf{m}_{k_i}$ by moving it towards $\mathbf{p}_i$ along $\mathbf{p}_i - \mathbf{m}_{k_i}$ in the way of

$$\mathbf{m}_{k_i} \leftarrow \mathbf{z} = \mathbf{m}_{k_i} + \eta_t(\mathbf{p}_i - \mathbf{m}_{k_i}), \tag{26}$$

instead of moving the winner prototype $\mathbf{m}_{k_i}$ towards $\phi(\mathbf{x}_i)$ along $\phi(\mathbf{x}_i) - \mathbf{m}_{k_i}$ in Eq. (3). As shown in Fig. 1, $\mathbf{m}_{k_i}$ will be still in $\mathcal{H}_b$.

3.4. Model computation

We now detail the computation procedure of the model described in 3.3. For description, we need some notations defined in the following.

Definition 1. A prototype coefficient matrix is denoted as $\gamma \in \Re^{m\times c}$ such that

$$\mathbf{m}_k = \hat{\Phi}\gamma_{:,k}, \quad k = 1, \ldots, c, \tag{27}$$

where $\gamma_{:,k}$ is the k-th column of γ.

Definition 2. Let $\theta \in \Re^{m\times n}$ be a matrix with its ith column $\theta_{:,i}$ equal to $\hat{K}^{-1}(K_B)^T_{i,:}$, $\rho \in \Re^n$ be a column vector with its ith entry $\rho_i = (K_B)_{i,:}\theta_{:,i}$ and $\nu \in \Re^c$ be a column vector with its k-th entry ν_k being $\mathbf{m}_k^T\mathbf{m}_k$.

Based on these definitions, we detail the computations of the model below.

3.4.1. Winner selection

Theorem 3. *The winner selection rule (Eq. (12)) can be realised by*

$$k_i = \arg\min_{k=1,\ldots,c} f_k(K_{i,i} - 2(K_B)_{i,:}\gamma_{:,k} + \nu_k). \tag{28}$$

Proof. First of all, $\phi(\mathbf{x}_i)^T\mathbf{m}_k$ can be written in the equivalent form

$$\phi(\mathbf{x}_i)^T\mathbf{m}_k = \phi(\mathbf{x}_i)^T\hat{\Phi}\gamma_{:,k} = (K_B)_{i,:}\gamma_{:,k}, \tag{29}$$

according to Eq. (27).

Then we expand the squared Euclidean distance $\|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2$ as follows:

$$\|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2 = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T\mathbf{m}_k + \mathbf{m}_k^T\mathbf{m}_k. \tag{30}$$

By substituting $K_{i,i}$, ν_k and Eq. (29) for $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_i)$, $\mathbf{m}_k^T\mathbf{m}_k$ and $\phi(\mathbf{x}_i)^T\mathbf{m}_k$, respectively, we can get the required winner selection rule. □

3.4.2. Winner update

Theorem 4. *The winner update rule (Eq. (13)) can be realised by*

$$\nu_{k_i} \leftarrow (1-\eta_t)^2\nu_{k_i} + \eta_t^2\rho_i + 2\eta_t(1-\eta_t)(K_B)_{i,:}\gamma_{:,k_i}, \tag{31}$$

$$\gamma_{:,k_i} \leftarrow (1-\eta_t)\gamma_{:,k_i} + \eta_t\theta_{:,i}. \tag{32}$$

Algorithm 1 Approximate Kernel Competitive Learning

Input: Matrices K_B and $\hat{K}$, the number of clusters c, the convergence threshold ϵ and the maximum number of iterations t_{max}.
Output: The cluster sets π_k, $k = 1, 2, \ldots, c$.
Initialisation:
Randomly assign labels to all data points, initialise $\bar{U}$ and set $t = 0$.
Compute $\theta = [\hat{K}^{-1}K_B^T]_{m\times n}$, $\gamma = [\theta\bar{U}]_{m\times c}$, $\rho = [diag(K_B\theta)]_{n\times 1}$ and $\nu = [diag(\gamma^T\hat{K}\gamma)]_{c\times 1}$.
repeat
  Randomly generate an index sequence $\mathcal{S} = \{s_i | i = 1, \ldots, n \wedge s_i \in [1, n] \wedge (s_i \neq s_j, \forall i \neq j)\}$
  t = t + 1;
  for $j = 1, 2, \ldots, n$ do
    1) Select the winner $\mathbf{m}_{k_i}$ for data point $\mathbf{x}_i$ ($i = s_j$) via Eq. (28).
    2) Update $\mathbf{m}_{k_i}$ using Eq. (31) and Eq. (32).
    3) Update the frequency using Eq. (14).
  end
  Compute the convergence criterion e using Eq. (36).
until $e < \epsilon || t > t_{max}$

Proof. For the winner cluster prototype $\mathbf{m}_{k_i}$, denote $\hat{\mathbf{m}}_{k_i}$ as the updated k_i-th prototype along with the updated coefficient vector $\hat{\gamma}_{:,k_i}$. It is not hard to verify that $\mathbf{p}_i^T\mathbf{m}_{k_i} = (K_B)_{i,:}\gamma_{:,k_i}$ and $\mathbf{p}_i^T\mathbf{p}_i = \rho_i$.

So, we can update $\nu_{k_i} = \hat{\mathbf{m}}_{k_i}^T\hat{\mathbf{m}}_{k_i}$ by substituting Eq. (13) for $\hat{\mathbf{m}}_{k_i}$, i.e.

$$\begin{aligned}\nu_{k_i} &\leftarrow \left((1-\eta_t)\mathbf{m}_{k_i} + \eta_t\mathbf{p}_i\right)^T\left((1-\eta_t)\mathbf{m}_{k_i} + \eta_t\mathbf{p}_i\right)\\ &= (1-\eta_t)^2\nu_{k_i} + \eta_t^2\rho_i + 2\eta_t(1-\eta_t)(K_B)_{i,:}\gamma_{:,k_i}. \end{aligned} \quad (33)$$

Then, one can update the k_i-th cluster prototype $\mathbf{m}_{k_i}$ by Eq. (13) as

$$\mathbf{m}_{k_i} \leftarrow \hat{\mathbf{m}}_{k_i} = \hat{\Phi}\left((1-\eta_t)\gamma_{:,k_i} + \eta_t\theta_{:,i}\right), \quad (34)$$

and therefore, the coefficient vector $\gamma_{:,k_i}$ can be set as

$$\gamma_{:,k_i} \leftarrow \hat{\gamma}_{:,k_i} = (1-\eta_t)\gamma_{:,k_i} + \eta_t\theta_{:,i}. \quad \square \quad (35)$$

3.4.3. Convergence criterion

Theorem 5. *The convergence criterion e (Eq. (5)) can be computed by means of*

$$e = \sum_{k=1}^{c}\left(\nu_k^{(t)} + \nu_k^{(t-1)} - 2(\gamma_{:,k}^{(t)})^T\hat{K}\gamma_{:,k}^{(t-1)}\right), \quad (36)$$

where $\nu_k^{(t)}$ and $\gamma_{:,k}^{(t)}$ are the values of ν_k and $\gamma_{:,k}$, respectively, in the t-th iteration.

Proof. The theorem can be proved by expanding $\|\mathbf{m}_k^{(t)} - \mathbf{m}_k^{(t-1)}\|^2$ in Eq. (5) as

$$\begin{aligned}\|\mathbf{m}_k^{(t)} - \mathbf{m}_k^{(t-1)}\|^2 &= (\mathbf{m}_k^{(t)})^T\mathbf{m}_k^{(t)} + (\mathbf{m}_k^{(t-1)})^T\mathbf{m}_k^{(t-1)}\\ &\quad - 2(\mathbf{m}_k^{(t)})^T\mathbf{m}_k^{(t-1)}\\ &= \nu_k^{(t)} + \nu_k^{(t-1)} - 2(\gamma_{:,k}^{(t)})^T\hat{K}\gamma_{:,k}^{(t-1)}. \quad \square \end{aligned} \quad (37)$$

Finally, the algorithm is shown in Algorithm 1.

3.5. Computational complexity

In our algorithm, since $m \ll n$ and $c \ll n$, the space complexity of the matrix K_B is $O(nm)$. When $m \ll n$, it is less than $O(n^2)$ remarkably, so that it can be used for processing large scale dataset.

The computation of the proposed method consists of two parts: the matrix initialisation and the iterations used to update the winners. For matrix initialisation, computing the matrix θ, which is the dominant term in the initialisation, takes $O(nm^2 + m^3)$ time, as computing the inverse $\hat{K}^{-1}$ costs $O(m^3)$ computations and the matrix multiplication $\hat{K}^{-1}K_B$ costs $O(nm^2)$. In the update steps, for each data point, it takes $O(m)$ time to calculate the distance from each prototype, involving the computation $(K_B)_{i,:}\gamma_{:,k}$, in one iteration. Hence, selecting the winner prototype for each data point costs $O(cm)$ time. To update the L_2 norm ν of the winner prototype and the prototype coefficient matrix γ, it spends $O(1)$ and $O(m)$ time, respectively. So the total time used to update the prototypes is $O(\tau cnm)$. The time used to compute the convergence criterion is $O(\tau cm^2)$. Therefore, if excluding the initialisation complexity, the overall optimisation complexity of the proposed algorithm is $O(\tau cnm)$.

