Tsapanos, N., Tefas, A., Nikolaidis, N., & Pitas, I. (2016). Kernel matrix trimming for improved Kernel K-means clustering. In 2015 IEEE International Conference on Image Processing (ICIP 2015): Proceedings of a meeting held 27-30 September 2015, Quebec City, Quebec, Canada. (pp. 2285-2289). Institute of Electrical and Electronics Engineers (IEEE). DOI: 10.1109/ICIP.2015.7351209

Peer reviewed version

KERNEL MATRIX TRIMMING FOR IMPROVED KERNEL K-MEANS CLUSTERING

Nikolaos Tsapanos, Anastasios Tefas, Nikolaos Nikolaidis and Ioannis Pitas

Aristotle University of Thessaloniki

ABSTRACT

The Kernel k-Means algorithm for clustering extends the classic k-Means clustering algorithm. It uses the kernel trick to implicitly calculate distances on a higher dimensional space, thus overcoming the classic algorithm's inability to handle data that are not linearly separable. Given a set of n elements to cluster, the $n \times n$ kernel matrix is calculated, which contains the dot products in the higher dimensional space of every possible combination of two elements. This matrix is then referenced to calculate the distance between an element and a cluster center, as per classic k-Means. In this paper, we propose a novel algorithm for zeroing elements of the kernel matrix, thus trimming the matrix, which results in reduced memory complexity and improved clustering performance.

1. INTRODUCTION

The *k-Means* algorithm [1] is one of the earliest algorithms for clustering data [2]. It is very simple, basic, yet popular and widely used. As its name implies, it involves k cluster centers. When applied to a set of n elements $a_i, i = 1 \dots n$, each represented by a vector in the feature space, it labels each element with the cluster center it is eventually assigned to. In its basic form, it is an iterative process with two steps: assigning the elements to the closest cluster center and then updating each cluster center to the mean of the elements assigned to it in the previous step. This continues until there are no changes, or a set number of iterations has been reached.

A limitation of k-Means is that the surfaces separating the clusters can only be linear hyperplanes in the dimensionality of the elements. This means that its performance on more challenging clustering tasks can be rather poor. In order to overcome this limitation, the classic algorithm has been extended into *Kernel k-Means* [3]. The basic idea behind kernel approaches is to project the data into a higher, or even infinite dimensional space. It is possible for a linear separator in that space to have a non-linear projection back in the original space, thus solving the issue. The *kernel trick* [4] allows us to circumvent the actual projection to the higher dimensional space. The trick involves using a *kernel function* to implicitly calculate the dot products of vectors in the kernel space using the feature space vectors. If $\mathbf{x}_i$, $\mathbf{x}_j$ are the feature vectors representing elements a_i, a_j and $\phi(\mathbf{x}_i)$, $\phi(\mathbf{x}_i)$ are the projections of the corresponding feature vectors on the kernel space, then $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is a kernel function. Different kernel functions correspond to different projections. Finally, distance in the kernel space can be measured using dot products.

In order to have quick, repeated access to the dot products without calculating the kernel function every time, the function is calculated once for every possible combination of two elements. The results are stored in a $n \times n$ matrix $\mathbf{K}$ called the *kernel matrix*, where $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Interestingly, it has been proven that Kernel k-Means, Spectral Clustering and Normalized Graph Cuts are, under the right conditions, mathematically equivalent tasks [5]. The kernel matrix can, therefore, be viewed as the weighted adjacency matrix of a full graph, whose nodes are the elements a_i and whose edge weights are the kernel function values.

The remaining issue with Kernel k-Means is that the kernel matrix grows quadratically with respect to the number of elements n. With the advances in data generation, collection and storage, modern clustering datasets are constantly growing bigger and applying Kernel k-Means becomes problematic. One way to deal with large amounts of data, *Approximate Kernel k-Means* [6], is to restrict the cluster centers from the entire kernel space spanned from all elements to a smaller kernel subspace spanned by randomly sampled elements. Another approach is to work on the m-nearest neighbor graph instead of the full graph [7]. This involves discarding all edges except the m strongest edges for every element from the adjacency matrix, or, in our case, from the kernel matrix.

