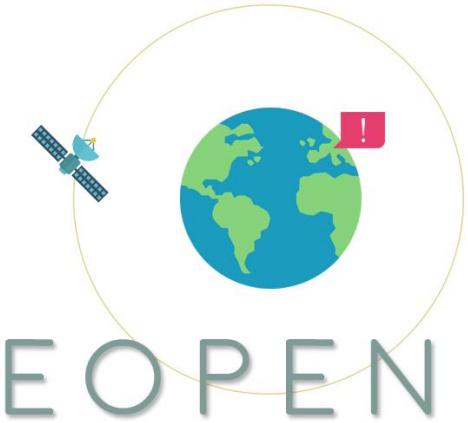


EOPEN

opEn interOperable Platform for unified access and analysis of Earth observatioN data

H2020-776019

D4.2 Report on data clustering of EO and non-EO data

Dissemination level:	Public
Date of delivery:	Month 25, 03.06.2020 (resubmission)
Workpackage:	WP4 Knowledge discovery and content extraction
Task:	T4.4 Data clustering and exchange among federated databases
Type:	Demonstrator
Approval Status:	Approved
Version:	2.0
Number of pages:	52
Filename:	d4.2-clustering_of_eo_and_non_eo_data_2020-06-03_v2.0.docx

Abstract

This deliverable reports on the current status and latest advances made with respect to work package WP4, specifically, T4.4 about the data clustering and exchange among federated databases within EOPEN. This task is closely linked to Task 4.1, Task 4.5, as well as tasks in work package WP6 (System development and integration) and work package WP3 (EO and non-EO data acquisition). The development and implementation of clustering algorithms described in this deliverable enable the co-clustering of both EO and non-EO data. The key contributions of this deliverable are: (a) a methodology to estimate the number of clusters; (b) a description of Twitter information clustering, both textual and imagery; (c) a description of pixel clustering for image segmentation; (d) a novel methodology for clustering satellite image patches in an unsupervised way; (e) the status of the implementation on HPDA and HPC platform; and (f) the status of the integration on the EOPEN platform.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 776019

History

Version	Date	Reason	Revised by	Approved By
0.1	22.11.2019	Initial draft and document structure	Li Zhong, Stelios Andreadis	Dennis Hoppe
0.2	25.11.2019	Integration of NOA's contribution	Li Zhong, Vasileios Sitokonstantinou	Dennis Hoppe
0.3	29.11.2019	Revise context based on review comments	Li Zhong, Vasileios Sitokonstantinou, Stefanos Vrochidis	Dennis Hoppe
0.4	29.11.2019	Revise context based on review comments	Stelios Andreadis, Gilles Lavaux (Serco)	Li Zhong
1.0 (Final Version)	29.11.2019	Final check of the document	Li Zhong	M. Gabriella Scarpino
2.0 (revised version)	31.05.2020	Revised document with final comments	Stelios Andreadis, Ilias Gialampoukidis, Li Zhong	Dennis Hoppe, M. Gabriella Scarpino

Author list

Organization	Name	Contact Information
USTUTT	Li Zhong	li.zhong@hlrs.de
USTUTT	Dennis Hoppe	dennis.hoppe@hlrs.de
CERTH	Stelios Andreadis	andreadisst@iti.gr
CERTH	Ilias Gialampoukidis	heliasgj@iti.gr
NOA	Vasileios Sitokonstantinou	vsito@noa.gr
CERTH	Stefanos Vrochidis	stefanos@iti.gr

Executive Summary

In this deliverable, we report on our key contributions towards the clustering of EO and non-EO data (both textual and imagery) as well as the integration on HPC/HPDA and EOPEN platform. In order to achieve this ambitious goal, an overview of state-of-the-art algorithms on clustering is described, a methodology to estimate the number of clusters is introduced, and the implementation of a clustering service is presented.

Key contributions and achievements with respect to Task T4.4 are:

- An overview of state-of-the-art-clustering algorithms and a novel approach to estimate the number of clusters, presented in Section 2.

- Existing or new algorithms, developed in the frame of EOPEN, to deal with the problem of clustering EO and non-EO data. More information will be given in Section 3 (clustering of non-EO data) and Section 4 (clustering of EO data).

- Algorithms transferred on Hadoop and executed on HLRS's Big Data infrastructure, aiming at linear computational and memory complexity. Results and an evaluation of process time and cost will be shown in Section 5.1 .

- Integration of a clustering service into the EOPEN platform, visualisation of the results in the EOPEN Dashboard, and an outline of the upcoming workflow, all depicted in Section 5.2 .

It should be noted here that this document is a resubmitted version of D4.2 and includes all the progress achieved for the Task T4.4 since the initial submission of the deliverable. Specifically, newly added Sections 4.2 and 5.1.4 refer to a new approach to cluster EO imagery patches in an unsupervised way and its parallel implementation on HPC. Any remaining work on the task will be reported in the final deliverable of WP4, D4.4.

Taking into strong consideration the reviewer's suggestion to get informed about the work of other projects on similar tasks and seek synergies, we have initiated communications with the projects CANDELA and BETTER. Regarding CANDELA, we worked together to prepare comparative tables, which show the similarities and differences of each project's approach and can serve as a first step towards discovering potential collaborations. However, these tables are not included in this deliverable, because they refer to tasks other than T4.4. In detail, a table about information retrieval (DLR) will be added to D4.3, a table about semantic technologies (IRIT) to D5.2, and a table about change detection (TAS FR) to resubmitted D4.1. Regarding BETTER, we have already started two collaborative exercises with Fraunhofer and DEIMOS that involve EOPEN's work on social media collection and analysis. Again, these exercises are not reported in D4.2, but will be in the relevant D3.3.

Abbreviations and Acronyms

ANN	Artificial Neural Network
API	Application Programming Interface
CBIR	Content-Based Image Retrieval
CERTH	The Centre for Research & Technology, Hellas
DL	Deep Learning
DNN	Deep Neural Network
EO	Earth Observation
HDFS	Hadoop Distributed File System
HLRS	High-Performance Computing Center Stuttgart
HPC	High-Performance Computing
HPDA	High-Performance Data Analysis
KLD	Kullback–Leibler Divergence
LDA	Latent Dirichlet Allocation
MSE	Mean Square Error
ML	Machine Learning
NOA	National Observatory of Athens
PCA	Principal Component Analysis
RF	Random Forest
SoA	State of the Art
t-SNE	t-distributed Stochastic Neighbour Embedding
USTUTT	University of Stuttgart

Table of Contents

1 INTRODUCTION

Work package WP4 addresses the challenge of extracting information from EO and non-EO data collected in WP3. Specifically, the core objective is to detect changes, concepts, events and communities in large streams of data. In order to achieve this challenging goal, the work package is split into multiple tasks covering change detection (Task 4.1), concept and event detection in non-EO data (Task 4.2), similarity fusion (Task 4.3) and community detection (Task 4.5).