4. Pseudo-parallelled approximate kernel competitive learning

Multi-cores programming or parallelised computing has been a powerful means to accelerate the computation of machine learning models. However, due to the competition update rule adopted by competitive learning, the proposed approximate kernel competitive learning in the last section cannot be executed in parallel. In this section, we further modify our proposed model and propose a set-based update strategy to develop a pseudo-parallelled approximate kernel competitive learning.

4.1. Model and computation

As shown in Inokuchi and Miyamoto (2006), MacDonald and Fyfe (2000), Mizutani and Miyamoto (2005), Qin and Suganthan (2004) and Wang et al. (2010), kernel competitive learning updates the winner of the competition for each data $\phi(\mathbf{x}_i)$; that is for each input data point $\phi(\mathbf{x}_i)$, a prototype must be selected for update before processing the next input data point $\phi(\mathbf{x}_j)$. This actually prevents parallelled processing of all the data points by making use of multiple CPU cores. In contrast, in the original kernel k-means, the selection of the nearest prototype for each input data is independent and therefore parallelled processing can be used. However, the independent selection of prototypes in kernel k-means ignores the co-interaction between the update of prototypes and therefore may lead to the limitations/weaknesses as mentioned in the introduction.

In this section, we would like to make a compromise between kernel competitive learning and kernel k-means. The main idea is to divide the dataset into subsets, perform the independent selection of prototypes during the processing of each subset and update the prototypes subset by subset. So, we can retain the co-interaction between the update of prototypes at the subset level and perform parallelled processing in each subset. We derive the new model based on the approximate kernel competitive learning proposed in the last section and we call it the pseudo-parallelled approximate kernel competitive learning.

To be detailed, suppose the data points are reordered in the random sequence $\{s_1, s_2, \ldots, s_n\}$. We partition the dataset into R subsets in each iteration, where the subsets are denoted as $B_1, \ldots, B_R$ and br is the size of B_r, $r = 1, \ldots, R$. For convenience of model description, without loss of generality, we assume that the dataset can be divided into R subsets of the same size and bs is the subset size. Let the r-th subset B_r contain the data points $\{\mathbf{x}_{s_{(r-1)*bs+i}}\}_{i=1}^{bs}$.

For subset B_r, we update the prototypes in the batch model as follows. Firstly, we compute the frequency of each cluster (i.e. f_k)

(a) Caltech 101.

(b) Caltech 256.

(c) Imagenet.

Fig. 2. Examples of images from Caltech 101 (a), Caltech 256 (b) and Imagenet (c). See text for details.

that has been selected as a winner historically by Eq. (39). Secondly, we assign labels to the data points in this subset by a weighted minimisation criterion based on the frequencies f_k as follows:

$$\min_{U_{i,k}^{(r)}} \sum_{k=1}^{c} \sum_{i=1}^{bs} U_{i,k}^{(r)} f_k \|\phi(\mathbf{x}_{s_{(r-1)*bs+i}}) - \mathbf{m}_k\|^2, \tag{38}$$

where $U_{i,k}^{(r)}$ is the (i,k)-entry of the cluster membership indicator matrix $U^{(r)} = [\mathbf{u}_1^{(r)}, \ldots, \mathbf{u}_c^{(r)}]$ for data points in B_r. It can be minimised by assigning labels to data points in this subset by using Eq. (28). This step can be completed using multiple CPU cores in parallel. Then, the model updates the frequency for each cluster that is selected as the winner by

$$n_k \leftarrow n_k + |\pi_k^{(r)}|$$
$$f_k = n_k \Big/ \sum_{l=1}^{c} n_l, k = 1, \ldots, c, \tag{39}$$

where $\pi_k^{(r)}$ is the k-th cluster set and $|\pi_k^{(r)}|$ is the number of points assigned to the k-th cluster in B_r. We then update the prototypes by moving the prototypes towards the mean of the projection vectors in the corresponding clusters as follows:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + \eta_t(\mathbf{P}^{(r)}\bar{\mathbf{u}}_k^{(r)} - \mathbf{m}_k), \tag{40}$$

where $\mathbf{P}^{(r)} = [\mathbf{p}_1^{(r)}, \ldots, \mathbf{p}_{bs}^{(r)}]$ is the projection matrix consisting of the projections of the selected images in this subset, $\bar{\mathbf{u}}_k^{(r)}$ is the k-th column of the corresponding normalised cluster membership indicator matrix. $\mathbf{p}_i^{(r)}$ and $\bar{\mathbf{u}}_k^{(r)}$ are defined as

$$\mathbf{p}_i^{(r)} = \mathbf{p}_{s_{(r-1)*bs+i}}, \quad i = 1, \ldots, bs, \tag{41}$$
$$\bar{\mathbf{u}}_k^{(r)} = \mathbf{u}_k^{(r)} / |\pi_k^{(r)}|, \quad k = 1, \ldots, c. \tag{42}$$

Here $\mathbf{p}_{s_{(r-1)*bs+i}}$ is the $s_{(r-1)*bs+i}$-th column of $\mathbf{P}$. Different from the prototype update in approximate kernel k-means using Eq. (8), the proposed method uses a greedy update strategy by moving the prototype towards the mean of the projection vectors in the corresponding cluster gradually. This allows more room for co-interaction between update of prototypes during the processing over subsets, which inherits the idea of competitive learning that the prototype of each cluster is actually updated gradually after each competition.

Similar to Theorem 4, we can have the following theorem.

Theorem 6. *Let $K_B^{(r)}$ be a sub-matrix from K_B consisting of the selected columns defined as*

$$(K_B^{(r)})_{i,:} = (K_B)_{s_{(r-i)*bs+i},:} \tag{43}$$

Then, the subset update rule (Eq. (40)) can be realised by

$$\nu = (1-\eta_t)^2\nu + 2\eta_t(1-\eta_t)\mathrm{diag}(\gamma^T(K_B^{(r)})^T\bar{U}^{(r)}) + \eta_t^2\mathrm{diag}((\bar{U}^{(r)})^T K_B^{(r)}\hat{K}^{-1}(K_B^{(r)})^T\bar{U}^{(r)}) \tag{44}$$

$$\gamma = (1-\eta_t)\gamma + \eta_t\hat{K}^{-1}(K_B^{(r)})^T\bar{U}^{(r)}, \tag{45}$$

where $\bar{U}^{(r)} = [\bar{\mathbf{u}}_1^{(r)}, \ldots, \bar{\mathbf{u}}_c^{(r)}]$ and diag$(\cdot)$ *is the column vector consisting of the diagonal entries of a matrix.*

Finally, the algorithm proposed in this section is shown in Algorithm 2.

4.2. Computational complexity

Similar to the algorithm proposed in 3.2, the initialisation costs $O(nm^2 + m^3)$. In our algorithm, since data of each subset are clustered into groups (Eq. (38)), the update for the norm vector ν of the prototypes and the prototype coefficients γ costs $O(2c + 2c^2m + cmbs + cm)$, so the time took to update prototypes is $O(cmn)$ in one iteration, because bs is much larger than c in general. Therefore, if excluding the initialisation complexity, the overall optimisation complexity of the proposed algorithm is $O(\tau cmn)$, which is the same as approximate kernel k-means (Chitta et al., 2011). Finally, regarding the space complexity, it is $O(nm)$ for the algorithm presented in this section.

