Proceedings of the 16th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2016
4–8 July, 2016.

Cloud implementation of the K-means algorithm for hyperspectral image analysis

Juan Mario Haut[1], Mercedes Paoletti[1], Javier Plaza[1] and Antonio Plaza[1]

[1] *Department of Technology of Computers and Communications, University of Extremadura, Escuela Politecnica, Avda. de la Universidad s/n*

emails: `juanmariohaut@unex.es`, `mpaolett@alumnos.unex.es`, `jplaza@unex.es`, `aplaza@unex.es`

Abstract

Remotely sensed hyperspectral imaging offers the possibility to collect hundreds of images, at different wavelength channels, for the same area on the surface of the Earth. Hyperspectral images are characterized by their large volume and dimensionality, which makes their processing and storage difficult. As a result, several techniques have been developed in previous years to perform hyperspectral image analysis on high performance computing architectures. However, the application of cloud computing techniques has not been as widespread. There are many potential advantages in exploiting cloud computing architectures for distributed hyperspectral image analysis. In this paper, we present a cloud implementation (developed using Apache Spark) of the popular K-means algorithm for unsupervised hyperspectral image clustering. The experimental results suggest that cloud architectures allow for the efficient distributed processing of large hyperspectral image data sets.

Key words: hyperspectral imaging, cloud computing, k-means clustering.

1 Introduction

Hyperspectral images comprise hundred of contiguous spectral bands, thus imposing significant requirements in terms of storage and data processing. These requirements have increased exponentially with the technological advances in satellite and airbone remote sensing, leading to the creation of high-dimensional hyperspectral data repositories [1].

The availability of new hyperspectral missions is now generating a continuous stream of multi/hyperspectral data, and this has introduced important challenges for scalable and efficient processing of hyperspectral data in the context of different applications [1]. For instance, the NASA Jet Propulsion Laboratory's Airbone Visible/Infrared Imaging Spectrometer (AVIRIS) [2] has a data collection rate of 2.5 MB/s (nearly 9 GB/hour). A similar case is the space-borne Hyperion instrument [1], which collects almost 71.9 GB/hour (over 1.6 TB/day). Most of the satellite missions that will be soon in operation, such as the environmental mapping and analysis program (EnMAP)[1] present similar data collection ratios. Hyperspectral data repositories are becoming increasingly massive and often distributed among several geographic locations due to their volume, which makes hard to meet the storage and computational requirements of large-scale hyperspectral data processing applications without resorting to distributed computing facilities.

In recent years, cloud computing platforms have been adopted for remotely sensed data processing. The cloud is now a standard for distributed computing due to its advanced capabilities for internet-scale computing, service-oriented computing, and high-performance computing. The use of cloud computing for the analysis of large hyperspectral data repositories can be considered a natural solution and an evolution of previously developed techniques for other kinds of computing platforms [3]. However, there are few efforts in the recent literature oriented to the exploitation of cloud computing infrastructure for hyperspectral imaging techniques in general, and for unsupervised clustering algorithms in particular.

Clustering can be defined as a segmentation process in which pixels are assigned into a group that represents a specific land-cover class [4]. The main advantage of clustering is that there is no need for labeled samples which are common in supervised classification techniques [5]. In this regard, clustering offers an unsupervised alternative that has been widely used in various fields. However, clustering is also a very challenging task due to the large spectral variability and complex spatial structures present in hyperspectral images. A widely used family of clustering algorithms is represented by centroid-based clustering methods such as K-means [4], which assumes that similar pixels always form clusters in feature space. When applied to hyperspectral images, these methods can provide satisfactory results but are hampered by their large computational complexity.

In this paper we explore the possibility of using a distributed framework for massive hyperspectral image processing based on cloud computing. We use unsupervised clustering as a case study, focusing on the K-means algorithm to demonstrate the applicability of utilizing cloud computing technologies to efficiently perform distributed parallel processing of hyperspectral data and accelerate computations.