This document specifically describes the status and progress made with respect to Task T4.4, whose objective is to develop and apply clustering algorithms on EO and non-EO data in EOPEN. Specifically, the core work is to enable accurate and intuitive services that could be accessed by end users. In order to achieve this ambitious target, a lot of work has been done. First, the problem of estimating the number of clusters (Romesburg C., 2004) is defined and a novel approach that is independent of the type of input to be clustered, namely the DBSCAN-Martingale, is presented. Next, focusing on non-EO data, two clustering procedures are described; one performed on textual information and one on imagery, both originating from Twitter posts. For the textual data, a dedicated service uses the aforementioned approach to estimate the number of clusters, then assigns the available posts to the clusters and finds the most frequent terms for each group. Aside from this, an unsupervised image clustering is performed on the images crawled from Twitter. Different techniques including deep neural networks (DNNs) such as AutoEncoder are utilized. Regarding the EO data, due to the data imbalance and insufficiency, an algorithm based on clustering was implemented to automatically identify the rice pads from EO images in an unsupervised manner, so that the great labour coming from manually labelling the data is eliminated. Furthermore, the implementation of algorithms is transferred to HPC and HPDA infrastructure utilizing Apache Spark, TensorFlow, Keras and so on to achieve high efficiency. Finally, the text clustering service is integrated in EOPEN platform, its results are visualised in the EOPEN Dashboard and its transferability to HPC is evaluated.

The remainder of this deliverable is organized as follows. Firstly, in Section 2, we describe the problem of estimating the number of clusters and present the novel method of DBSCAN-Martingale. Section 3 provides insight on the algorithms and methods adopted to cluster non-EO data, especially on Twitter textual and imagery data, while also presenting the corresponding results. For the clustering of EO data, methods that are used to cluster pixels of satellite images are described in Section 4 along with an analysis of their drawbacks and further improvement. Section 5 continues with the implementation of clustering strategies on HPDA and HPC infrastructure provided by HLRS. Besides, the status of integration and visualisation of clustering on EOPEN platform is also given. The deliverable concludes with Section 6, which summarizes the progress made and gives a brief outlook towards further 15.

2 ESTIMATING THE NUMBER OF CLUSTERS

The EOPEN datasets involve Big Data collections of EO and non-EO data that need to be grouped in order to be effectively managed. All these data sources are initially collected as raw data and the problem is to cluster them per a target modality, such as visual, textual, spatiotemporal or combinations of multiple modalities.

Clustering often requires as input the desired number of clusters and this is a challenging pattern recognition problem in several real-life applications, such as in bioinformatics (Getz, et al. 2000), in computer vision (Liu, et al. 2017) (Moumtzidou, et al. 2017), in sociology and biology (Girvan, et al. 2002) and in computer science applications (Newman, et al. 2004).

Assuming that each modality can be represented in a feature space of one multi-dimensional vector per item (a tweet or satellite image or text or Twitter image), we present the EOPEN approach in estimating the number of clusters in a dataset of n-instances (vectors).

2.1 Related Work

Over the last two decades clustering using density-based approaches has been a notable open pattern recognition problem. (Ester, et al. 1996) has introduced DBSCAN which has then been enhanced into a modern hierarchical version, namely HDBSCAN by (Campello, et al. 2013), which produces cluster hierarchies. Later, (Dockhorn, et al. 2016) introduced a non-hierarchical cut of the DBSCAN hierarchy in a deterministic but variable density-based clustering. Meanwhile, (Gialampoukidis, et al. 2016) proposed a probabilistic density-based clustering having variable density levels that are randomly generated from the uniform distribution. (Gialampoukidis, et al. 2017) proposed the generation of skewed samples of density levels, aiming to reduce the time needed to extract all clusters. Contrary to (Gialampoukidis, et al. 2017), the methodology presented here analytically obtains the expected number of extracted clusters per DBSCAN-realization and the minimum required number of iterations for the extraction of all clusters.

F-OPTICS, presented by (Schneider and Vlachos 2013), has reduced the computational cost of the OPTICS algorithm, originally introduced in (Ankerst, et al. 1999), using a probabilistic definition of the reachability distance, without significant accuracy reduction. The OPTICS-ξ algorithm (Ankerst, et al. 1999) requires an extra parameter ξ, which has to be manually set in order to find “dents” in the OPTICS reachability plot. (Gan and Tao 2015) introduced an approximate version of DBSCAN, which could be combined with the DBSCAN-Martingale approach that is proposed in this section by replacing DBSCAN with its corresponding approximate clustering algorithm. NQ-DBSCAN is a recent algorithm for density-based clustering, (Chen, et al. 2018), which aims to reduce the number of unnecessary distance computations in order to provide a faster version of DBSCAN, without significant deviation from the exact output of the original DBSCAN algorithm. In NQ-DBSCAN the selection of the parameters ($\in$and$minPts$) is considered optimal and a priori given. (Schubert, et al. 2017) considered the estimation of the density level $\in$ more challenging than the intuitive task to determine $minPts$.

Recent sequential density-based clustering approaches involve several iterations to obtain the final cluster structure (Louhichi, et al. 2014) (Mai, et al. 2017). In contrast to these approaches, the following analysis is based on recent advances in statistical mechanics

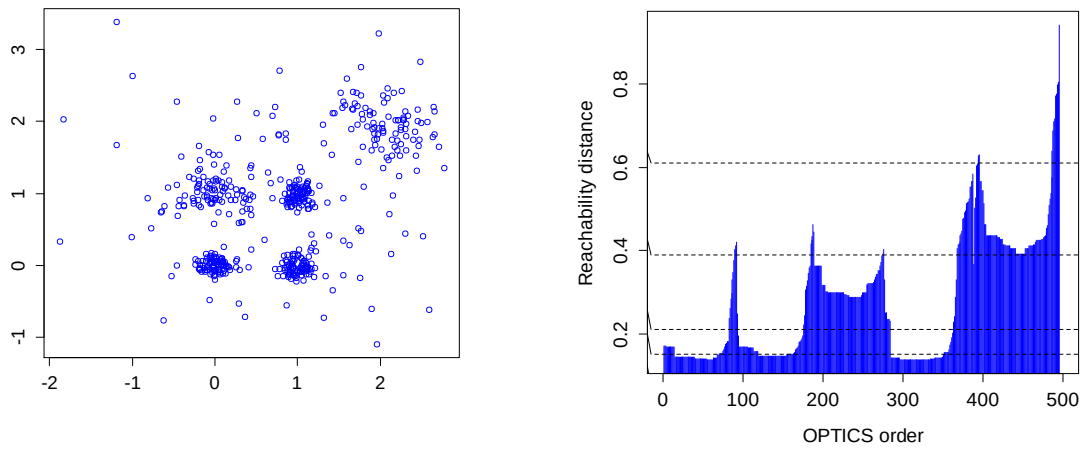
(Gialampoukidis, et al. 2013) (Antoniou, et al. 2015) and martingale theory, and optimizes two martingale stochastic processes in density-based clustering, originally introduced in (Gialampoukidis, et al. 2016).