5. Experiments

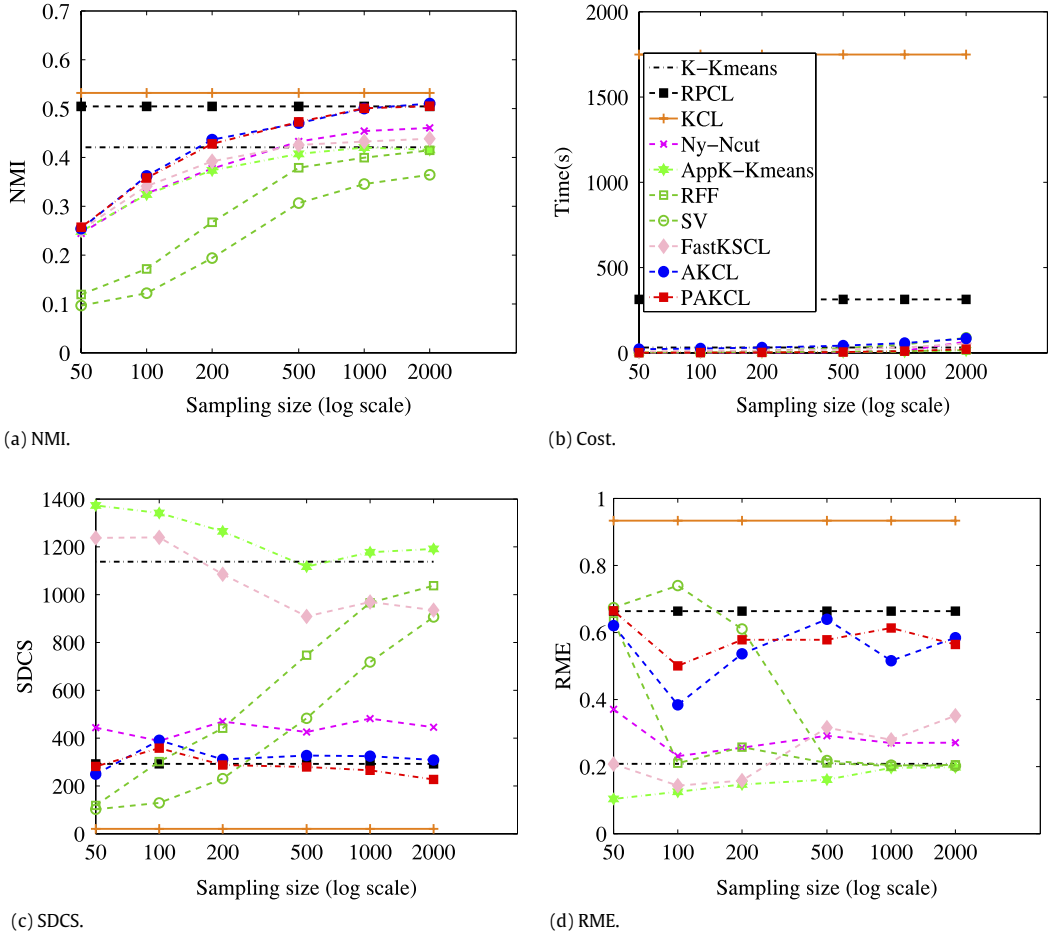
5.1. Datasets

In this section, we evaluate the proposed algorithms and the compared methods on five popular datasets: Newsgroup 20,[1] MNIST,[2] Caltech 101 (Fei-Fei, Fergus, & Perona, 2006), Caltech 256 (Griffin, Holub, & Perona, 2007) and Imagenet (Deng et al., 2009). Table 1 summarises the properties of these datasets and the details are described below.

[1] http://qwone.com/~jason/20Newsgroups/.
[2] http://yann.lecun.com/exdb/mnist.

Algorithm 2 Pseudo-Parallelled Approximate Kernel Competitive Learning

Input: Matrix K_B and $\hat{K}$, the number of clusters c, the subset size bs, the convergence threshold ϵ and the maximum number of iterations t_{max}.
Output: The cluster sets π_k, $k = 1, 2, \ldots, c$.
Initialisation:
Randomly assign labels for all data points, initialise $\bar{U}$ and set $t = 0$.
Compute $\theta = [\hat{K}^{-1}K_B^T]_{m\times n}$, $\gamma = [\theta\bar{U}]_{m\times c}$, $\rho = [diag(K_B\theta)]_{n\times 1}$ and $\nu = [diag(\gamma^T\hat{K}\gamma)]_{c\times 1}$.
repeat
  Randomly generate an index sequence $\mathcal{S} = \{s_i | i = 1, \ldots, n \wedge s_i \in [1, n] \wedge (s_i \neq s_j, \forall i \neq j)\}$
  t = t + 1;
  $R = ceil(n/bs)$;
  for $r = 1, 2, \ldots, R$ do
    1) Assign labels to data points in parallel in the r-th subset by optimising Eq (38) with $U^{(r)}$ and f_k fixed.
    2) Update the prototypes selected as winners in this subset using Eq. (44) and Eq. (45).
    3) Update the frequency for each prototype using Eq. (39).
  end
  Compute the convergence criterion e using Eq. (36).
until $e < \epsilon || t > t_{max}$



(a) NMI. (b) Cost. (c) SDCS. (d) RME.

Fig. 3. Performance of the compared approaches on Newsgroup 20.

- Newsgroup 20: Newsgroup 20 is a collection of about 20,000 newsgroup documents partitioned into 20 different newsgroups. There are about 4.5% of the documents are presented in more than one group. In this paper, we use its sub-collection of 18,828 documents by removing the duplicates. After removing the stop words, ignoring file headers, lowering the upper case characters, stemming the words and selecting the top 2000 words by mutual information as the keywords as done in Dhillon, Mallela, and Modha (2003), we remove the empty documents and form our dataset, which consists of 18,780 documents. Then we represent each document by using the tf-idf weighting scheme.
- MNIST: MNIST is a medium size dataset. It is a subset of the handwritten digits selected from NIST. It consists of 70,000 784-dimensional binary images from 10 classes.
- Caltech 101: This dataset consists of 8677 images of objects covering 101 categories, whose sizes vary from 40 to 800. Most categories contain more than 50 images and the images in this

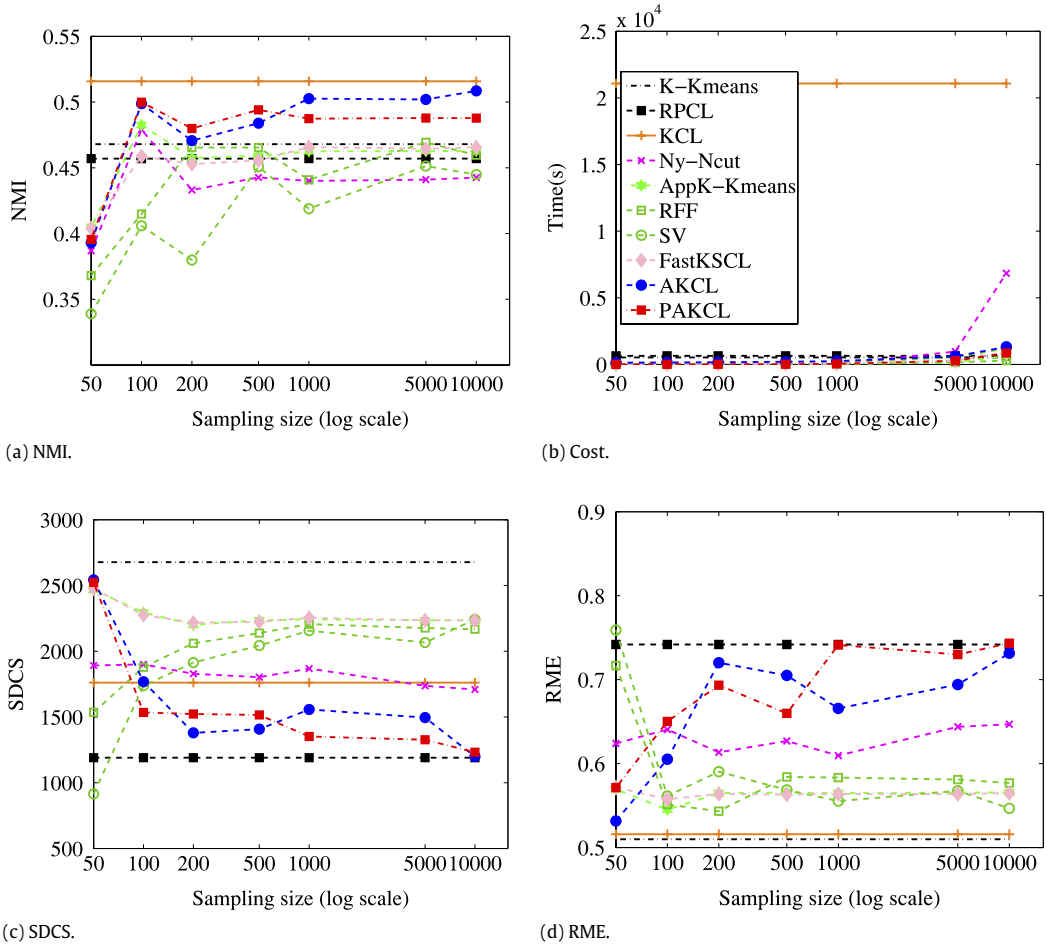


Fig. 4. Performance of the compared approaches on MNIST using the Gaussian kernel.

Table 1
Basic information of datasets.

Dataset	Size	Dimension	#Class
Newsgroup 20	18,780	2,000	20
MNIST	70,000	784	10
Caltech 101	8,677	10,752	101
Caltech 256	29,780	10,752	256
Imagenet	1,261,402	1,000	1000

dataset are left-aligned. Some example images are shown in Fig. 2(a).

- Caltech 256: Caltech 256 is much larger and consists of 30,607 images of objects belonging to 256 categories and 1 clutter. In this paper, we only use all the non-background images from the 256 categories to form our dataset, which consists of 29,780 images. Different from images in Caltech 101, images in this collection are harder to classify, as they are not left-aligned (as shown in Fig. 2(a) and (b)) and distribute in more clusters.
- Imagenet: Imagenet is the most challenging dataset due to the large number of images and large number of classes. It consists of 1,261,402 images. These images are organised into 1000 leaf synsets in the synset tree, in which each leaf synset represents a class of images (Deng et al., 2009). We directly use the SIFT features,[3] and each image is represented by a 1000-dimensional histogram vector. Some example images are shown in Fig. 2(c).