The remainder of the paper is organized as follows. Section II presents the distributed framework design that will be used in our implementation. Section III presents the K-means algorithm and its distributed implementation. Section IV experimentally assesses the proposed method in terms of both accuracy and computational performance. Finally, section IV concludes with some remarks and hints at future research lines.

[1]http://www.enmap.org/

2 Distributed framework design

In order to develop a distributed framework for implementing the K-means algorithm on cloud computing architectures, two main issues need to be addressed: 1) the distributed programming model and 2) the computing engine.

For distributed programming, we resort to the MapReduce model [3], taking full advantage of the high-performance capabilities provided by cloud computing architectures. In this model, a task is processed by two distributed operations: map and reduce. The datasets are organized as key/value pairs, and the map function processes a key/value pair to generate a set of intermediate pairs, dividing a task into several independent subtasks to be run in parallel. The reduce function is in charge of processing all intermediate values associated with the same intermediate key, then collecting all the subtask results to gather the result for the whole task.

Regarding the distributed computing engine, a first solution considered was Apache Hadoop[2] due to its reliability and scalability, as well as its completely open source nature. However, Apache Hadoop only supports simple one-pass computations and is generally not appropriate for iterative algorithms such as K-means. Apache Spark[3] is a newly developed computing engine for large-scale data processing on cloud computing architectures, which implements a fault-tolerant abstraction for in-memory cluster computing, and provides fast and general data processing on large distributed platforms. It not only supports simple one-pass computations, but can also be extended to the case of multi-pass, iterative algorithms.

With the aforementioned issues in mind, the design of our distributed parallel framework for hyperspectral data clustering using Apache Spark is graphically sketched in Fig. 1.

As shown by Fig. 1, the architecture has two main parts:

- The hardware zone: it contains the physical machines that support our virtual machines, which are created in the OpenStack[4] platform, a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.

- The platform zone: the Apache Spark framework in Fig. 1) is installed over a set of Ubuntu Linux virtual machines, created by OpenStack. Our cluster has various types of nodes. The K-means algorithm is designed using the MLib machine learning library [5], and the implementation is embedded into a joint Spark and OpenStack framework, as illustrated graphically in Fig. 2. When we launch an instance of K-means, the master node manages the resources of the cluster and the slaves (workers) perform

[2]http://hadoop.apache.org
[3]http://spark.apache.org/
[4]https://wiki.openstack.org/wiki/Main_Page
[5]https://spark.apache.org/docs/latest/mllib-guide.html

individual tasks on the data. The driver node divides the work into tasks, and the master node coordinates the allocation of tasks with the idea that all the task will be executed in the worker nodes by the executors, following the MapReduce model.

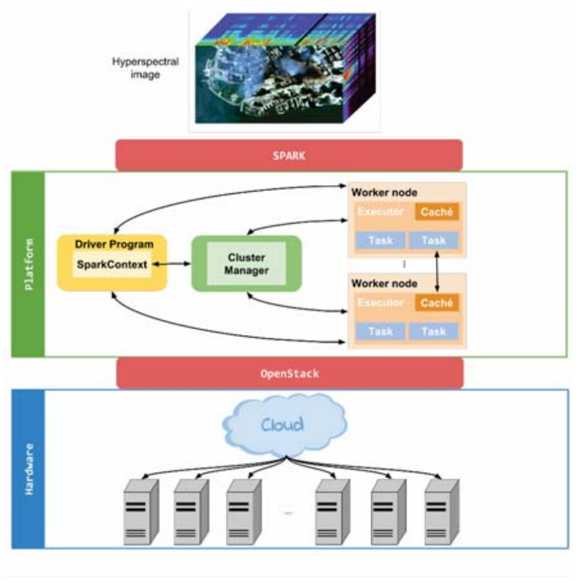


Figure 1: Integrated Apache Spark and OpenStack framework used for the implementation of K-means.