2.2 Background and Notation in Density-based Clustering on the Feature Space

Given a dataset of n-instances, density-based clustering algorithms, such as DBSCAN (Ester, et al. 1996), provide as output a clustering vector C with values the cluster IDs $C[j]$ for each instance $j = 1,2, \dots, n$, assigning each element j to a cluster. The cluster may refer to a topic of text documents (tweets in our case) or groups of images (social media images or satellite ones), or even communities of user accounts that usually mention each other on Twitter.

In case the j-th element is marked as noise, then the cluster ID is zero: $C[j] = 0$, where we denote by $C[j]$ the j-th element of a vector C. This is also written as $C = 0$, where 0 is a vector of zeros. The algorithm DBSCAN has two parameters: a density level ϵ and a lower bound for the number of clusters in a dataset $minPts$. The parameter $minPts$ is usually predefined based on the minimum size of the desired clusters.

The density level ϵ is hard to be estimated and, if so, then the algorithm is not able to output all clusters using one unique density level. The cluster structure is visualised using the OPTICS (Ankerst, et al. 1999) diagram of reachability distances, where the dents represent clusters and it is also possible to observe the density level at which the desired clusters are extracted. An example of some randomly generated clusters on the plane along with the corresponding OPTICS plot of reachability distances is presented in Figure 1 below.



(a) 5 random clusters of various densities

(b) Reachability distance plot generated by OPTICS

Figure 1: Illustrative example of the problem statement in density-based clustering.

The reachability distance determines the density level ϵ, while the parameter $minPts$ is a pre-defined fixed value, approximately equal to 10, as initially proposed by (Ankerst, et al. 1999). For each density level ϵ, the output of DBSCAN is one clustering vector and is denoted by $C_{DBSCAN(\epsilon)}$.

Small values of ϵ result to $C_{DBSCAN(\epsilon)} = 0$, because all points are marked as noise. However, large values of ϵ, result to $C_{DBSCAN(\epsilon)} = 1$, where 1 is a vector of ones, since all points are reachable from any other point, hence, all points are assigned to the same cluster.

Lemma 1. (Gialampoukidis, et al. 2016) Two clustering vectors C_i, C_l are mutually orthogonal, if and only if they contain different clusters.

The proof of Lemma 1 is given in (Gialampoukidis, et al. 2016).

The DBSCAN-Martingale algorithm (Gialampoukidis, et al. 2016) generates a probability distribution after a certain number of realisations of the stochastic process. For example, in the case of the dataset given in Figure 1 above, the generated probability distribution is given in Figure 2 below.

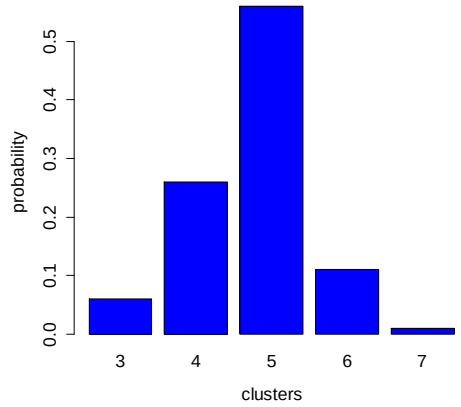


Figure 2: Outcome of the DBSCAN-Martingale process after 100 realisations.

However, the existing version of the DBSCAN-Martingale algorithm[1] is not scalable to Big Data problems. Since it is written in R language, by default it runs only on a single thread on the CPU. Our purpose, within EOPEN project, is to optimise it in order to be executed on High-Performance Computing (HPC) infrastructures with as less resource allocation as possible.

In the following, we present the Time Operator associated with the DBSCAN-Martingale process, along with the Age estimations. The Time Operator theory has been applied to Stochastic Processes like Martingales, aiming to quantify the average innovation time, i.e. the exact amount of (clock) time that a process takes in generating unpredicted and unexpected knowledge. In our case, it is necessary to minimise the processing time of Big Data problems, executed on multiple processing nodes and cores, such as the clustering of EO and non-EO data.

2.3 Time Operator and Age of the DBSCAN-Martingale processes

Let$X$ and Y_1, Y_2, be any random variables having finite expectations $E[|X|] < \infty$. Then if X_t is defined by $X_t = E[X|Y_1, Y_2, \dots, Y_t]$, the sequence of $X_t, t = 1,2$, is a martingale, known as Doob's Martingale.

[1] https://github.com/MKLab-ITI/topic-detection/blob/master/DBSCAN_Martingale.r

The DBSCAN-Martingale generates first a random sample of uniformly distributed random numbers $\epsilon_t, t = 1,2,\dots,T$ in $[0, \epsilon_{max}]$. The sample of $\epsilon_t, t = 1,2,\dots,T$ is sorted in increasing order and for each density level ϵ_t a clustering vector is provided by DBSCAN, denoted by $C_{DBSCAN(\epsilon_t)}$.

Let F_t be the σ-algebra generated by $\\\\\{C_{DBSCAN(\epsilon_1)}, C_{DBSCAN(\epsilon_2)}, \dots, C_{DBSCAN(\epsilon_t)}\\\\\}$ and let also F_0 be the trivial σ-algebra $\\\\\{\Omega, \emptyset\\\\\}$ at stage $t = 0$.

The passage from F_0 to F_1 keeps all clusters found by DBSCAN at stage $t = 1$, defining the clustering vector $C^{(1)} \coloneqq C_{DBSCAN(\epsilon_1)}$, corresponding to the lowest density level ϵ_1.

At stage, $t = 2$, some of the detected clusters by $C_{DBSCAN(\epsilon_2)}$ are new and some of them have also been extracted at stage $t = 1$. DBSCAN-Martingale keeps only the newly detected clusters of the second stage, $t = 2$, by taking only groups of points of the same cluster ID with size greater than $minPts$:

$$C^{(t)}[j] \coloneqq \begin{cases} 0 if point j belongs \\ a previous cluster C_{DBSCAN(\epsilon_t)}[j] otherwise \end{cases} \quad (1)$$

where $C^{(1)} = C_{DBSCAN(\epsilon_1)}$. The phrase "if point j belongs to a previous cluster" means that point j has been already identified as being part of a cluster before, i.e. at a lower density level ϵ (or larger $minPts$). By "newly detected clusters" we refer to groups of points which were previously regarded as noise at a lower density level ϵ and now they formulate one cluster. Observe that each vector $C^{(t)}, t = 1,2,\dots,T$ has only the newly extracted clusters and all other points are marked as noise. The number of clusters is also a random variable, obtained by the maximum cluster ID observed in C.