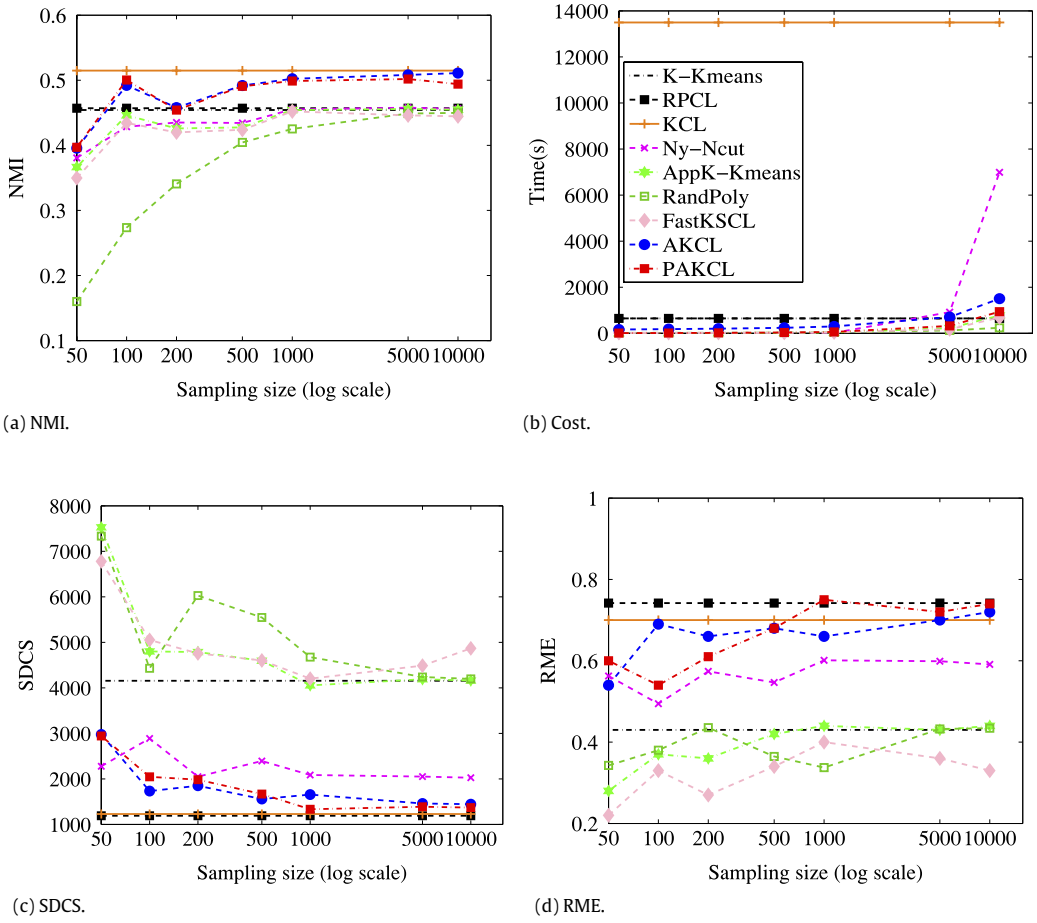
[3] www.image-net.org.

5.2. Settings

5.2.1. Compared methods

In this section, we evaluate the proposed algorithms AKCL (Approximate Kernel Competitive Learning) and PAKCL (Pseudo-Parallelled Approximate Kernel Competitive Learning.)[4] We compare AKCL and PAKCL with two representative kernel clustering methods, namely the kernel k-means (K-Kmeans)[5] (Schölkopf et al., 1998), kernel competitive learning (KCL) (Inokuchi & Miyamoto, 2006; MacDonald & Fyfe, 2000; Mizutani & Miyamoto, 2005; Qin & Suganthan, 2004; Wang et al., 2010), where we implement KCL using a recent development[6] (Wang et al., 2010). We also compare recently developed clustering methods for large scale problem, namely Nyström based normalised cut (Ny-Ncut)[7] (Fowlkes et al., 2004), approximate kernel k-means (AppK-Kmeans)[8] (Chitta et al., 2011), random Fourier feature clustering (RFF and its k-dimensional approximation SV, where k is set to the number of clusters.)[9] (Chitta et al., 2012), random polynomial kernel[10] (Pham & Pagh, 2013) based kernel k-means (RandPoly)

[4] http://sist.sysu.edu.cn/~zhwshi/code/PAKCL.rar.
[5] http://www.mathworks.com/matlabcentral/fileexchange/26182-kernel-k-means.
[6] https://sites.google.com/site/sunnycdwhl/.
[7] http://homes.cs.washington.edu/~sagarwal/code.html.
[8] https://sites.google.com/site/radhacr/academics/projects/software.
[9] https://sites.google.com/site/radhacr/academics/projects/software.
[10] http://www.itu.dk/people/ndap/TensorSketch.m.



(a) NMI. (b) Cost.

(c) SDCS. (d) RME.

Fig. 5. Performance of the compared approaches on MNIST using the Polynomial kernel.

and fast kernel soft competitive learning (FastKSCL) (Schleif et al., 2012), and a priori competitive learning algorithm Rival Penalised Competitive Learning (RPCL) (Xu et al., 1993). Note that the same subset of data is used to span the subspace for learning AKCL, PAKCL, Ny-Ncut, AppK-Kmeans and FastKSCL in order to achieve a fair comparison.

As RFF and SV (Chitta et al., 2012) are approximate kernel methods that are dependent on the shift-invariant kernels (e.g. Gaussian kernel), we only evaluate them on Newsgroup 20, MNIST (using Gaussian kernel) and Imagenet (Chitta et al., 2011, 2012); as RandPoly (Pham & Pagh, 2013) approximates the dot product kernels, such as the normalised Polynomial kernel, we only evaluate it on MNIST (using the normalised Polynomial kernel). We form the Gaussian kernel as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$, where the kernel width is set as $\sqrt{l\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\|\mathbf{x}_i-\mathbf{x}_j\|^2}{n(n-1)}}$ and l is tuned in the range $[0.001, 0.005, 0.01, \ldots, 10, 50, 100]$. The normalised Polynomial kernel is formed as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T\mathbf{x}_j+1)^d$ and the kernel parameter d is set to 5 for all the compared methods by following (Chitta et al., 2011; Zhang & Rudnicky, 2002).

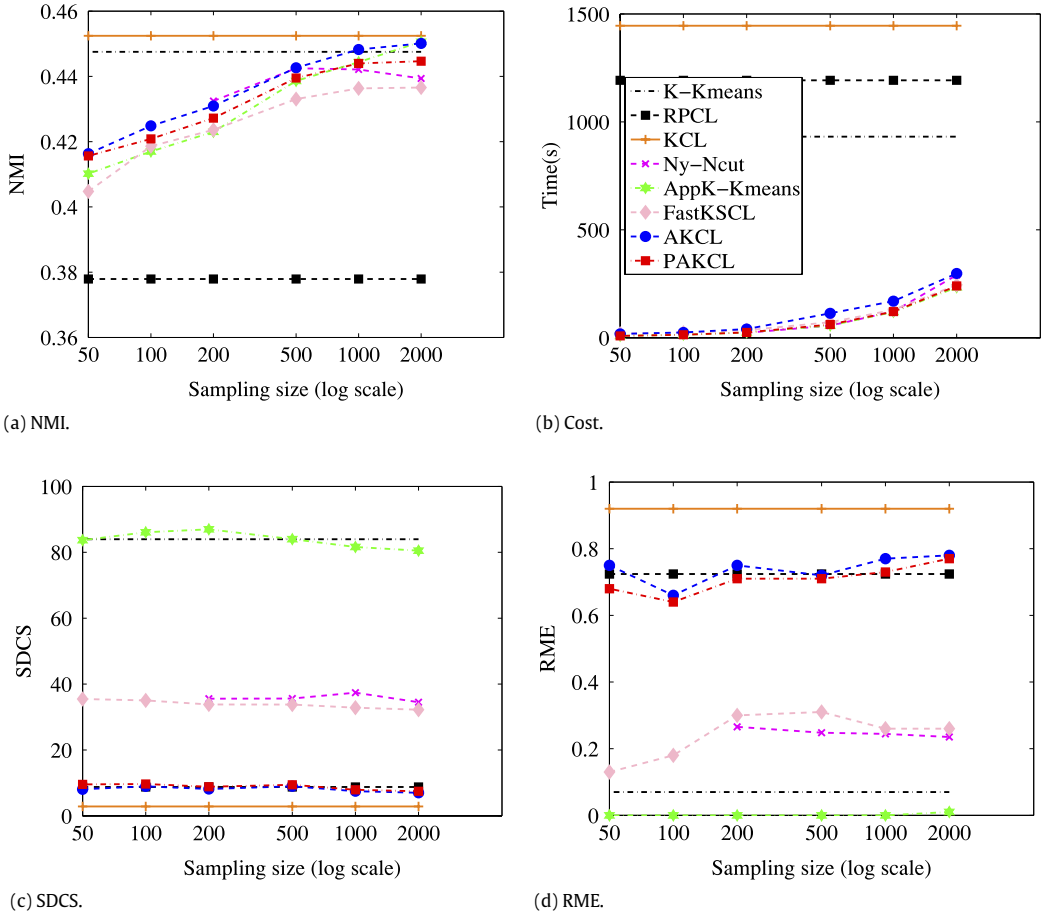
All the algorithms were implemented in Matlab and run on an Intel Xeon 2.67GH with 128 GB RAM. All the results are averaged over 10 runs. In all the experiments, if not otherwise mentioned, the default parameters of our proposed models are: the stopping criterion ϵ is set as $\epsilon = 10^{-4}$, $t_{\max} = 100$ and $bs = 1000$ (for Imagenet, $bs = 10,000$). We set different learning rates for samples in the t-th iteration, i.e., for the ith sample in the t-iteration the learning rate is set as $\eta_{t,i} = ai * (af/ai)^{((t-1)*n+i)/(t_{\max}*n)}$ and $ai = 1.0$, $af = 10^{-5}$, which is called exponentially decaying learning rate. It can give the system the ability to escape from poor local optimum by introducing noise to the system that is gradually removed (Fritzke, 1997).