In this paper, we propose a novel kernel matrix trimming algorithm that reduces the size of the clustering problem, while also improving clustering performance. We consider the kernel matrix edges that connect elements of the same cluster to be "good" and edges connecting elements of different clusters to be "bad". We aim to eliminate the "bad" edges, while retaining as many "good" edges as possible. This is difficult to do using the m-nearest neighbor graph mentioned above, since it requires ground truth knowledge or parameter tuning and a static value for m may not be suitable, if the

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 316564 (IMPART). This publication reflects only the authors views. The European Union is not liable for any use that may be made of the information contained therein.

dataset contains both larger and smaller clusters. In our proposed algorithm, it is possible to retain a different number of edges for different elements. This is achieved by estimating the size of the cluster that each element belongs to. The cluster size estimation is not performed on an element by element basis, as all the elements contribute, when making a decision that one or more clusters of a certain size exist.

The paper is organized as follows: section 2 introduces the basics of the Kernel k-Means algorithm, section 3 details our novel algorithm for trimming the kernel matrix, section 4 presents the experimental evaluation of the proposed Kernel k-Means on trimmed matrix approach and section 5 concludes the paper.

2. KERNEL K-MEANS

In this section we will provide a small introduction to the Kernel k-Means algorithm [8]. Let there be k clusters $\pi_c, c = 1 \dots k$ and elements $a_i, i = 1 \dots n$. Each cluster π_c has a center in the higher dimensional space $\mathbf{m}_c$ and every element a_i has a feature vector $\mathbf{x}_i$, whose projection in the higher dimensional space is $\phi(\mathbf{x}_i)$. Assuming that there is an assignment of every element to a cluster, then cluster π_c's center is computed as

$$\mathbf{m}_c = \frac{\sum_{a_j \in \pi_c} \phi(\mathbf{x}_j)}{|\pi_c|} \tag{1}$$

where $|\pi_c|$ is the number of elements assigned to π_c. The squared distance $D(\mathbf{x}_i, \mathbf{m}_c) = ||\phi(\mathbf{x}_i) - \mathbf{m}_c||^2$ between vector $\mathbf{x}_i$ and $\mathbf{m}_c$ can be written as

$$D(\mathbf{x}_i, \mathbf{m}_c) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T \mathbf{m}_c + \mathbf{m}_c^T \mathbf{m}_c \tag{2}$$

substituting $\mathbf{m}_c$ from (1) into (2) we get

$$D(\mathbf{x}_i, \mathbf{m}_c) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\frac{\sum_{a_j \in \pi_c} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{|\pi_c|} + \\ + \frac{\sum_{a_j \in \pi_x} \sum_{a_l \in \pi_x} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_l)}{|\pi_c|^2}$$

we can calculate the dot products using the kernel function

$$D(\mathbf{x}_i, \mathbf{m}_c) = \kappa(\mathbf{x}_i, \mathbf{x}_i) - 2\frac{\sum_{a_j \in \pi_c} \kappa(\mathbf{x}_i, \mathbf{x}_j)}{|\pi_c|} + \\ + \frac{\sum_{a_j \in \pi_x} \sum_{a_l \in \pi_x} \kappa(\mathbf{x}_j, \mathbf{x}_l)}{|\pi_c|^2}$$

since the kernel function results are stored in the kernel matrix, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = K_{ij}$, we finally obtain

$$D(\mathbf{x}_i, \mathbf{m}_c) = K_{ii} - 2\frac{\sum_{a_j \in \pi_c} K_{ij}}{|\pi_c|} + \frac{\sum_{a_j \in \pi_c} \sum_{a_l \in \pi_c} K_{jl}}{|\pi_c|^2}. \tag{3}$$

In order to simplify the notation, we will use the following symbols: $n_c = |\pi_c|$, $S_c^{(i)} = \sum_{a_j \in \pi_c} K_{ij}$, $C_c = \sum_{a_j \in \pi_c} \sum_{a_l \in \pi_c} K_{jl}$. Thus (3) becomes

$$D(\mathbf{x}_i, \mathbf{m}_c) = K_{ii} - 2\frac{S_c^{(i)}}{n_c} + \frac{C_c}{n_c^2} \tag{4}$$

After measuring the distance of data point $\mathbf{x}_i$ to each of the k clusters centers, the data point is reassigned to the cluster π_c with the minimum distance $D(\mathbf{x}_i, \mathbf{m}_c)$. This is an iterative process, in which the distances are measured and the cluster assignments are updated, until there are no more changes in the assignment, or a maximum number of iterations has been reached. The initial assignment can either be manual, or completely random.