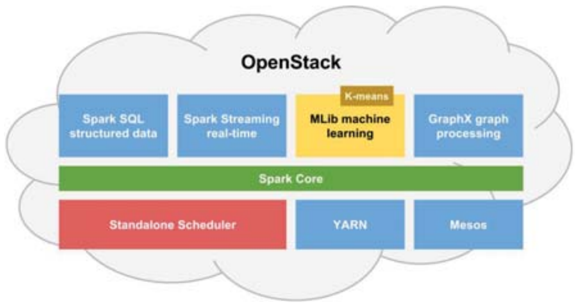


Figure 2: Description of the Apache Spark architecture used in our experiments.

3 The K-means clustering algorithm

K-means is one of the most widely used unsupervised algorithms to group data in a specified number of clusters. The procedure begins with a set of data or observations, $X = [x_1, x_2, ..., x_n]$, in $\mathbb{R}^d$ (so $x_i = [x_{i1}, x_{i2}, ..., x_{id}]$, with $i = 1, 2, ..., n$), that needs to be grouped into a number of *clusters* ($k <= n$). Iteratively, K-means calculates the centers of the k groups, optimizing the error of each group as follows:

$$\min \sum_{j=1}^{k} \sum_{i=1}^{n_k} \parallel x_i^j - C_j, \parallel^2 \tag{1}$$

where $\parallel x_i^j - C_j \parallel^2$ is the distance between a data point x_i^j of the cluster j (n_k is the number of observations within each cluster) and the cluster center C_j.

The K-means algorithm strongly depends on the choice of the initial centers (K-means can converge to a local minimum). So, a proper initialization will result in a final best solution. In order to obtain a set of good initial cluster centers, several methods have been proposed. One of them is the K-means++ method [6]. This algorithm obtains a set of k initial centers which are generally very close to the final solution by the following five steps:

1. An initial point is chosen from the set of samples $X = [x_1, x_2, ..., x_n]$ by an uniform random variable. This point c_1, is the first center.

2. For each sample x_i, the distance, $D(x)$, between x_i and the center, C_1, is calculated.

3. Then, a new candidate to become center is randomly selected using the probability-weighted distribution $\frac{D(x)^2}{\sum_{i=0}^{k} D(x_i)^2}$.

4. Steps 2 and 3 are repeated until k initial centers have been selected.

5. Once the k initial centers have been chosen, we apply the standard K-means algorithm.

4 Parallel/distributed implementation of K-means

4.1 Distributed implementation on Apache Spark

Apache Spark implements a parallel version of the K-means++ method, called *k-means||* [7] using the MapReduce model of computation. K-means|| is very similar to K-means++ with the difference that, instead of choosing a single point after calculating the probability distribution of each of the points in the data set, several clusters are chosen at each iteration. At the end of the parallel algorithm $O(k \log n)$ points are obtained, which are clustered in k centers. This process is illustrated in Algorithm 1.

In order to use the K-means method in Apache Spark, the user must specify the following parameters: number of desired clusters, k; maximum number of iterations that the algorithm can be executed, *maxIterations*; and number of times to execute the K-means algorithm completely, *runs*[6]. The *initializationMode* indicates the type of initialization (if completely random or with K-means|| method. Also, a set of initial centers can be introduced using the *initialModel* option. Finally, the number of steps to be executed during the K-means++ phase, *initializationSteps*, and a pre-defined error threshold for convergence, ϵ, should also be specified.

[6]Since the K-means may not find the optimal overall solution, it is recommended to run it several times to converge to a better final solution.

Algorithm 1 K-means|| algorithm

```
1: procedure K-MEANS||(k, l)    ▷ $k \to$ number of clusters, $l \to \Theta(K)$
2:     $C \leftarrow$ uniform_rand($X$)
3:     $\psi \leftarrow \phi_X(C)$
4:     for $O(\log\psi)$ do
5:         $C' \leftarrow x \in X$ with $p(x) = \frac{l \cdot D(x)^2}{\phi_X(C)}$    ▷ Probability-weighted distribution
6:         $C \leftarrow C \bigcup C'$
7:     end for
8:     For $x \in C$, set $w_x$ to be the number of points in $X$
   closer to $x$ than any other point in $C$
9:     Reclusted ($C$, $k$)
10: end procedure
```