5.2.2. Evaluation criterion

To evaluate the performance of different methods, we apply normalised mutual information (NMI) (Dhillon, Guan, & Kulis, 2004) to measure the clustering quality, the standard deviation in cluster sizes (SDCS) (Banerjee & Ghosh, 2004) and the ratio of minimum to expect (RME) (Banerjee & Ghosh, 2004) to show how balanced the clusters are. NMI is a commonly used measure of clustering evaluation (Banerjee & Ghosh, 2004; Chen et al., 2011; Chitta et al., 2011; Dhillon et al., 2004; Dhillon, Guan, & Kulis, 2007; Jain, 2010; Wang et al., 2010; Wang, Wang et al., 2012). Its value lies in [0, 1]. The larger the NMI value is, the better the matching between the cluster labels and the true class distribution is. SDCS is the measure that helps understanding the balanced behaviour of a clustering algorithm. It tends to zero when the clusters are well balanced. RME is a useful tool to check whether a clustering algorithm yields extremely small clusters or empty clusters, where 1 indicates the clusters are balanced and 0 indicates there is at least one empty cluster.

5.3. Comparison results

In the following, we report the experimental comparison from medium scale datasets to large scale datasets.



(a) NMI. (b) Cost.

(c) SDCS. (d) RME.

Fig. 6. Performance of the compared approaches on Caltech 101.

5.3.1. Newsgroup 20

Fig. 3 compares the performance of the proposed algorithms with K-Kmeans, RPCL, KCL, Ny-Ncut, AppK-Kmeans, RFF, SV and FastKSCL in terms of NMI, SDCS, RME and the time spent on clustering on Newsgroup 20. As expected, KCL gets the best clustering accuracy and the most balanced clustering due to the effectiveness of competitive learning. However, it costs much more than the other approaches. Compared to KCL, the proposed AKCL and PAKCL approximate it well when the sampling size is larger than 500, i.e., the NMIs they achieve are close to the ones of KCL, and spend much less than KCL. Similar as KCL, the proposed AKCL and PAKCL significantly outperform AppK-Kmeans, RFF, SV, FastKSCL and even K-Kmeans in both aspects of clustering quality and clustering balance measure. For example, compared to Ny-NCut and AppK-Kmeans, AKCL and PAKCL get more than 9.6% and 14.2% higher NMI on average; compared to RFF and SV, they get more than 59.3% and 102.9% higher NMI on average; compared to FastKSCL, they get more than 9.7% higher NMI on average. Although AKCL and PAKCL need to sample more than 1000 data points to achieve better clustering performance than RPCL, their running time spending on clustering is much less than RPCL's. Moreover, as plotted in Fig. 3(a), using a much smaller subset of sampled data points (i.e. the sampling size in the figures), e.g. 500, the proposed methods can significantly outperform or perform closely to the other compared approaches (except KCL and RPCL) in terms of clustering accuracy and clustering balance measure. Hence, by using a smaller sampling size, our approaches can spend much less time to achieve better or comparable performance.

5.3.2. MNIST

Figs. 4 and 5 illustrate the performance of the compared approaches on MNIST using the Gaussian kernel and the Polynomial kernel, respectively. The results show that the proposals outperform the approximate methods with clear improvements on the prediction using both kernels. Take the performance of the compared approaches using Polynomial kernel as an example. PAKCL gets improvement on the prediction accuracy, about 8.6% over Ny-Ncut, 10.1% over AppK-Kmeans, 46.4% over RandPoly and 12.4% over FastKSCL on average; AKCL achieves a much higher accuracy, about 9.3% over Ny-Ncut, 10.8% over AppK-Kmeans, 46.9% over RandPoly and 13.1% over FastKSCL on average.

Similar to the results on Newsgroup 20, our methods also achieve comparable performance when using a much smaller subset of sampled data points on this dataset when using both of kernels. As shown in Figs. 4(a) and 5(a), the performance of our methods using 100 sampled data points nearly outperforms K-Kmeans, RPCL and the other approximate methods that are conducted under the sampling size 10,000. As a result, our methods can save a lot of time on obtaining comparable clustering quality, although they cost a little more time than AppK-Kmeans, Ny-NCut, RFF, SV and RandPoly and FastKSCL when they have the same sampling size.

Moreover, the proposed methods not only obtain better clustering quality, but also get more balanced clusterings, i.e., the SDCSs are much smaller and the RMEs are much larger as shown in Figs. 4(c),(d), 5(c) and (d).

Finally, compared to KCL, Figs. 4(a) and 5(a) show that when $m \geq 1000$, AKCL and PAKCL perform similarly as KCL, but the cost is much less as shown in Figs. 4(b) and 5(b).

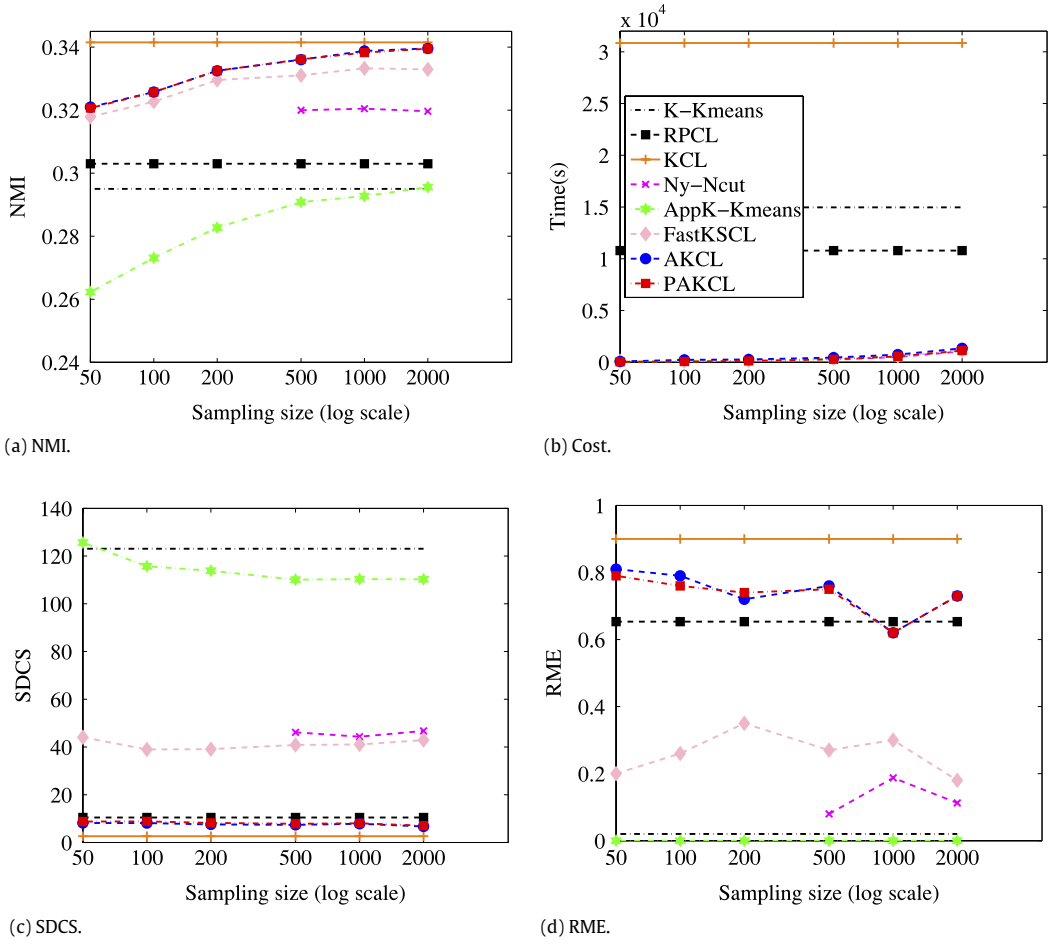


Fig. 7. Performance of the compared approaches on Caltech 256.