3. TRIMMING ALGORITHM

In general, the aim of the trimming algorithm is to remove the "bad" edges from the kernel matrix. It attempts to accomplish this by determining the size w of the cluster that each element belongs to through a voting system and then only retaining edges the strongest w edges from the corresponding kernel matrix row. Each element casts votes on the various candidate sizes for itself. The votes for each cluster size are summed up for every element. Each size is then assigned a score by a suitability function. The suitability function for cluster size j essentially measures how close the number of votes for j is to the nearest integer, non-zero product of j. For example, if the number of votes for size 50 is 23, then size 50 will not receive a very good score. If, on the other hand, the number of votes for size 50 is 148, then this is a good indication that there are 3 clusters of size 50 and the score is accordingly high. The winning size is the one with the highest score. Every element that voted for the winning size is determined to belong to a cluster of that size and its votes are removed. The process is repeated on the remaining votes, with each size receiving a new, updated score, until there are no votes left. We will proceed to describe this process in further detail.

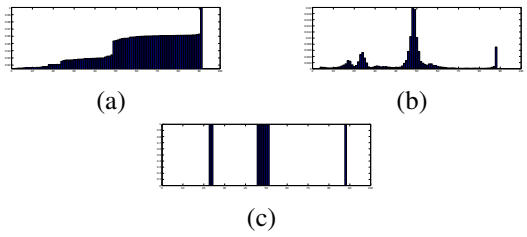


Fig. 1. The vote determination process for a single element: a) the corresponding row values sorted in ascending order, b) the numerically calculated first derivative of the sorted sequence and c) the binary votes.

We begin by sorting each of the n rows of the kernel matrix in ascending order, in a similar fashion to Hartigan's Dip Test for unimodality [9]. Let the sorted sequence for element a_i be mapped to a function $r_i(x)$. We then numerically calculate the first derivative of $r_i(x)$ as:

$$r_i'(x) = \sum_{h=1}^{3} \frac{r_i(x+h) - r_i(x-h)}{2h}.$$

High values for the first derivative imply that there is a possible change in mode and, thus, a possible cluster size. Let $\mathbf{v}^{(i)}$ be the binary vector containing the cluster size votes, $v_j^{(i)} = 1$, if $r_i'(j)$ is among the 10% of highest values of $r_i'(x)$, and $v_j^{(i)} = 0$ otherwise. The determination of a_i's cluster size votes is illustarted in Figure 1.

We add all the voting vectors for every element into vector $\mathbf{v}^* = \sum \mathbf{v}^{(i)}$. Thus, v_j^* is the number of votes for cluster size j. We calculate the score vector $\mathbf{s}$, whose elements are obtained from the elements of $\mathbf{v}^*$ through this function:

$$s_j = (1 - \frac{1}{j}) \max(e^{-|\frac{v_j^* - \lfloor \frac{v_j^*}{j} \rfloor j}{j}|}, e^{-|\frac{v_j^* - \lceil \frac{v_j^*}{j} \rceil j}{j}|}) \quad (5)$$

where $|\frac{v_j^* - \lfloor \frac{v_j^*}{j} \rfloor}{j}|$ is the normalized distance of v_j^* to the closest integer product of j from below and $|\frac{v_j^* - \lceil \frac{v_j^*}{j} \rceil}{j}|$, respectively, from above. Both of these distances are passed through an exponential activation function and the best result is retained. Finally, the score is weighed by $1 - \frac{1}{j}$. This represents the probability that the score is not the result of random chance. Since there are only j possible values for the unweighed score of cluster size j, we assume that there is a $\frac{1}{j}$ probability that this happened by random chance. Therefore, $1 - \frac{1}{j}$ is the probability that the score is valid. This generally favors larger clusters and prevents the process from degenerating into finding a very big number of very small cluster sizes. The winning size $w = \arg\max_j(s_j)$ is selected. Every element a_i that voted for w in its $\mathbf{v}^{(i)}$ is determined to belong to a cluster of size w and its $\mathbf{v}^{(i)}$ is subtracted from $\mathbf{v}^*$ for the next iteration. The voting and scoring process is illustarted in Figure 2.

