Efficient MapReduce Kernel k-Means for Big Data Clustering

Nikolaos Tsapanos
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
niktsap@aiia.csd.auth.gr

Anastasios Tefas
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
tefas@aiia.csd.auth.gr

Nikolaos Nikolaidis
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
nikolaid@aiia.csd.auth.gr

Ioannis Pitas
Department of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
pitas@aiia.csd.auth.gr

ABSTRACT

Data clustering is an unsupervised learning task that has found many applications in various scientific fields. The goal is to find subgroups of closely related data samples (clusters) in a set of unlabeled data. A classic clustering algorithm is the so-called k-Means. It is very popular, however, it is also unable to handle cases in which the clusters are not linearly separable. Kernel k-Means is a state of the art clustering algorithm, which employs the kernel trick, in order to perform clustering on a higher dimensionality space, thus overcoming the limitations of classic k-Means regarding the non linear separability of the input data. It has recently received a distributed implementation, named Trimmed Kernel k-Means, following the MapReduce distributed computing model. In addition to performing the computations in a distributed manner, Trimmed Kernel k-Means also trims the kernel matrix, in order to reduce the memory requirements and improve performance. The trimming of each row of the kernel matrix is achieved by attempting to estimate the cardinality of the cluster that the corresponding sample belongs to, and removing the kernel matrix entries connecting the sample to samples that probably belong to another cluster. The Spark cluster computing framework was used for the distributed implementation. In this paper, we present a distributed clustering scheme that is based on Trimmed Kernel k-Means, which employs subsampling, in order to be able to efficiently perform clustering on an extremely large dataset. The results indicate that the proposed method run much faster than the original Trimmed Kernel k-Means, while still providing clustering performance competitive with other state of the art kernel approaches.

SETN '16, May 18-20, 2016, Thessaloniki, Greece

DOI: http://dx.doi.org/10.1145/2903220.2903255

CCS Concepts

•Information systems → Clustering; •Computer systems organization → Cloud computing; •Software and its engineering → Development frameworks and environments;

Keywords

Kernel k-Means, clustering, Big Data, distributed computing, MapReduce

1. INTRODUCTION

The objective of *data clustering* is to divide a given group of unlabeled data samples in subgroups (*clusters*), so that data samples belonging to the same cluster are similar to each and dissimilar to data samples belonging to any other clusters. Clustering has found many applications in different scientific fields. Despite the fact that there has been an extremely rich bibliography on this subject for years now [11], it is still an active research field.

Kernel k-Means [18] provides a popular starting point for many state of the art clustering schemes [22, 23, 12, 7]. A recent survey on kernel clustering methods can be found in [9], while [14] presents a comparative study which supports the superiority of kernel clustering methods, over more conventional clustering approaches. The basic idea behind kernel approaches is to project the data into a higher, or even infinite dimensional space. The *kernel trick* [2] 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.

An arising research trend is the so-called *Big Data* research. With the recent advances in technology, digital data is being generated, stored and broadcast at unprecedented rates. Digital cameras, including those in cell phones, are widely available and people all over the world are taking pictures or shooting video clips. The bandwidth of Internet Service Providers has also improved to the point where broadband connections are very common. The Web itself has been growing at ever increasing rates. The connection

graphs of various social networks easily number nodes in the millions.

Distributed computing can provide the means to handle problems on very large datasets that would otherwise be almost impossible to solve [1]. It provides virtually limitless memory and processing power. Provided that a task can be split into many independent subtasks, then it can theoretically be performed in a reasonable amount of time, regardless of the data size, given enough processing units. A distributed approach that can work with any serial clustering algorithm entails using the serial algorithm on data subsets, then merging the clusters [8]. Distributed versions of other clustering algorithms related to Kernel k-Means, like classic k-Means [17] and k-Medians [6] have already been discussed.

Recently, a distributed, MapReduce based implementation of Kernel k-Means, named Trimmed Kernel k-Means, was proposed in [19]. It involves calculating the kernel matrix of the input dataset, then trimming that matrix, in order to both reduce the kernel matrix size and improve clustering results. In this paper, we extend the MapReduce implementation of [19] to include a subsampling step in the begining, which provides a much smaller subset of samples on which we perform the distributed Trimmed Kernel k-Means algorithm to obtain a labelling of the subset and then assign the rest of the samples to the nearest labeled sample. We use this approach, in order to perform clustering on a dataset that contains $2,864,056$ samples.

The paper is organized as follows: Sections 2 and 3 briefly introduce Kernel k-Means and the MapReduce distributed computing framework, respectively. Section 3 fully details an efficient MapReduce implementation for Trimmed Kernel k-Means. Section 5 provides the experimental evaluation of the proposed MapReduce implementation. Finally, Section 6 conludes the paper.