5.3.3. Caltech

We conduct experiment on two Caltech databases, namely Caltech 101 and Caltech 256. SIFT features/discriptors (Lowe, 1999) were extracted from each image and used to form the histogram for computing the Spatial Pyramid Matching kernel (Lazebnik, Schmid, & Ponce, 2006), which is suggested in computer vision (Boureau, Bach, Lecun, & Ponce, 2010; Lazebnik et al., 2006; van Gemert, Veenman, Smeulders, & Geusebroek, 2010) for object categorisation. More specifically, to compute the Spatial Pyramid Matching kernel, 128-dimensional SIFT descriptors were extracted from each image. Then the bag-of-features modelling (Fei-Fei & Perona, 2005) was applied to obtain 512 prototypes for the SIFT features and the spatial pyramid modelling was applied to obtain a three-level based pyramid histogram representation, based on which the Spatial Pyramid Matching kernel was computed.

As Ny-Ncut has to do the eigenvalue decomposition of the sampled affinity matrix, the number of sampled columns from the affinity matrix has to be greater than or equal to the number of clusters. Hence, the performance of Ny-Ncut with the sampling size greater than or equal to the number of clusters is reported here.

Caltech 101. For the experiment on Caltech 101, K-Kmeans, AppK-Kmeans and FastKSCL tend to generate very imbalanced clustering and extremely small or even empty clusters (Bradley et al., 2000; Fort et al., 2002). Figs. 6(c) and (d) verify that the proposed methods yield more balanced cluster sets than the other methods (except KCL and RPCL). Note that Fig. 6(c) shows that the SDCSs obtained by the other methods are much higher and Fig. 6(d) shows that K-Kmeans and FastKSCL generate extremely small cluster sets and AppK-Kmeans generates empty cluster sets when $m \leq 1000$ and extremely small cluster sets when $m = 2000$.

Similar to the case in Newsgoup 20, not only obtain more balanced cluster sets, AKCL and PAKCL also obtain more superior clustering performance than AppK-Kmeans and FastKSCL for different sampling size as shown in Fig. 6(a), except the case when the sampling size is set to 2000.

Caltech 256. For the experiment on Caltech 256, all the compared algorithms do not work very well. However, our proposed algorithms still achieve much better performance than K-Kmeans, RPCL, Ny-NCut, and AppK-Kmeans, and obtain comparable performance compared to FastKSCL on this dataset. For example, AKCL and PAKCL gain more than 12.6%, 9.6%, 5.6% and 17.5% higher clustering accuracy on average over K-Kmeans, RPCL, Ny-NCut and AppK-Kmeans, respectively. Similar to the case in Newsgroup 20, Fig. 7(a) indicates that AKCL and PAKCL using a small subset of sampled data points such as 200 data points can still outperform AppK-Kmeans using 2000 sampling data points remarkably and perform closely to FastKSCL using 2000 sampling data points. In addition, Figs. 7(c) and (d) also show that K-Kmeans and AppK-Kmeans more likely tend to yield extremely small cluster sets or empty cluster sets on this dataset due to the increasing number of clusters as compared to the results on Caltech 101. In comparison, the proposed models get almost the same balanced clusterings.

Finally, compared to RPCL, due to the much smaller sampling size than data dimensionality, AKCL and PAKCL spend much less time than RPCL and meanwhile get much better clustering performance on the two Caltech datasets. Compared to KCL, AKCL and PAKCL approximate the performance of KCL very well and they take much less time than KCL on both datasets as illustrated in Figs. 6 and 7. On Caltech 101, AKCL is about 30 times faster and

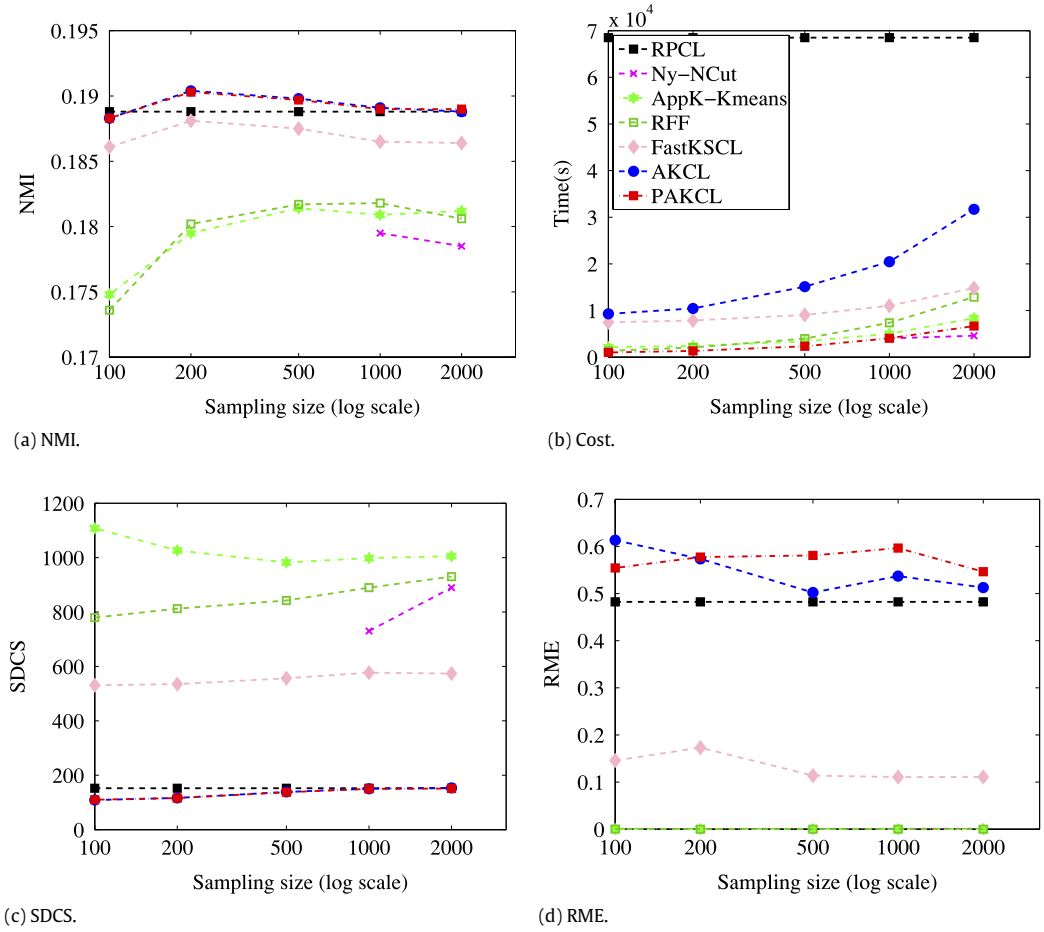


Fig. 8. Performance of the compared approaches on Imagenet.

PAKCL is about 57 times faster; on Caltech 256, AKCL is about 109 times faster and PAKCL is about 210 times faster. Compared to FastKSCL, when the sampling sizes are set to be the same, PAKCL consumes less time than FastKSCL, about tens of seconds. Although AKCL consumes a little more time than FastKSCL, AKCL and PAKCL achieve much better clustering quality in terms of NMI on Caltech 101 and much more balanced clusterings on both Caltech datasets.