When there are no more votes, it means that every element has received an estimate of the size of the cluster it belongs to. The trimming of the kernel matrix $\mathbf{K}$ happens in a row-wise manner. Suppose that the estimated cluster size for element a_i is w_i. We can now zero every element K_{ij} in the i-th row of $\mathbf{K}$ that is less than the w_i-th largest value of the row, however, since underestimating the cluster size can result in "good" edges being cut, in practice we use the $(w_i + 0.01n)$-th value, as the cut-off threshold. Let $\hat{\mathbf{K}}$ be the resulting matrix, after every row of $\mathbf{K}$ has been trimmed. Since $\hat{\mathbf{K}}$ may no longer be symmetric, the final trimmed matrix is obtained as $\mathbf{K}^* = \max(\hat{\mathbf{K}}, \hat{\mathbf{K}}^T)$.

Note that the voting process can be implemented in $O(n)$ memory and runs in $O(n^2 \log n)$ time, while subsequent executions of the Kernel k-Means algorithm can run in $O(n_z)$ time and memory, where n_z is the number of non-zero elements of $\mathbf{K}^*$ [10], though note that n_z can be $O(n^2)$ in the worst case. Also note that, as a by-product of this process, we also acquire an estimate for the total number of clusters and their sizes. After determining the cluster size w for each element a_i, we can trim every edge, except the ones with the w largest values. Figure 3 shows the results of applying this process on a sample kernel matrix.

4. EXPERIMENTS

We used the MNIST handwritten digit dataset for our experiments. More specifically, the training set, which contains 60000 images of handwritten digits and the test set, which contains 10000 images where both used for clustering for a total of 70000 samples. In accordance with [11] and [6], each sample image was concatenated into a vector, then each feature of the vector was divided by 255, thus normalizing every feature in $[0, 1]$.

The following kernel functions were used: the *Neural* kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i^T \mathbf{x}_j + \beta)$, the *Polynomial* kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$ and the *Radial Basis Function* (RBF) kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2}$. Again, in accordance with [11] and [6], we set $\alpha = 0.0045$, $\beta = 0.11$ and $d = 5$. For the RBF kernel, we chose $\gamma = 1$. For each function, the full kernel matrix $\mathbf{K}$ was calculated. We then used our algorithm, as described in Section 3, to obtain the trimmed kernel matrix $\mathbf{K}^*$ for every function. We run the Kernel k-Means algorithm 10 times each for all 6 possible approaches (baseline/proposed,Neural/Polynomial/RBF). We then used the *Normalized Mutual Information* (NMI) metric [12] to measure the similarity between the clustering results and the ground truth. We also measure the reduction in the size of the kernel matrix as $\frac{n_z}{n^2}$. The results of this experiment

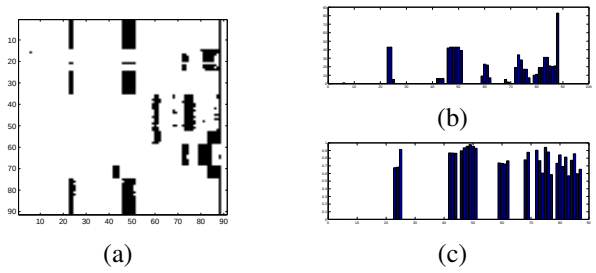
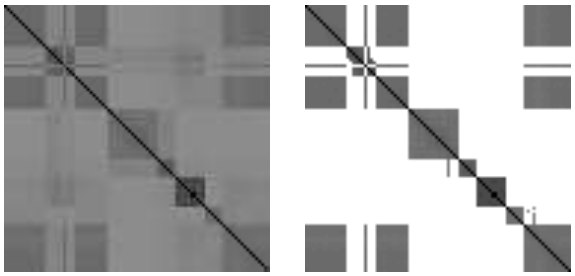


Fig. 2. The voting and scoring process: a) the voting matrix, where each row represents the votes of the corresponding element in black, b) the column-wise sum of the voting matrix c) the results of the scoring function, the winning size is 43.



(a) (b)

Fig. 3. The trimming algorithm applied to a sample kernel matrix. Black is 1, white is 0, the images have been slightly manipulated to improve visualization. a) Original kernel matrix. Ground truth is 6 clusters with sizes $43, 19, 10, 9, 6, 6$. b) The trimmed kernel matrix. Estimated cluster number is 6 clusters with sizes $43, 19, 10, 7, 7, 7$.

Table 1. Experimental results on the MNIST dataset. The baseline column refers to using Kernel k-Means on the full kernel matrix, while the proposed column refers to using Kernel k-Means on the trimmed kernel matrix. The reduction column lists the ratio of retained edges over initial edges.