2. KERNEL K-MEANS

The Kernel k-Means algorithm [5] is an extension of the classic k-Means clustering algorithm. Taking advantage of the kernel trick, it implicitly projects the data onto a higher dimensional space and measures Euclidean distances between data samples in that space. This circumvents the limitation of linear separability imposed by k-Means. Let there be k clusters $C_\delta, \delta = 1, \ldots, k$ and data samples $a_i, i = 1, \ldots, n$, each of which is represented by the feature vectors $\mathbf{x}_i, i = 1, \ldots, n$. Each cluster C_δ has a center $\mathbf{m}_\delta$ in the higher dimensional space $\mathbb{R}^{d'} (d << d')$, where $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ is the mapping function. Assuming that there is an assignment of every data sample to a cluster, then the center of cluster C_δ is computed as follows:

$$\mathbf{m}_\delta = \frac{\sum_{a_j \in C_\delta} \phi(\mathbf{x}_j)}{|C_\delta|}, \tag{1}$$

where $|C_\delta|$ is the cardinality of cluster C_δ. The squared distance $D(\mathbf{x}_i, \mathbf{m}_\delta) = ||\phi(\mathbf{x}_i) - \mathbf{m}_\delta||^2$ between the vectors $\mathbf{x}_i$ and $\mathbf{m}_\delta$ can be written as:

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

By substituting $\mathbf{m}_\delta$ from (1) into (2), we get:

$$\begin{aligned} D(\mathbf{x}_i, \mathbf{m}_\delta) &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\frac{\sum_{a_j \in C_\delta} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{|C_\delta|} + \\ &+ \frac{\sum_{a_j \in C_\delta} \sum_{a_l \in C_\delta} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_l)}{|C_\delta|^2} = \\ &= \kappa(\mathbf{x}_i, \mathbf{x}_i) - 2\frac{\sum_{a_j \in C_\delta} \kappa(\mathbf{x}_i, \mathbf{x}_j)}{|C_\delta|} + \\ &+ \frac{\sum_{a_j \in C_\delta} \sum_{a_l \in C_\delta} \kappa(\mathbf{x}_j, \mathbf{x}_l)}{|C_\delta|^2} = \\ &= K_{ii} - 2\frac{\sum_{a_j \in C_\delta} K_{ij}}{|C_\delta|} + \frac{\sum_{a_j \in C_\delta} \sum_{a_l \in C_\delta} K_{jl}}{|C_\delta|^2} = \\ &= K_{ii} - 2\frac{S_\delta^{(i)}}{n_\delta} + \frac{C_\delta}{n_\delta^2}, \end{aligned} \tag{3}$$

where $S_\delta^{(i)} = \sum_{a_j \in C_\delta} K_{ij}$, $T_\delta = \sum_{a_j \in C_\delta} \sum_{a_l \in C_\delta} K_{jl}$ and $n_\delta = |C_\delta|$.

After measuring the distance of data sample $\mathbf{x}_i$ to each of the k clusters centers, the data sample is reassigned to the cluster C_δ with the minimum distance $D(\mathbf{x}_i, \mathbf{m}_\delta)$. 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 cluster entry assignments, or a maximum number of iterations has been reached. The initial cluster entry assignments can either be manual, or completely random.

A convenient way to have quick, repeated access to the dot products without calculating the kernel function every time, is to calculate the function once for every possible combination of two data samples. The results can be stored in a $n \times n$ matrix $\mathbf{K}$ called the *kernel matrix*, where $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. This means that the i-th row of the kernel matrix contains the kernel function entry for every possible sample combination that includes $\mathbf{x}_i$. We will use $\mathbf{K}^*$ to denote the trimmed kernel matrix that is involved.

3. MAPREDUCE FRAMEWORK

The *MapReduce* programming model for distributed computing was inspired by the map and reduce procedures of functional programming languages, like Lisp [4]. MapReduce implementations include Hadoop and Spark. It simplifies the coding of distributed programs that follow this model. It was specifically developed to allow easy processing of very big datasets on computing clusters consisting of many workers. A master node in the MapReduce framework automatically splits the dataset up into smaller data sample collections and distributes them to the workers, where each worker can process the assigned data collection, independently of other workers.