Theorem 1. The Age η of the clustering vector C is given by:

$$\eta = \sum_{t=1}^{T} t \frac{|C^{(t)}|^2}{|C|^2} \quad (2)$$

Proof. Let H_t be the Hilbert space spanned by conditioning $E_t = E[.|F_t]$ to our knowledge up to stage t, as provided by the σ-algebras $F_t, t = 1,2,\dots,T$. All clustering vectors $C^{(t)}, t = 0,1,\dots,T$ are, by construction, mutually orthogonal since they contain different clusters, projecting the outcome of the final clustering vector C to the innovation spaces $N_t = H_t \ominus H_{t-1}$, which are spanned by $C^{(t)}$. The final clustering vector C is uniquely decomposable as the direct sum of innovations composing the space of fluctuations $H = N_1 \oplus N_2 \oplus \cdots N_T$:

$$C = C^{(1)} \oplus C^{(2)} \oplus \dots \oplus C^{(T)} \quad (3)$$

The projections $E_t C = E[C|F_t], t = 1,2,\dots,T$ are the final outcome of the clustering vector C which needs to be determined so as to extract all available clusters of various densities.

The projections onto the innovation spaces N_t are defined as:

$$P_t C = C^{(t)} = E[C|F_t] \ominus E[C|F_{t-1}] = (E_t \ominus E_{t-1})C \quad (4)$$

Our knowledge about the final clustering vector C up to stage t is restricted to $E_t C$ and finally, at stage $t = T$, we have gained all available knowledge about the vector C.

The Time Operator, as defined in (Gialampoukidis, et al. 2013), acts on the clustering vector C in H, defining also the internal Age through the corresponding innovation probabilities:

$$p_t = Prob\{\{C \in N_t\}\} = \frac{|P_t C|^2}{|C - E[C]|^2} = \frac{|C^{(t)}|^2}{|C|^2} \quad (5)$$

where $E[C] = 0$ because at the beginning of the process the clustering vector C is a vector of zeros and there are no expected clusters without any application of the DBSCAN algorithm.

Figure 3 illustrates the innovation process of an exemplary sequential extraction of three clusters. The projections $P_t, t = 1,2$ of the final clustering vector C onto each innovation space $N_t, t = 1,2$ keep only the newly detected clusters at each stage.

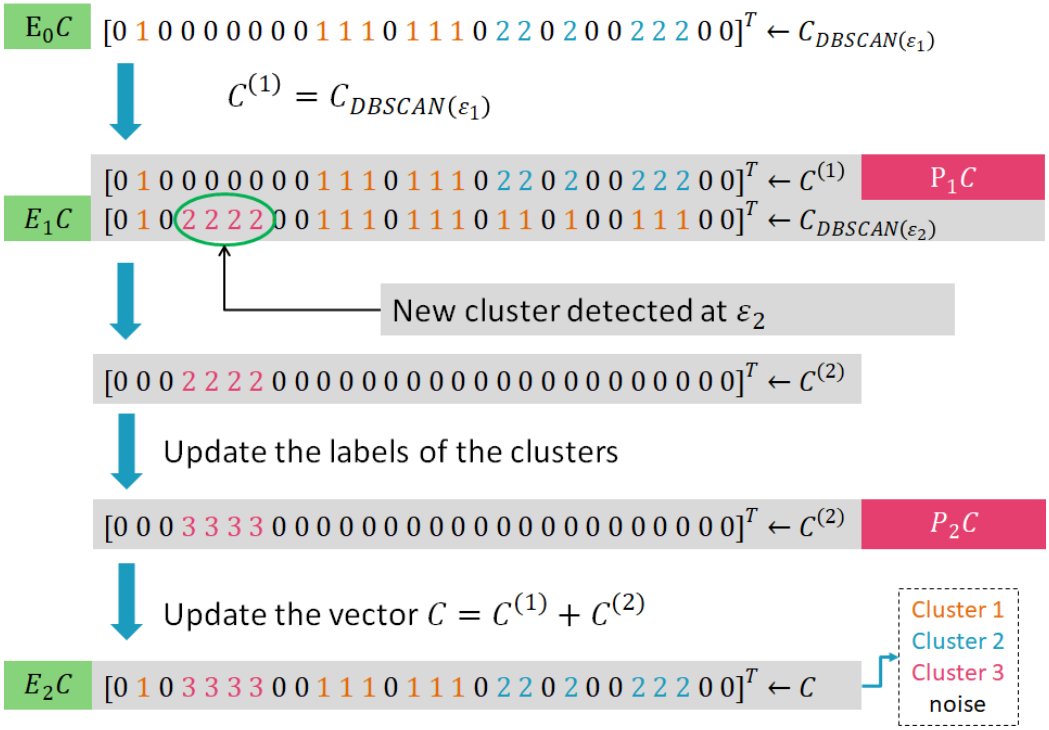


Figure 3: An example where two different density layersε_1 andε_2 are used to generate two different outputs of DBSCAN and are then combined into one single result.

Theorem 2: The probability to extract exactly κ clusters at one DBSCAN-realisation is

$$Prob(Y = \kappa) = \sum_{A \in A_\kappa} \prod_{i \in A} \delta(C_i) \prod_{j \in \acute{A}} \left(1 - \delta(C_i)\right)$$

where A_κ is the set of all subsets of κ integers that can be selected from $1,2,\dots,k$, $\acute{A}$ is the complement of A, the cluster C_i has maximum reachability distance $d_{max}(C_i)$, $d_{min}(C_i) = min_{x_j \in C_i} d_{min}(x_j, C_i)$, $d_{min}(x_j, C_i)$ is the minimum reachability distance at which document x_j no longer belongs to cluster C_i, and

$$\delta(C_i) = \frac{d_{max}(C_i) - d_{min}(C_i)}{\epsilon_{max}}$$

The proof of Theorem 2 is given in (Gialampoukidis, et al. 2019).

Remark 1: The expected number of extracted clusters per DBSCAN-Martingale realisation is:

$$E[Y] = \sum_{\kappa=0}^{k} \kappa \cdot Prob(Y = \kappa)$$

where $Prob(Y = \kappa)$ is given by Theorem 2.