4.2 Paralllel implementation on Scikit-Learn

Scikit is a Python Library for machine learning. This library contains its own k-means++ version, as an option of the *kmeans* class into the *sklearn.cluster* package, allowing the parallel execution of K-means method with the parameter *n_jobs*. The operation is quite similar to Spark, since the user must indicate the number of clusters, *n_clusters*; the number of iterations of one complete execution of the K-means algorithm, *max_iter*; and the number of times the k-means will run with different centers, *n_init*[7]. Much like Apache Spark, Scikit has the option to initialize the first set of centroids randomly, using k-means++ or through an array of data, called *init*. Finally, we must indicate the *tol* or relative increment in the results before declaring convergence, and the number of CPUs that will be used during the execution, *n_jobs*.

5 Experiments

5.1 Hyperspectral datasets

The images used in our experiment were collected by the Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) [2]. The well-known Indian Pines dataset was adquired in 1992 over an agricultural site composed of agricultural fields with regular geometry and with a multiple crops:

1. The first scene has a size of 145x145 pixels, and it was collected over a mixed forest and agriculture area. It has 220 spectral bands in the range from 400 to 2500 nm, with spectral resolution of 10 nm, moderate spatial resolution of 20 nm, and 16 bits

[7]After iterating, the algorithm takes only the best solution reached.

radiometric resolution. After an initial analysis, 8 bands have been removed due to noise, ending up with a total of 212 bands. About half of the pixels in the image (10366 of 21025) contain ground-truth information, which comes in the form of a single label assignment having a total of 16 ground-truth classes.

2. The second scene has a much larger size of 2678x614 pixels. It was collected over the same area, but spanning a much larger extent. It contains 220 spectral bands in the range from 400 to 2500 nm, with spectral resolution of 10 nm, moderate spatial resolution of 20 nm and 16 bits of radiometric resolution. The percentage of pixels with ground truth information is 20.33% (334245 out of 1644292 pixels) and the total number of classes is 58.

5.2 Experimental Configuration

The distributed environment in which we have tested our implementation is the one described in section 2. As mentioned in that section, the cloud computing platform used for experimental evaluation OpenStack. This software is a collection of Open Source technologies that provide a scalable deployment of a cloud computing environment. Our environment is composed of Intel(R) Xeon(R) CPUs E5430 @ 2.66GHz (8 cores), 16 GB RAM, Shared storage, NetApp FAS3140.

The virtual nodes that we use over the hardware specified have two virtual CPUs, 4GB of RAM and 40 GB hard disk each. In addition, we have developed a parallel version of the algorithm for comparative purposes. This version has been implemented on a paltform with Intel(R) Core(TM) i7-4790 CPUs @ 3.60GHz (8 cores), 16 GB RAM, SanDisk SDSSDA240G.

In our experiments, we used Java 1.8.0_92-b14, Ubuntu 14.04 x64 LTS as operating system, Python 2.7.10, Scikit Learn 0.14.1 version and the newest (under-development) Apache Spark 2.11.

5.3 Description of experiments

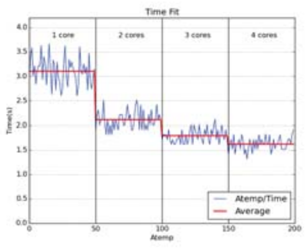
5.3.1 Single vs multiple cores

In a first experiment, we considered the small Indian Pines image and launched 50 executions of the algorithm changing the number of cores between 1 and 4. In this test, we set the tolerance threshold to 1e-15, we performed 10 iterations with different centroid seeds, and used k-means++ to select initial cluster centers. The obtained results are summarized in Table 1, which reports on the learning fit, the prediction accuracy, the clustering accuracy and the reliability (average and standard deviation). As shown in this table, the values remain constant as we increase the number of cores except the trainning of the model. The fit time is reduced until 3 cores are used (see Fig. 3a). At that point, there is no difference to add more workers because the algorithm no longer is able to parallelize more information and therefore the speedup remains constant.