As the name implies, there are two major components to this programming model. The *Map* command, in which every worker applies a user defined function to each data sample. Each worker can then return the results to the master node, thus computing that function output for the entire dataset. Additionally using the *Reduce* command, a worker applies a commutative and associative operation to collect the data elements, or the results of a previously mapped function, into a single result. As the operation is commutative and associative, the results for each worker are indepen-

dent from other workers and they can also be combined in the same way on the master node. A variation of the Reduce command is ReduceByKey, in which, given a distributed set of (*key*, *value*) pairs and a target operation, the operation is performed on the *value* parts for each key separately. If there were k total keys, then the output would be k (*key*, *total*) pairs, where each *total* is the result of performing the operation only on the *value* parts that are associated with the specific *key*.

4. EFFICIENT MAPREDUCE KERNEL K-MEANS

A theoretical model for the efficiency of MapReduce computations is presented in [13]. It introduces the *MapReduce Class* ($\mathcal{MRC}$) of algorithms, which enforces limitations on the memory, number of processors and execution time of an algorithm that belongs to it. We summarize the definition of $\mathcal{MRC}^i$ from [13] here, for ease of reference:

DEFINITION 1. *Fix an $\epsilon > 0$. An alforithm belongs to $\mathcal{MRC}^i$ if:*

- *The Map and Reduce operations are implemented by a RAM with $O(\log n)$ words, $O(n^{1-\epsilon})$ available space and execute in time polynomial to N.*
- *The total memory space required is $O(N^{2-2\epsilon})$.*
- *The number of MapReduce rounds R is $O(\log^i N)$.*

Note that the above restrictions imply the existence of $\Theta(N^{1-\epsilon})$ available machines, something that is also clearly stated in [13]. In order for a clustering algorithm to belong to $\mathcal{MRC}^i$ for some i, a usual practice is to subsample the input dataset [8, 6]. We fix $\epsilon = 0.5$. The kernel matrix computation, kernel matrix trimming and Kernel k-Means steps detailed below have been proven to belong to $\mathcal{MRC}^1$, if they are applied to $O(\sqrt{N})$ sata samples [19].

4.1 Subsampling

This step is very simple and straight-forward. When presented with a dataset containing N samples, we randomly select $n = O(\sqrt{N})$ of them to perform clustering on. This is achieved by shuffling the sample feature vectors, then writing the first n to one file and the rest $N - n$ to another file.

4.2 Distributed kernel matrix computation

Computing the kernel matrix under the MapReduce model is pretty straight forward. Assuming there are n data samples, each of which has d features, we read the data samples into n d-dimensional data vectors, which are distributed to the cluster worker nodes. Then we iterate through every data vector and map the kernel function of the current vector with every other vector. This provides us with a single row of the kernel matrix, which we can then write to the disk. After n iterations, the computation is complete.

4.3 Distributed kernel matrix trimming

After the kernel matrix has been computed and written to the disk, we read the n-dimensional kernel matrix rows and distribute them to the cluster nodes. Note that we shall never need every one of these rows in memory at the same time. Therefore, the framework can swap them to and from the disk, whenever any row is needed. This is an iterative process, in which the nodes vote for cluster cardinalities, the winning cardinality is determined, the votes of the nodes that voted for the winner are removed and the corresponding rows are trimmed.

We begin an iteration by mapping a sorting function on every matrix row. We then map the numerical derivative computation function. Finally, we map a function that returns the vote vectors $\mathbf{v}_i$ of each node, as described in [19]. We use the Reduce operation to add up all the voting vectors into the vector $\mathbf{v}^*$ containing the total votes for every cluster cardinality. The scoring function is applied to this vector per data sample and the winning cluster cardinality w is determined.

In order to remove the winning votes, we map a function that takes the winning cluster cardinality w and outputs the vote vector $\mathbf{v}_i$ of a node, if that node voted for cardinality w, or an all-zero vector $\mathbf{0}$ otherwise. Again, we use the Reduce operation to obtain the vector summing the winning votes, which is then subtracted from the total votes vector $\mathbf{v}^*$, to obtain the remaining votes. As a final step of the iteration, we map the trimming function, which sets the row entry of every row that voted for w, which is not in the top w highest weights, to zero. This completes an iteration step. The process is repeated, until the cluster cardinality of every node is determined and all the rows are trimmed accordingly.

4.4 Distributed Kernel k-Means

In order to save memory, instead of reading the full kernel $n \times n$ matrix $\mathbf{K}$ as a set of n n-dimensional data vectors, we instead read the trimmed kernel matrix that resulted from the previous step as a set of n adjacency lists. The adjacency list for row $\mathbf{k}_i^*$ contains all the non-zero K_{ij}^* of the trimmed kernel matrix $\mathbf{K}^*$.

We initialize the data sample assignment to clusters randomly. The assignment is a n-dimensional vector $\mathbf{o}$, where the i-th entry indicates which cluster (1 to k) data sample a_i belongs to. This assignment is updated at every iteration and will eventually contain the final cluster assignment of every data sample.