Algorithm 1: DBSCAN-Martingale($minPts, \epsilon_{max}$) **return** $\hat{k}$
1: Generate a random sample of $\lceil \eta \rceil$ values in $[0, \epsilon_{max}]$
2: Sort the generated sample $\epsilon_t, t = 1,2, \dots, \lceil \eta \rceil$
3: **for** $t = 1$ to $\lceil \eta \rceil$
4: compute $C_{DBSCAN(\epsilon_t)}$
5: compute $C^{(t)}$
6: update the cluster IDs
7: update the vector C
8: update $\hat{k} = \max_j C[j]$
9: end **for**
10: **return** $\hat{k}$

We denote by $\lceil \omega \rceil$ the ceiling function of the number ω and we propose the use of the closest larger integer to η as the necessary number of iterations to extract all clusters. We also note that Step 4 allows direct parallelisation by sharing the job in $\lceil \eta \rceil$ processing cores, i.e. instances in a cloud computing service.

The presented DBSCAN-Martingale algorithm requires T iterations of the DBSCAN algorithm, which runs in $O(nlogn)$ if a tree-based spatial index can be used and in $O(n^2)$ without tree-based spatial indexing (Ankerst, et al. 1999), or in $O(n)$ in the case of an approximate DBSCAN algorithm is employed (Chen, et al. 2018). Hence, DBSCAN-Martingale runs in $O(Tnlogn)$ for tree-based indexed datasets, in $O(Tn^2)$ without tree-based indexing and in $O(Tn)$ in the case of an approximate DBSCAN is adopted per iteration. However, in EOPEN project, each one of the T iterations and number of realisations are allocated in multiple nodes and cores so the algorithm can scale up using an HPC infrastructure. The implementation of DBSCAN-Martingale on HPC is presented in Section 5.1 .

3 CLUSTERING OF NON-EO DATA

3.1 Twitter Textual Data Clustering

The undiminished popularity of social media, and especially of the well-known platform of Twitter, results in a very large amount of posts on a daily basis, making it challenging to identify interesting topics and events that are covered by the users. Text clustering aims at grouping together documents or social media posts (tweets, in the frame of EOPEN) that discuss the same topic. Topics should be considered as general categories, e.g. weather, politics, industry, but rather as particular thematic areas and trending topics that are continuously updated.

Topic detection assumes a vector representation of a text document and is usually considered as a clustering problem (Aggarwal and Zhai 2012), in absence of training sets. In the case of text clustering, Latent Dirichlet Allocation (LDA) (Blei, et al. 2003) has been a model that performs very well but requires the number of topics (i.e. clusters) to be given as input. There are density based approaches (Ester, et al. 1996) (Ankerst, et al. 1999) (Campello, et al. 2013) that do not require a priori knowledge of the number of clusters (i.e. topics), but they are less effective than LDA. In addition, LDA has been generalised to nonparametric Bayesian approaches, such as the hierarchical Dirichlet process (Teh, et al. 2004) and DP-means (Kulis and Jordan 2004).

Other topic detection approaches in the social media domain involve graph-based approaches (Petkos, et al. 2017), where a graph clustering algorithm is applied on the graph of text documents and the decision, whether to link two social media posts or not, is based on the output of a classifier, which assigns or not the candidate items in the same cluster.

Contrary to the above graph-based approach, the solution adopted in EOPEN runs in an unsupervised way, using the DBSCAN-Martingale to estimate the number of clusters, as described in Section 2, and then LDA to assign posts to topics. Experiments on running this method in a parallel manner are reported in Section 5.1.3 , while its implementation and integration in the EOPEN system are presented in Section 5.2 .

3.2 Twitter Imagery Data Clustering

In recent years there has been a growing interest in developing effective methods for content-based image retrieval (CBIR) (Zheng, X., et.al. 2004). Image clustering and categorization is a technique which could describe the image content at a high level. The goal of doing image clustering is to find a mapping of the archive images into classes (clusters) such that the set of classes provide essentially the same information about the image archive as the entire image-set collection. The generated classes provide a concise summarization and visualization of the image content that can be used for different tasks related to image data preprocessing, image database management, user interface designing and so on. In the usage scenario of EOPEN, we need to fulfil end users' willing to find their patterns of interests, thus the social media images need to be clustered into groups. In this section, we will describe how the image data crawled from Twitter is processed.

Error! Reference source not found. illustrates the process of how the Twitter images are clustered. Regarding the image dataset we get from Twitter, they cannot be directly fed to the model we built for clustering, as it contains a lot of noises and the size of images in most cases are not unified. Thus an image pre-processing is necessary as the first step. The aim of pre-processing is an improvement of the image data that suppresses unwilling distortions or enhances some image features important for further processing. It usually contains several operations: data cleaning (remove noise images and pixels), image resizing, rotating, scaling and so on. After pre-processing, the images will be given to the AutoEncoder model as input.

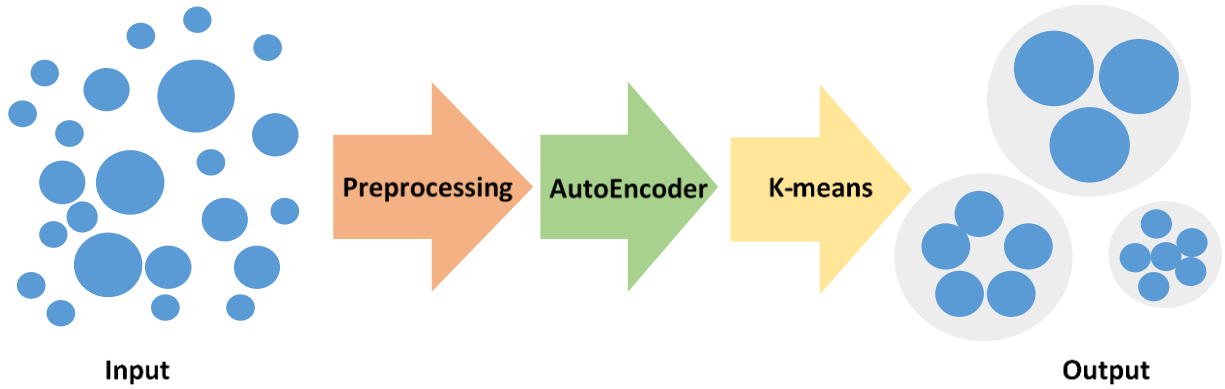


Figure 4: Process of Twitter image clustering

AutoEncoder is a type of artificial neural network (ANN) which is designed to learn data in an unsupervised manner to reconstruct the original input, thus to discover a more efficient and compressed representation. The idea was originated in the 1980s, and later promoted by the seminal paper by (Hinton and Salakhutdinov 2006).