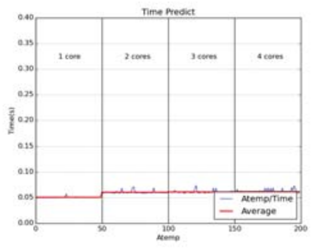
To validate the results obtained from K-means++, we used a confusion matrix [8] which is graphically represented in Fig. 3c. The confusion matrix indicates the agreement between the ground-truth classes and the clusters identified in the process, which appears to indicate a good fit between each ground-truth class and at least one of the identified clusters. Finally, the clusters obtained by the K-means++ algorithm for the small Indian Pines image are shown in Fig. 4. The figure shows the clusters with the background of the image removed (i.e., those pixels that do not have associated ground-truth) and also without removing the background.

Cores	**AVG Fit**	**Std Fit**	**AVG Predict**	**Std Predict**	**AVG Accuracy**	**Std Accuracy**	**AVG Reliability**	**Std Reliability**
1	3.1055	0.2826	0.0508	0.0010	0.5446	0.0019	0.4770	0.0058
2	2.1135	0.1846	0.0606	0.0028	0.5456	0.0029	0.4772	0.0067
3	1.7955	0.1387	0.0612	0.0028	0.5452	0.0022	0.4770	0.0059
4	1.6213	0.1282	0.0620	0.0037	0.5453	0.0017	0.4779	0.0054

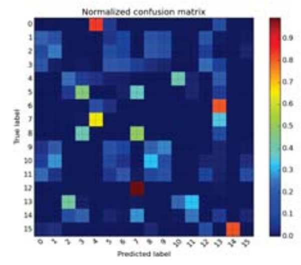
Table 1: Summary of the execution of K-means in multiple cores with the small Indian Pines image.



(a) Time to training

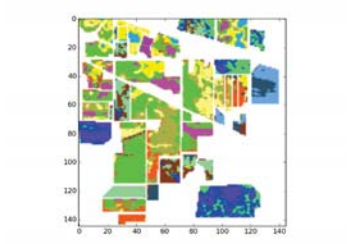


(b) Time to clustering

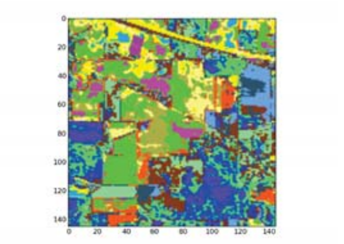


(c) Confusion matrix

Figure 3: Graphical representation of the time to train, time to clustering and the confusion matrix obtained after applying the K-means algorithm to the small Indian Pines image



(a) Without background



(b) With background

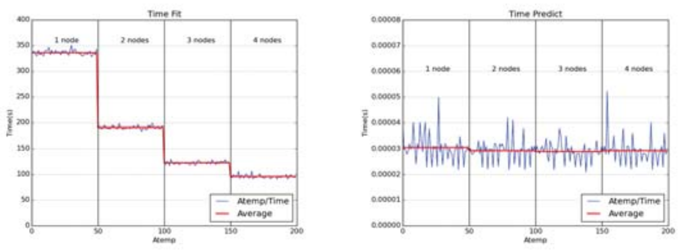
Figure 4: Clustering results obtained with the small Indian Pines image

5.3.2 Single vs multiple nodes

In this experiment, we launch 50 executions with the large Indian Pines image changing the number of slave nodes to 1, 2, 3 and 4. We select a tolerance threshold of 1e-15 and 10 run with different centroid seeds and using the k-means|| algorithm. Here, we use the large Indian Pines data set. The results obtained are shown in Table 2. In the table, we can observe how the values remain constant as we increase the number of nodes (except the trainning model). The fit time is reduced exponentialy (see Fig. 5a). Moreover, to validate the results obtained from K-Means, we show the confusion matrix for the large Indian Pines image in Fig. 6a. The final clusters obtained by the algorithm are shown in Fig. 7.