5.3.4. Imagenet

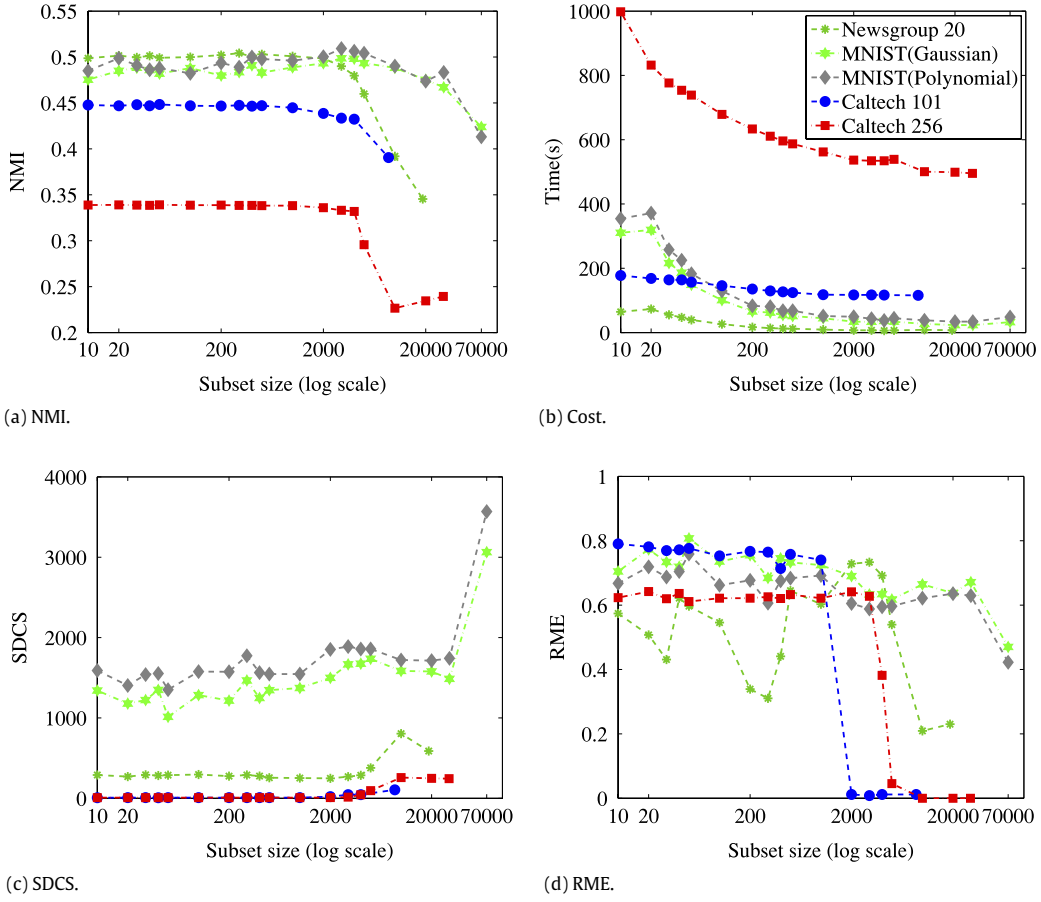
Since Imagenet is a large scale one, and K-Kmeans and KCL are not applicable for large scale clustering problem, we cannot implement K-Kmeans and KCL on it. We evaluate AKCL, PAKCL, RPCL, Ny-Ncut, AppK-Kmeans, RFF and FastKSCL on this dataset. As said in Chitta et al. (2012), when the sampling size m is much larger than the number of clusters c, i.e., $m \gg c$, SV will be an efficient approximation of RFF. Hence, we will not evaluate SV on this dataset because the number of clusters does not differ from the number of sampling size too much in this experiment.

The prediction accuracies measured by NMI are all small for all the compared methods, because Imagenet is large in both aspects of the dataset size and the number of clusters. However, AKCL and PAKCL can still outperform RPCL, Ny-Ncut, AppK-Kmeans, RFF and FastKSCL with improvements on the prediction accuracy. Fig. 8(a) shows that AKCL and PAKCL get 5.6% higher NMI against Ny-Ncut on average, 5.4% higher NMI against AppK-Kmeans on average, 5.4% higher NMI against RFF on average, and 1.3% higher NMI against FastKSCL on average, respectively. As there are a large number of classes in this dataset, Ny-Ncut, AppK-Kmeans, RFF and FastKSCL get very imbalanced cluster sets; in contrast, similar to the cases of the other datasets, AKCL and PAKCL obtain more balanced partitions on this dataset, since the SDCSs of Ny-NCut, AppK-Kmeans, RFF and FastKSCL are much larger than the ones of AKCL and PAKCL as shown in Fig. 8(c) and the RMEs of Ny-NCut, AppK-Kmeans, RFF and FastKSCL equal to 0 or are very close to 0 as shown in Fig. 8(d). Similarly, the performance of Ny-Ncut in the cases that the number of sampled data points is greater than or equal to the number of clusters is reported here.

Finally, we compare the cost the compared approaches spending on clustering. Among all these compared methods, PAKCL achieves the fastest clustering compared to other methods except Ny-NCut. It costs about 4.5% of RPCL's computational time, 66.2% of AppK-Kmeans', 60.7% of RFF's, 27.6% of FastKSCL's and 16.0% of AKCL's as shown in Fig. 8(b). Compared to FastKSCL, although AKCL costs more time than FastKSCL, its clustering qualities in terms of NMI, SDCS and RME are better than the ones of FastKSCL.

5.4. Evaluation of PAKCL: parameters

In this section, we evaluate PAKCL with different subset sizes bs on the first four datasets: Newsgroup 20, MNIST, Caltech 101 and Caltech 256, because Imagenet is too large to evaluate PAKCL with a wide range of subset sizes on it. The sampling size m is set to 1000 for PAKCL, since all the approximate algorithms using this sampling size work well on all the datasets. Fig. 9 shows that when the subset size bs is smaller than about one tenth of the size of the dataset, PAKCL performs robustly on these datasets, that is the NMI value it achieves changes slightly. When bs becomes larger, the performance of PAKCL on these datasets decays quickly. That is because when bs is too large, the size of subset is large, so that the whole performance of PAKCL becomes what approximate kernel



(a) NMI. (b) Cost.

(c) SDCS. (d) RME.

Fig. 9. The performance of PAKCL with different subset sizes on MNIST(Gaussian), MNIST(Polynomial), Newsgroup 20, Caltech 101 and Caltech 256.

k-means can achieve. In this case, the algorithm is more likely to be trapped in a local minimal and converges earlier. Meanwhile, Fig. 9(b) shows that the time spent on clustering decreases along with the increasing subset size, because the larger the subset size is, the more benefits obtained from the matrix manipulation is and the less iterations used is as well.

5.5. Summary

Compared to the other methods, AKCL and PAKCL have the following advantages:

1. AKCL and PAKCL spend much less time and need much less memory space on clustering, and they obtain the similar clustering results as KCL. For example, on Imagenet, AKCL and PAKCL only need about 20 GB memory to keep the matrices for computation (with double precision that needs 8 Bytes to store a real number) when the sampling size is set to be 2000; in comparison, KCL must need about 12,000 GB memory to keep the matrices, which is not applicable. So, AKCL and PAKCL can be applied to large scale clustering problem.
2. AKCL and PAKCL can obtain better clustering under much smaller sampling size than RPCL, AppK-Kmeans and FastKSCL with much less consuming.
3. AKCL and PAKCL overcome the shortage existing in iterative clustering techniques (e.g., K-Kmeans, AppK-Kmeans and FastKSCL) that tend to yield extremely small clusters or even empty clusters in the high dimensional space and in the clustering problem with a large number of classes (Bradley et al., 2000).
4. PAKCL always performs comparably as AKCL, but reduces the computational time a lot by using approximately parallelled processing, as the average optimisation cost of PAKCL is about from 8.5% to 24.8% of the cost of AKCL on these five datasets, where the initialisation cost is excluded.
5. The approximation in AKCL, PAKCL as well as AppK-Kmeans are not restricted to specific kernels, so they can be applied in a wider range of applications. In contrast, the approximation in RFF, SV and RandPoly is kernel dependent, such as RFF and SV for Gaussian kernel and RandPoly for the Polynomial kernel.
6. It is experimentally verified that PAKCL is insensitive to the subset size.

6. Conclusions

In this paper, we address the approximate modelling for kernel competitive learning. Our strategy includes (1) a subspace-approximation based kernel competitive learning model and (2) a pseudo-parallelled approximate kernel competitive learning model. Although the utilisation of the random sampling techniques may restrict the performance, this may be the price one has to pay for dealing with large scale data, as similar random sampling techniques have been also used for handling the large scale clustering recently, such as the approximate kernel k-means (Chitta et al., 2011), Nyström methods (Fowlkes et al., 2004; Williams & Seeger, 2001), and so on. Sampling is effective to approximate the performance of the exact approaches. The experimental results in this work on several datasets including ImageNet also demonstrate its effectiveness in accelerating kernel competitive learning algorithm for large scale datasets and yielding better or comparable performance compared with the related kernel k-means (Schölkopf et al., 1998), rival penalised competitive learning (Xu et al., 1993), kernel competitive learning (Wang et al., 2010), Nyström based normalised cut (Fowlkes et al., 2004), approximate kernel k-means

(Chitta et al., 2011) and the fast kernel soft competitive learning (Schleif et al., 2012). Nevertheless, we think optimising the random sampling could be a future issue for improving quite a lot of approximation techniques used for large scale clustering. Additionally, in this work, despite we develop approximate kernel competitive learning for large scale clustering problems by kernelising the frequency sensitive competitive learning algorithm (Ahalt et al., 1990), the proposals can be extended to other competitive learning methods, such as rival penalised competitive learning.

Acknowledgements

This research was supported by the National Natural Science of Foundation of China (Nos. 61102111, 61173084, 61472456, U1135001), the 12th Five-year Plan China S&T Supporting Programme (No. 2012BAK16B06), Guangzhou Pearl River Science and Technology Rising Star Project under Grant 2013J2200068, and in part by the Guangdong Natural Science Funds for Distinguished Young Scholar under Grant S2013050014265. The code of the proposed method is available at: http://sist.sysu.edu.cn/~zhwshi/code/PAKCL.rar.