Figure 5 illustrates the basic structure of an AutoEncoder network. It consists of two networks:

- *Encoder* network: The original high-dimension input is translated into a latent low-dimensional presentation. The output size is accordingly smaller than the original input size.
- *Decoder* network: The output of encoder network is the input of decoder network, and through the decoder network the translated image is recovered. Ideally the recovered image shall be the same as the input image.

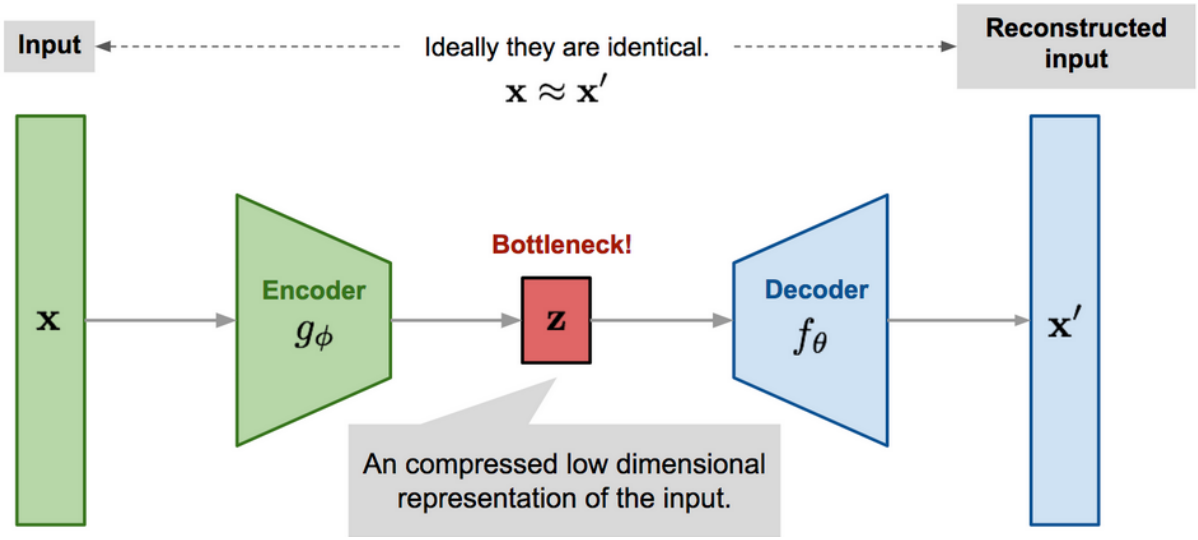


Figure 5: Architecture of AutoEncoder model (Lilian Weng 2019)

The purpose of using AutoEncoder here is to do the dimension deduction, as clustering is difficult to be done in high dimensions because the distance between most pairs of points is similar. Using an AutoEncoder enable you re-represent high dimensional points in a lower-dimensional space without or with little loss. And compared with other dimension deduction technologies, its main advantage is that AutoEncoder can learn non-linear transformations, unlike PCA (Wold et al. 1987), with a non-linear activation function and multiple layers. However, a single AutoEncoder does not promise a good performance of clustering. In order to achieve SoA accuracy of clustering, a combination of AutoEncoder and other clustering methods is usually adopted, for example K-means.

K-means is a method of vector quantization, which aims to partition n observations into $k$clusters in which each partition belonging to the cluster has the least mean distance (squared Euclidean distances). The algorithm is processed in two stages:

- **Assignment step**: Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean.
- **Update step**: Calculate the new means (centroids) of the observations in the new clusters.

The algorithm converges when the assignments no longer change. However, as a type of greedy algorithm, K-means only solve the local optimum problem and cannot guarantee a global best solution.

t_SNE: After the clustering of images is performed, how the cluster quality can be monitored is also a problem, in order to have intuitive overview of the output, the images and their corresponding clusters shall be visualized. However, as high dimensional data, the visualization is always a critical problem. The method “t-distributed Stochastic Neighbour Embedding (t-SNE)” was developed to solve such a problem. t-SNE visualizes high-dimensional data by giving each data point a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbour Embedding (Hinton and Roweis, 2002),

which is more straightforward to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the centre of the map (Maaten, L. V. D. and Hinton, G., 2008). The t-SNE algorithm is mainly composed of two main stages:

- The first stage is to build a probability distribution over pairs of high-dimensional objects. With this distribution, it is highly probable that similar objects are picked, while with little possibility that dissimilar objects being picked.
- In the second stage, another probability distribution over the points in the low-dimensional map is defined, by which the Kullback–Leibler divergence (KLD) between the two distributions with respect to the locations of the points in the map is minimized.

With t-SNE, it is possible to visualize large real-world data sets with limited computational demands. It has been shown that t-SNE outperforms existing state-of-the-art techniques for visualizing a variety of images.With all the images crawled from Twitter, a prototype of 1,000 flooding images was clustered as shown in Figure 6. As can be seen, the images are clustered into 3 clusters, each has a distinct centroid. The cluster is implemented using Keras with TensorFlow on HLRS‘s HPDA infrastructure.