Nodes	AVG Fit	Std Fit	AVG Predict	Std Predict	AVG Accuracy	Std Accuracy	AVG Reliability	Std Reliability
1	336.0076	4.5684	0.0000305	0.00000522	0.4118	0.0132	0.2547	0.0043
2	191.1449	3.0629	0.0000293	0.00000415	0.4107	0.0153	0.2543	0.0046
3	122.8018	2.8471	0.0000289	0.00000335	0.4149	0.0114	0.2542	0.0050
4	95.9846	3.2338	0.0000293	0.00000502	0.4127	0.0120	0.2536	0.0048

Table 2: Summary of the execution of K-means in multiple nodes with the large Indian Pines image.



(a) Time to training (b) Time to clustering

Figure 5: Confusion matrix obtained after applying the K-means algorithm to the large Indian Pines image.

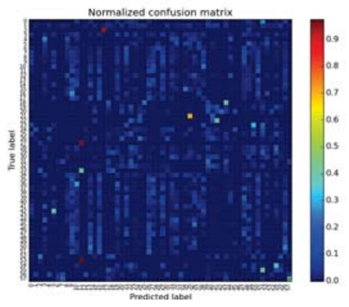


Figure 6: Graphical representation of the time to train and time to clustering after applying the K-means algorithm to the large Indian Pines image.

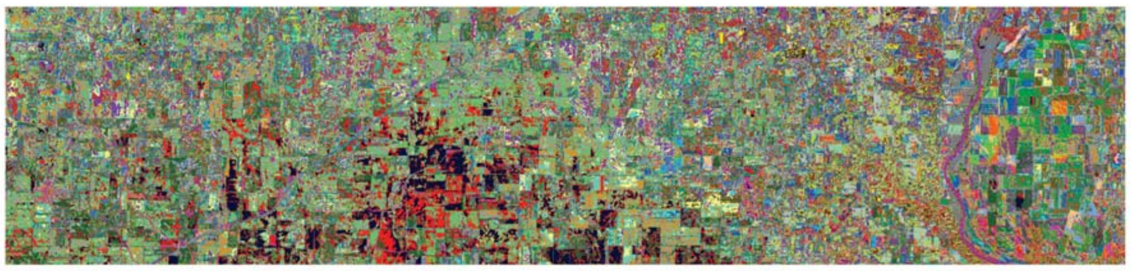


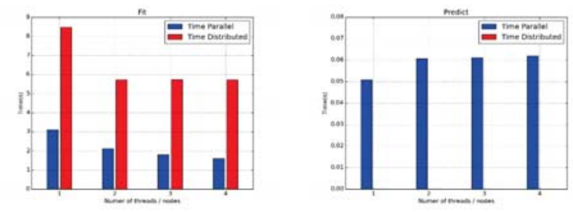
Figure 7: Clustering results obtained with the large Indian Pines image.

5.4 Parallel vs distributed

Finally we compare the parallel and distributed implementations of K-means. Table 3 shows the timing average (in seconds) measured after the parallel/distributed execution of K-means using both images and multiple cores/nodes. Figs. 8 and 9 respectively provide a graphical representation of the times for fitting and predicting for the parallel and distributed version with both images. Finally, we show in Fig. 10 a graphical representation of speedup evolution in all cases.

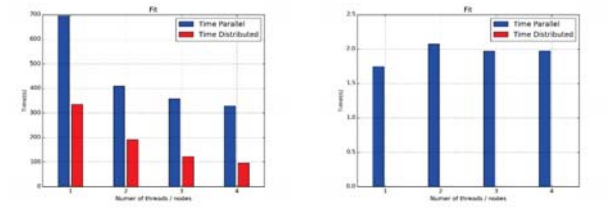
Image used	small Indian Pines image				large Indian Pines image			
Cores or nodes	**Par Fit**	**Par Predict**	**Dist Fit**	**Dist Predict**	**Par Fit**	**Par Predict**	**Dist Fit**	**Dist Predict**
1	3.1055	0.0508	8.4701	0.0000279	694.8973	1.7438	336.0076	0.0000305
2	2.1135	0.0606	5.7111	0.0000278	409.5856	2.0710	191.1449	0.0000293
3	1.7955	0.0612	5.7370	0.0000289	357.4873	1.9692	122.8018	0.0000289
4	1.6213	0.0620	5.7213	0.0000287	327.6066	1.9747	95.9846	0.0000293

Table 3: Timing average (in seconds) measured after the parallel/distributed execution of K-means using both images and multiple cores/nodes (Par refers to the parallel version and Dist refers to the distributed version).