We will now provide an algorithm to compute (3) in a distributed fashion. Note that the sum $\sum_{a_j \in C_\delta} \sum_{a_l \in C_\delta} K_{jl}$ remains the same for every individual cluster C_δ. Therefore, it only must be computed once for each corresponding cluster. The first step is to compute the k such sums. This can be accomplished by mapping a function that takes the cluster assignment vector $\mathbf{o}$, the node ID j and the node adjacency list L as arguments and returns a (*key*, *value*) pair. In such a pair, *key* is the cluster that data sample a_j is assigned to (o_j) and *value* is the partial sum $\sum_{a_l \in C_\delta} K_{jl}$, where C_δ is the cluster identified by *key* and the entries K_{jl} are retrieved from the adjacency list L. The function goes through the node adjacency list and sums every entry that belongs to the same cluster as node j. The total sums for every cluster are obtained from these (*key*, *value*) pairs by applying the ReduceByKey operation to add the appropriate partial sums for each cluster and store them in vector $\mathbf{q}$.

In the next distributed processing step, the distance computations are completed and the new node assignments are determined in the same function. This is accomplished by mapping a function that takes the cluster assignment vector $\mathbf{o}$, the node ID i and the cluster sums vector $\mathbf{q}$ as argu-

Table 1: The perfomance comparison between the proposed method, Approximate Kernel k-Means and Trimmed Kernel k-Means operating on the entire dataset

	Approximate Kernel k-Means [3]	Trimmed Kernel k-Means [19]	Proposed
Computational time	24 hours [19]	14 days [19]	9 hours
NMI performance	0.8402 [19]	0.857 [19]	0.8412

ments and returns the new cluster assignment for node i. The function initializes a vector $\mathbf{d}_i$, which is meant to store the distance of data sample a_i to every cluster, so each entry is initialized to $\mathbf{d}_{i\delta} = K_{ii} + \frac{1}{|C_\delta|^2}\mathbf{q}_\kappa$. It then goes through i node adjacency list L_i and subtracts the corresponding values $2\frac{K_{ij}}{|C_\delta|}$ from the appropriate entry in vector $\mathbf{d}$. When it goes through the entire list, then each entry of vector $\mathbf{d}$ will contain the value of $-2\frac{\sum_{a_j \in C_\delta} K_{ij}}{|C_\delta|} + \frac{\sum_{a_j \in C_\delta}\sum_{a_l \in C_\delta} K_{jl}}{|C_\delta|^2}$ for every cluster. The new cluster assignment of node i is determined by the minimum entry in vector $\mathbf{d}$.

4.5 Distributed nearest neighbor assignment

At this point, a subset of $n = O(\sqrt{N})$ labeled data samples. Since we fixed $\epsilon = 0.5$, The data samples can fit in the memory of every worker, which is $O(N^{0.5})$. Furthermore, the rest of the $N - n$ samples can be distributed to the $O(N^{0.5})$ workers, each of which has $O(N^{0.5})$ available memory. Once this is accomplished, a single MapReduce step is required to find the nearest neighbor for each unlabeled point. Therefore, this step belongs to $\mathcal{MRC}^0$.

We map a functions that takes the feature vectors of the labeled data samples, their corresponding labels and the feature vector of an unlabeled data sample as arguments and returns the label of the data sample that is closest to the unlabeled sample. This is achieved by calculating the Euclidean distance between each labels sample and the input unlabeled sample, updating the minimum value and the corresponding label, aswe go through the labeled samples. The resulting labels are returned to the master and written to the disk.

5. EXPERIMENTS

In this section, we presented the results of applying our Efficient MapReduce Kernel k-Means approach to the Youtube Faces dataset, which was also used in [19]. It consists of LBP descriptors for faces of various celebrities, e.g., actors, athletes and politicians, extracted from Youtube videos [20]. There are 3 different, yet closely related types of descriptors provided by the dataset: Local Binary Patterns (LBP) [16], Center-Symmetric LBP (CSLBP) [10] and Four-Patch LBP (FPLBP) [21]. For our experiments, we selected the original LBP features, which yield the best performance in [20].

The dataset contains $n = 621126$ samples, each of which is described by a 1770-dimensional feature vector. Since the feature vectors (LBP) provided by the database are histograms, we used the *Histogram Intersection* kernel, $\kappa(\mathbf{x}, \mathbf{y}) = \sum \min(x_i, y_i)$, for this experiment. The number of ground truth clusters is 1595.