Kernel	Baseline	Proposed	Reduction
Neural	$0.4982(0.0226)$	$0.4959(0.0066)$	0.0743
Polynomial	$0.4945(0.0136)$	$0.5108(0.0095)$	0.0866
RBF	$0.4936(0.0136)$	$\mathbf{0.5687(0.0312)}$	$\mathbf{0.0439}$

are presented in Table 1, in which NMI values are presented as *mean*(*standard deviation*).

Overviewing the results, we note that the baseline RBF approach performance (0.4936) is worse than both the baseline Neural (0.4982) and baseline Polynomial (0.4945) approaches. Looking at these results, one might think that the RBF kernel function is not the best choice for this problem. Furthermore, it appears that the proposed trimming algorithm hinders the Neural approach (0.4959), but provides enough of an improvement on the Polynomial approach (0.5108) to become better than the baseline Neural combination. However, the proposed RBF approach provides the absolute best performance (0.5687), with a decent $0,0705$ lead over the second best approach. Looking at the kernel matrix size reduction, the proposed RBF approach retained only about 4% of the full kernel matrix, in order to achieve the best performance, while the other two proposed approaches used almost double that (8%). It appears that the RBF kernel suffers most from the presence of "bad" edges, but has better properties regarding the compactness of the clusters than the Neural and Polynomial kernels. Thus, it is able to outperform both, when most of the "bad" edges are removed.

In order to study the performance/kernel matrix size reduction trade-off, we used the approximate kernel k-means algorithm on the same MNIST dataset. We randomly sampled 2000, 4000 and 5000 from the Neural matrix rows and run the experiments 10 times. The NMI performance and corresponding size reduction achieved by approximate kernel k-means can be seen in Table 2, which includes the best performance/reduction of our approach for quick reference. As can be seen, approximate kernel k-means needs about 7% of the kernel matrix, in order to match the full kernel matrix performance (0.4941), while our approach achieves better performance (0.5687) with about 4% of the kernel matrix. However, since our approach requires adjacency lists, in practice it will require double the amount of memory. Concluding this comparison, our approach will require about the same memory space to run and yet provides a significant performance improvement over approximate kernel k-means.

Table 2. Performance/reduction trade-off for approximate kernel k-means [6] and our approach.

Method	NMI	reduction
[6]	$0.4898(0.0067)$	0.0285
	$0.4917(0.0079)$	0.0571
	$0.4941(0.0124)$	0.0714
proposed	$0.5687(0.0162)$	0.0439

5. CONCLUSIONS

In this paper, we have presented a novel kernel matrix trimming algorithm, that aims to remove the "bad" edges, i.e., edges connecting elements that belong to different clusters. Furthermore, we have provided a formalized analysis of the workings of cluster assignment in kernel space and concluded that element assignments to a wrong cluster can be attributed to both the presence of "bad" edges, as well as differences in the relative compactness of the clusters' projection to the higher dimensional space. Our experimental results support this conclusion, as the RBF kernel, while being the worst kernel choice by itself, has better projection properties, once the "bad" edges are reduced. On the contrary, the other two kernels do not benefit as much from such a reduction. As a result, the combination of the proposed trimming algorithm with the RBF kernel provides the best clustering performance.

6. REFERENCES

[1] J. B. Macqueen, "Some methods of classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data cluster-

ing: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sept. 1999.

[3] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, July 1998.

[4] A. Aizerman, E. M. Braverman, and L. I. Rozoner, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control*, vol. 25, pp. 821–837, 1964.

[5] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2004, KDD '04, pp. 551–556, ACM.

[6] Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain, "Approximate kernel k-means: solution to large scale kernel clustering.," in *KDD*, Chid Apte', Joydeep Ghosh, and Padhraic Smyth, Eds. 2011, pp. 895–903, ACM.

[7] Ulrike von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[8] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[9] J. A. Hartigan and P. M. Hartigan, "The dip test of unimodality," *The Annals of Statistics*, p. 7084, 1985.

[10] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis, "A Unified View of Kernel k-means, Spectral Clustering and Graph Cuts," Tech. Rep. TR-04-25, UTCS, July 2004.

[11] Rong Zhang and Alexander I. Rudnicky, "A large scale clustering scheme for kernel k-means.," in *ICPR (4)*, 2002, pp. 289–292.

[12] T. O. Kvalseth, "Entropy and correlation: Some comments," *IEEE Transactions on systems, Man and Cybernetics*, vol. 17, no. 3, pp. 517–519, 1987.