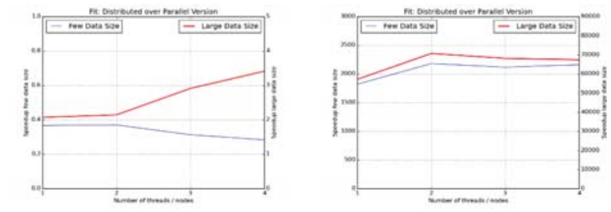
(a) Fitting (b) Predicting

Figure 8: Graphical representation of the times for fitting and predicting with the small Indian Pines image for the parallel and distributed implementation of K-means.



(a) Fitting (b) Predicting

Figure 9: Graphical representation of the times for fitting and predicting with the large Indian Pines image for the parallel and distributed implementation of K-means.



(a) Speedup (fitting) (b) Speedup (predicting)

Figure 10: Graphical representation of the speedup for fitting and predicting with the Indian Pines images for the parallel and distributed implementation of K-means.

6 Conclusions and future lines

In this paper, we have discussed the possibility of exploiting cloud computing architectures for hyperspectral image processing. As a case study, we have presented a cloud computing implementation of the K-means algorithm on Spark platform. Clustering has the advantage that it can be performed in unsupervised fashion. Our experimental results show the effectiveness of the proposed distributed implementation, not only in terms of clustering accuracy but also in terms of computational performance. As future work, we will implement other algorithms for hyperspectral data processing.

Acknowledgements

This work has been supported by Junta de Extremadura (decreto 297/2014, ayudas para la realización de actividades de investigación y desarrollo tecnológico, de divulgación y de transferencia de conocimiento por los Grupos de Investigación de Extremadura, Ref. GR15005). This work was supported by the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional

Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

References

[1] Antonio Plaza, Javier Plaza, Abel Paz, and Sergio Sanchez, "Parallel Hyperspectral Image and Signal Processing," *IEEE Signal Processing Magazine*, vol. 28, pp. 196–218, 2011.

[2] Robert O. Green, Michael L. Eastwood, Charles M. Sarture, Thomas G. Chrien, Michael Aronsson, Bruce J. Chippendale, Jessica A. Faust, Betina E. Pavri, Christopher J. Chovit, Manuel Solis, Martin R. Olah, and Orlesa Williams, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment*, vol. 65, pp. 227–248, 1998.

[3] Zebin Wu, Yonglong Li, Antonio Plaza, Jun Li, Fu Xiao, and Zhihui Wei, "Parallel and Distributed Dimensionality Reduction of Hyperspectral Data on Cloud Computing Architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, no. 99, pp. 1–9, 2016.

[4] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, "An Efficient k-Means Clustering Algorithm:Analysis and Implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881 – 892, 2002.

[5] Antonio Plaza, Javier Plaza, Gabriel Martin, and Sergio Sanchez, "Hyperspectral Data Processing Algorithms," in *Hyperspectral Remote Sensing of Vegetation*, Alfredo Huete Prasad S. Thenkabail, John G. Lyon, Ed., chapter 5, pp. 121–137. Taylor and Francis, Abingdon, United Kingdom, 2011.

[6] David Arthur and Sergei Vassilvitskii, "K-means++: The Advantages of Careful Seeding," in *Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, Ed., New Orleans, Louisiana, 2007, pp. 1027–1035, Society for Industrial and Applied Mathematics.

[7] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii, "Scalable K-means++," *Proceedings of the VLDB Endowment (PVLDB)*, vol. 5, no. 7, pp. 622–633, 2012.

[8] Stephen V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77–89, 10 1997.