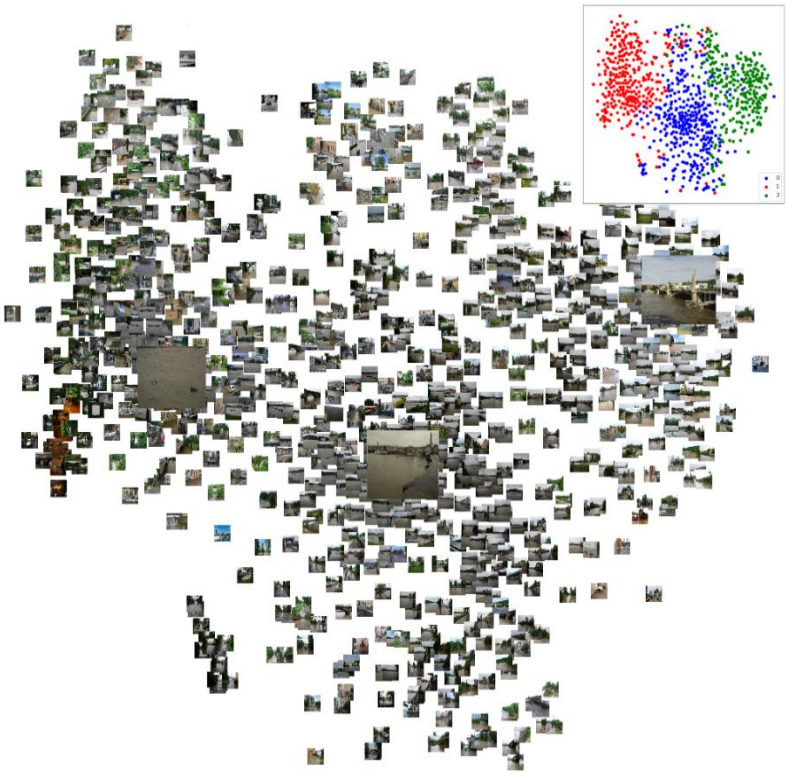


Figure 6: Prototype of clustering flood images from Twitter

4 CLUSTERING OF EO DATA

4.1 Image Segmentation

In the majority of Earth Observation applications ground truth information for training and validation is scarce. Even in scenarios where labelled and/or ground truth data exists (i.e. land cover maps) their quality is not guaranteed. In PUC 2 when dealing with the food security monitoring in South Korea, especially when predicting the production of rice, a significant task is to identify the rice pads from satellite images. As described in D4.1, a change detection algorithm was proposed to remove outliers from a South Korean land cover map of 2015, in order to function as a training dataset for the Random Forest (RF) based classification of rice in 2018. However, with the characteristics of EO applications stated above, there are several problems arising:

1. Data imbalance: the data which has been marked as rice pads is very little, thus cannot be directly used for model training.
2. Noise: The images are high resolution and consisted of a huge amount of pixels and thus contain lots of information. Among the vast information in an image, those which are not relevant to the interested topic can be treated as noise, how to find the necessary information with the presence of so much noise is quite challenging.
3. Data Insufficient: For the same area, the satellite only takes around 3 image per month, thus for the year we studied, there are not enough data to be analyzed with less than 40 images.

Previously, the rice pads were manually labelled within the satellite images, which consumes a lot of labour and time. To simplify this process and eliminate the effort spent on this step, methods which could automatically segment the image, analyse, detect and further label the rice pads need to be developed.

This section is in regard to the creation of a training dataset for supervised classification of rice using K-means clustering. Here, the idea is to construct labelled data for the training of a Random Forest classifier, with the exception that no ground truth information is utilized; labels are created purely from a clustering algorithm.

4.1.1 Related work in Image Segmentation

Image segmentation is a technique by which an image can be divided into many parts, such that the information contained in the image can be depicted in an intuitive way. The image segmentation in most cases is vital for analysis and interpretation of images. The purpose of image segmentation is to divide an image into semantically interpretable regions with regard to a particular application and to identify homogeneous regions within the image as discrete and belonging to distinct objects (Moftah, et al. 2014).

In the last decades, there are many technologies have been developed for segmenting an image. Among them, several methods are quite popular, which also produce relatively good results. Theoretically, these methods can be classified into four groups:

- Region Based Segmentation
- Edge Based Segmentation
- Threshold Segmentation

- Clustering Based Segmentation

Region based segmentation systems try to group pixels together with identical features (such as estimated grey level quality) into regions (Lalitha M., et al. 2013). Compared to other methods, region based segmentation is not so complicated however quite efficiency and noise resilient. The idea behind it is to split the image based on some criterion i.e. object, colour, shape and so on.

Edge based segmentation is one of the most basic processing method used in image segmentation. The idea behind it is to use the edge detection technology to find the boundary of regions (pixels), and then split the image by the closed margin of objects. The outcome of this method is usually an image in binary form. Gray histogram and Gradient are two primary techniques which are used for segmenting image via edge detection (Khan W., et al. 2013).

Thresholding method is a primary technique, which simply converts the gray scale image into binary images by choosing a threshold. What needs to be noticed is that during the conversion, information might be lost, thus the value of threshold must be optimum to make sure that the significant knowledge of interested object like shape and position are not lost.

Clustering algorithms are initially developed for defining groups of similar images. However, in image segmentation it is also widely used. Clustering in image segmentation is defined as identifying groups of similar image primitive (Puzicha J., et al. 1999). An image can be segmented based on different clustering criteria, like similarity, centre, continuous points, density and so on.

4.1.2 **Cluster of Pixels**

The purpose of this methodology is to make the PUC2 Food Security monitoring framework as general, transferable and scalable as possible. Although, the change detection methodology introduced in D4.1, proved adequate for training the RF algorithm, the 2015 land cover map utilized had multiple errors that could not be detected by the change detection algorithm.

Towards this direction and using unsupervised machine learning techniques, we'll produce a set of labelled rice pixels at Seosan and Dangjin districts of South Korea. The input data comprise of a dense time-series of NDVI features from April to September for the year of inspection (2018).

There are many different clustering methods that could be used to solve this problem, and one of the most popular is K-means clustering algorithm. K-means algorithm as an unsupervised algorithm which could be used to segment the interest area from the background. The process could be stated as follows:

1. Initialize the number of clusters
2. For each pixel of the image, calculate the Euclidean distance between the initial cluster centre and the pixel
3. Assign the pixel to a cluster based on its distance to each cluster centre
4. After all pixels of the input image have been assigned to a cluster, recalculate the new cluster centre
5. Repeat the process until it satisfied the fault tolerance or error

6. Reshape the cluster pixels into image

At first, K-means was applied for various numbers of clusters, as seen in Figure 7. The results appeared to be satisfactory; however, it was observed that many rice pixels were classified in the same cluster with water pixels, and vice versa. So, it was decided to apply a hierarchical K-means approach, performing a binary clustering at first to discriminate land and water. Then via extracting the land layer only K-means was applied for various numbers of clusters, as seen in Figure 8. The choice for the optimal number of clusters is selected based on F1-score for each clustering, evaluated against the updated land cover map, as described in D4.1.

In order to automatically detect which is the rice cluster in each clustering, an expected mean NDVI of rice pixels must be given for each NDVI feature. The cluster which centre has the minimum mean squared error (MSE) compared with the given mean NDVIs is considered as the rice cluster. Below we present figures with the corresponding recall and precision scores for each experiment. Our goal is to achieve a rather high precision score, ensuring that the rice cluster includes predominantly rice pixels. Additionally, we tried to achieve a recall that is as high as possible in order to create a representative training dataset, which included all different versions of rice in the area of interest.

Figure 7: Recall and precision scores for K-means algorithm with different number of clusters (from 8 to 19), without extracting water pixels.

In Figure 7, we present the scores achieved by the K-means algorithm for the first scenario, in which we do not extract the land and water extents first, for a varying number of clusters. It is obvious that for low number of clusters, recall is usually very high in contrast with precision, which is at most around 70%. That means that the cluster of rice includes most of the rice pixels but a large amount of non-rice pixels too. While the number of clusters is increasing, we observe a balance between the two accuracy metrics (precision, recall) and

for even more clusters the precision metric is improved, while recall shows a small reduction.