References

Ahalt, S. C., Krishnamurthy, A. K., Chen, P., & Melton, D. E. (1990). Competitive learning algorithms for vector quantization. *Neural Networks*, *3*(3), 227–290.

Bădoiu, M., & Indvk, S. H. -P. P. (2002). Approximate clustering via core-sets. In *Proc. of the 34th annual ACM symposium on theory of computing* (pp. 250–257).

Banerjee, A., & Ghosh, J. (2004). Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres. *IEEE Transactions on Neural Networks*, *15*(3), 702–719.

Boureau, Y. -L., Bach, F., Lecun, Y., & Ponce, J. (2010). Learning mid-level features for recognition. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 2559–2566).

Bradley, P. S., Bennett, K. P., & Demiriz, A. (2000). *Constrained k-means clustering, Tech. Rep. MSR-TR-2000-65*. Microsoft Research.

Chen, W.-Y., Song, Y., Bai, H., Lin, C.-J., & Chang, E. Y. (2011). Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(3), 568–586.

Chitta, R., Jin, R., Havens, T. C., & Jain, A. K. (2011). Approximate kernel *k*-means: solution to large scale kernel clustering. In *Proc. of the 17th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 895–903).

Chitta, R., Jin, R., & Jain, A. K. (2012). Efficient kernel clustering using random fourier features. In *Proc. of the 12th IEEE int. conf. on data mining* (pp. 161–170).

Cordeiro, R.L. F., Traina, C. Jr., Traina, A. J. M., López, J., Kang, U., & Faloutsos, C. (2011). Clustering very large multi-dimensional datasets with mapreduce. In *Proc. of the 17th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 690–698).

Cottrell, M., Hammer, B., Hasenfuß, A., & Villmann, T. (2006). Batch and median neural gas. *Neural Networks*, *19*(6), 762–771.

Deng, J., Dong, W., Socher, R., Li, L. -J., Li, K., & Fei-Fei, L. (2009). Imagenet: a large-scale hierarchical image database. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 248–255).

Desieno, D. (1988). Adding a conscience to competitive learning. In *Proc. of IEEE int. conf. on neural networks* (pp. 117–124).

Dhillon, I. S., Guan, Y., & Kulis, B. (2004). Kernel *k*-means, spectral clustering and normalized cuts. In *Proc. of the 10th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 551–556).

Dhillon, I. S., Guan, Y., & Kulis, B. (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(11), 1944–1957.

Dhillon, I. S., Mallela, S., & Modha, D. S. (2003). Information-theoretic co-clustering. In *Proc. of the 9th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 89–98).

Digital Universe Study (2010).

Ene, A., Im, S., & Moseley, B. (2011). Fast clustering using mapreduce. In *Proc. of the 17th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 681–689).

Fei-Fei, L., Fergus, R., & Perona, P. (2006). One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(4), 594–611.

Fei-Fei, L., & Perona, P. (2005). A Bayesian hierarchical model for learning natural scene categories. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 524–531).

Fort, J. -C., Letremy, P., & Cottrell, M. (2002). Advantages and drawbacks of the batch kohonen algorithm. In *Proc. of the 10th European symposium on artificial neural networks* (pp. 223–230).

Fowlkes, C., Belongie, S., Chung, F., & Malik, J. (2004). Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(2), 214–225.

Fritzke, B. (1997). Some competitive learning methods.

Griffin, G., Holub, A., & Perona, P. (2007). *Caltech-256 object category dataset, Tech. Rep. 7694*. California Institute of Technology.

Guha, S., Rastogi, R., & Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. In *Proc. of 1998 ACM SIGMOD int. conf. on management of data* (pp. 73–84).

Huang, Z. (1998). Extensions to the *k*-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, *2*(3), 283–304.

Inokuchi, R., & Miyamoto, S. (2006). Kernel methods for clustering: competitive learning and *c*-means. *International Journal of Uncertainly, Fuzziness and Knowledge-Based Systems*, *14*(4), 481–493.

Jain, A. K. (2010). Data clustering: 50 years beyond *k*-means. *Pattern Recognition Letters*, *31*(8), 651–666.

Kar, P., & Karnick, H. (2012). Random feature maps for dot product kernels. In *Proc. of the 15th int. conf. on artificial intelligence and statistics* (pp. 583–591).

Kashima, H., Ide, T., Kato, T., & Sugiyama, M. (2009). Recent advances and trends in large-scale kernel methods. *IEICE Transactions on Information and Systems*, *E92-D*(7), 1338–1353.

Kaufman, L., & Rousseeuw, P. J. (2005). *Finding groups in data: an introduction to cluster analysis*. Wiley.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*(9), 1464–1480.

Lay, D. C. (2002). *Linear algebra and its applications* (3rd ed.). Addison Wesley.

Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 2169–2178).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proc. of IEEE the 7th int. conf. on computer vision* (pp. 1150–1157).

Lühr, S., & Lazarescu, M. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data & Knowledge Engineering*, *68*(1), 1–27.

MacDonald, D., & Fyfe, C. (2000). The kernel self organising map. In *Proc. of the 4th int. conf. on knowledge-based intelligent engineering systems and allied technologies* (pp. 317–320).

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th Berkely Symposium on Mathematical Statistics and Probability* (pp. 281–297).

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 558–569.

Mizutani, K., & Miyamoto, S. (2005). Kernel-based fuzzy competitive learning clustering. In *Proc. of the 14th IEEE int. conf. on fuzzy systems* (pp. 636–639).

Ng, R. T., & Han, J. (2002). Clarans: a method for clustering objects for spatial data mining. *The IEEE Transactions on Knowledge and Data Engineering*, *14*(5), 1003–1016.

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *Proc. of advances in neural information processing systems* (pp. 849–856).

Nistér, D., & Stewénius, H. (2006). Scalable recognition with a vocabulary tree. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 2161–2168).

Ordonez, C., & Omiecinski, E. (2004). Efficient disk-based *k*-means clustering for relational databases. *The IEEE Transactions on Knowledge and Data Engineering*, *16*(8), 909–921.

Pham, N., & Pagh, R. (2013). Fast and scalable polynomial kernels via explicit feature maps. In *Proc. of the 19th ACM SIGKDD int. conf. on knowledge discovery and data mining* (pp. 239–247).

Philbin, J., Chum, O., Isard, M., Sivic, J., & Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 1–8).

Qin, A. K., & Suganthan, P. N. (2004). Kernel neural gas algorithms with application to cluster analysis. In *Proc. of the 17th int. conf. on pattern recognition* (pp. 617–620).

Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. In *Proc. of advances in neural information processing systems* (pp. 1177–1184).

Schleif, F.-M., Zhu, X., & Hammer, B. (2013). Soft competitive learning for large data sets. In *Advances in intelligent systems and computing: Vol. 185. New trends in databases and information systems* (pp. 141–151).

Schleif, F. -M., Zhu, X., Gisbrecht, A., & Hammer, B. (2012). Fast approximated relational and kernel clustering. In *Proc. of the 21th int. conf. on pattern recognition* (pp. 1229–1232).

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*(5), 1299–1319.

Tsang, I. W., Kwok, J. T., & Cheung, P.-M. (2005). Core vector machines: fast SVM training on very large data sets. *Journal of Machine Learning Research*, *6*, 363–392.

van Gemert, J. C., Veenman, C. J., Smeulders, A. W. M., & Geusebroek, J.-M. (2010). Visual word ambibuity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(7), 1271–1283.

Vesanto, J., & Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, *11*(3), 586–600.

Wang, C. -D., Lai, J. -H., & Zhu, J. -Y. (2010). A conscience on-line learning approach for kernel-based clustering. In *Proc. of the 10th IEEE int. conf. on data mining* (pp. 531–540).

Wang, C.-D., Lai, J.-H., & Zhu, J.-Y. (2012). Graph-based multiprototype competitive learning and its applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *42*(6), 934–946.

Wang, J., Wang, J., Ke, Q., Zeng, G., & Li, S. (2012). Fast approximate *k*-means via cluster closures. In *Proc. of IEEE int. conf. on computer vision and pattern recognition* (pp. 3037–3044).

Williams, C. K. I., & Seeger, M. (2001). Using the nyström method to speed up kernel machines. In *Proc. of advances in neural information processing systems* (pp. 682–688).

Xu, L., Krzyzak, A., & Oja, E. (1993). Rival penalized competitive learning for clustering analysis, RBF net, and curve detection. *IEEE Transactions on Neural Networks*, *4*(4), 636–649.

Xu, R., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, *16*(3), 645–678.

Zhang, Q., Liu, J., & Wang, W. (2007). Incremental subspace clustering over multiple data streams. In *Proc. of the 7th IEEE int. conf. on data mining* (pp. 727–732).

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). Birch: an efficient data clustering method for very large databases. In *Proc. of 1996 ACM SIGMOD int. conf. on management of data* (pp. 103–114).

Zhang, R., & Rudnicky, A. I. (2002). A large scale clustering scheme for kernel *k*-means. In *Proc. of the 16th int. conf. on pattern recognition* (pp. 289–292).