We shuffled the feature vectors and selected 100000 of them, on which we performed Trimmed Kernel k-Means. After obtaining labels for these samples, we used them to label the remaining 521126 samples on a nearest neighbor basis. In order to compare our results with another state of the art Kernel k-Means based approach, we also used Approximate Kernel k-Means [3]. We used the *Normalized Mutual Information* (NMI) metric [15] to measure the similarity between the clustering results and the ground truth in both cases.

The MapReduce experiments were carried out in a cluster with 8 VirtualBox virtual machines as workers, each of which had 2 processing cores and 4GB of memory devoted to the computing cluster. The master node had 16GB of RAM and did not run any worker. All nodes where connected on the same LAN and read and wrote data on a Network Attached Storage (NAS) device. The results of this experiment are presented in Table 1.

Overviewing the results, we can see that our proposed approach provided results in a few hours, which is orders of magnitude faster than the 2 weeks that computing and using the kernel matrix of the entire dataset takes. Additionally, the performance was marginally better than that of the Approximate Kernel k-Means approach, though still below the Trimmed Kernel k-Means on the entire dataset.

6. CONCLUSIONS

In this paper, we have presented a distributed clustering scheme that is capable of efficiently operating on extremely large datasets. It is based on a state of the art, distributed variation of the Kernel k-Means algorithm. Through subsampling, a dataset can be reduced to a more manageable size. The Trimmed Kernel k-Means [19] can then be used to obtain a clustering of the subsampled input dataset. After these labels are obtained, they can be used to label the entire original dataset, through nearest neighbor assignment.

This makes the proposed distributed clustering framework efficient, as it can be proven to belong to the $\mathcal{MRC}^1$ class of MapReduce algorithms. The proposed approach runs much faster than Trimmed Kernel k-Means operating on the full kernel matrix, while still providing an improvement in performance over Approximate Kernel k-Means.

Acknowledgment

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.

7. REFERENCES

[1] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: Current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

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

[3] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain. Approximate kernel k-means: solution to large scale kernel clustering. In C. Apte', J. Ghosh, and P. Smyth, editors, *KDD*, pages 895–903. ACM, 2011.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[5] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11):1944–1957, Nov. 2007.

[6] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 681–689, New York, NY, USA, 2011. ACM.

[7] M. R. Ferreira and F. de A.T. de Carvalho. Kernel-based hard clustering methods in the feature space with automatic variable weighting. *Pattern Recognition*, (0):–, 2014.

[8] R. L. Ferreira Cordeiro, C. Traina, Junior, A. J. Machado Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 690–698, New York, NY, USA, 2011. ACM.

[9] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176 – 190, 2008.

[10] M. Heikkila, M. Pietikainen, and C. Schmid. Description of interest regions with center-symmetric local binary patterns. In P. Kalra and S. Peleg, editors, *5th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP '06)*, volume 4338 of *Lecture Notes in Computer Science (LNCS)*, pages 58–69, Madurai, India, 2006. Springer-Verlag.

[11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[12] H. Jia, Y. ming Cheung, and J. Liu. Cooperative and penalized competitive learning with application to kernel-based clustering. *Pattern Recognition*, (0):–, 2014.

[13] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[14] D.-W. Kim, K. Y. Lee, D. Lee, and K. H. Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognition*, 38(4):607 – 611, 2005.

[15] T. O. Kvålseth. Entropy and correlation: Some comments. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):517–519, 1987.

[16] T. Ojala, M. Pietikäinen, and D. Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, Jan. 1996.

[17] L. M. Rodrigues, L. E. Zárate, C. N. Nobre, and H. C. Freitas. Parallel and distributed kmeans to identify the translation initiation site of proteins. In *SMC*, pages 1639–1645. IEEE, 2012.

[18] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319, July 1998.

[19] N. Tsapanos, A. Tefas, N. Nikolaidis, and I. Pitas. A distributed framework for trimmed kernel k-means clustering. *Pattern Recognition*, 48(8):2685 – 2698, 2015.

[20] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *in Proc. IEEE Conf. Comput. Vision Pattern Recognition*, 2011.

[21] L. Wolf, T. Hassner, and Y. Taigman. Descriptor based methods in the wild. In *Real-Life Images workshop at the European Conference on Computer Vision (ECCV)*, October 2008.

[22] S. Yu, L.-C. Tranchevent, X. Liu, W. Glanzel, J. A. Suykens, B. D. Moor, and Y. Moreau. Optimized data fusion for kernel k-means clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):1031–1039, 2012.

[23] F. Zhou, F. De la Torre Frade, and J. K. Hodgins. Hierarchical aligned cluster analysis for temporal clustering of human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(3):582–596, March 2013.