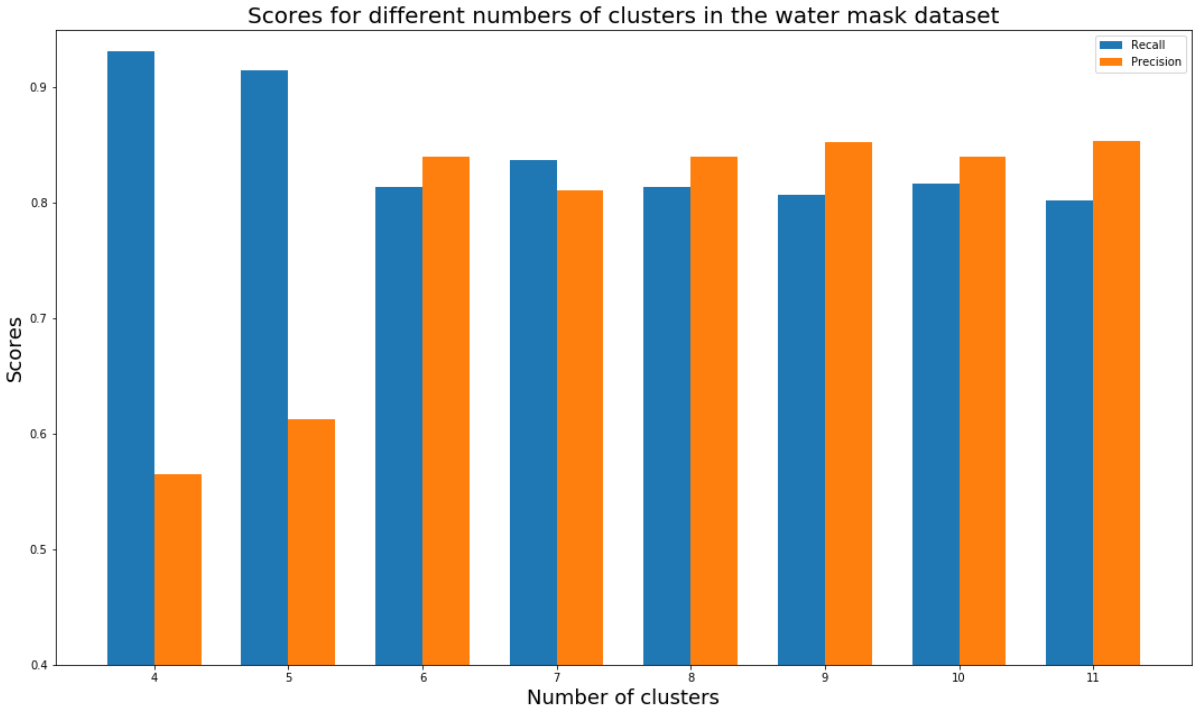


Figure 8: Recall and precision scores for K-means algorithm with different number of clusters (from 4 to 11), having removed water pixels.

Figure 8 shows the accuracy metrics achieved by the hierarchical K-means algorithm, for a varying number of clusters. We can make the same observations as for Figure 7. The only significant difference is the number of clusters. Here, it ranges from 4 to 11 and as we can see the algorithm achieves quite satisfying results from 6 and above, while the number of clusters for the first algorithm ranges from 8-19 and achieves satisfying results from 13 clusters and above.

4.1.3 **Evaluation**

In order to compare the ground truth land cover map (see D4.1) and the land cover map that results from clustering, we trained two separate Random Forest models, using the two aforementioned datasets respectively. Below, the scores achieved by the predictions of each model against both datasets, a) clustering-based rice map and b) updated land cover map (see D4.1) are presented.

Table 1: Classification report of predictions of model trained on clustering-based training dataset VS the updated land cover map.

Label	Precision	Recall	F1 score	Pixels
0	**0.93**	**0.95**	**0.94**	**11901997**
1	**0.85**	**0.82**	**0.83**	**4410363**

Table 2: Classification report of predictions of model trained on clustering-based training dataset VS clustering-based training dataset.

Label	Precision	Recall	F1 score	Pixels

0	**0.99**	**1**	**0.99**	**12032654**
1	**0.99**	**0.98**	**0.99**	**4279706**

From Tables 1 and 2 it is obvious that the predictions of the Random Forest model which was trained using dataset (a), outputs excellent results when compared to itself. On the contrary, when it is compared to dataset (b) accuracies are much lower. However, it is shown in the next paragraph that precision and recall accuracies are higher, since he updated land cove map is subject to significant noise.

To further examine the quality of the produced clustering based rice map, we record the misclassified pixels of each model, and from those we drop the common misclassifications and keep the remaining pixels. Then we created a raster with values only on the positions of the recorded non-matching misclassified pixels, which were then sieved in order to remove island pixels. Through the use of photo interpretation, we examined a subset of these misclassifications. It was observed that from 600 examined hectares, 314 had been correctly misclassified, while the remaining 286 were indeed errors of the algorithm. All this was done in order to show that the clustering-based training set actually results in better precision and recall than the ones showed in Table 1. This is true since the updated land cover map that was assumed to be a perfect validation dataset appears to have significant errors. Thus 52.33% of the pixels misclassified, according to the Table 1 metrics, were actually classified correctly.

4.1.4 **Future Work**

Although K-means has the great advantage of being easy to implement, however, it has some drawbacks. The quality of the final clustering results strongly depends on the arbitrary selection of initial centroid. So if the initial centroid is randomly chosen, it will get different result for different initial centres. And also computational complexity is another term which we need to consider.

Nowadays, as the DNN has shown its advantages in image processing, a lot of DNN based methods has come up and shown great improvement in processing different areas of images, including medical, remote sensing, street view images and so on. Thus, further research and study employing other algorithms like U-net (Ronneberger O., et al. 2015), W-net (Xia X., et al. 2017), Seget (Badrinarayanan V., et al. 2017) will be carried on in the future.

4.2 **Clustering satellite image patches**

In the previous section (Section 4.1 the work has focused on pixel clustering so as to segment a satellite image into patterns of similar content as groups of pixels. In this section, we provide a different clustering approach for EO data, where satellite image patches, mainly from Sentinel 2 imagery, are clustered into groups of patches, in an unsupervised way. The lack of training data in the EO domain results to the inability of the EO downstream sector to use supervised machine learning approaches for pattern recognition in satellite images. The problem becomes challenging when the number of clusters is not known and the labels are not known or not annotated. To that end, density-based algorithms are suitable, because they do not require the number of clusters as input, contrary to other clustering approaches such as k-means. Density-based algorithms require as input two other parameters: the minimum number of points per cluster and a density level. However, their estimation is hard to be done and often requires several executions and combination of

several outputs from multiple density levels. In this context, we present a parallel implementation of DBSCAN-Martingale, as it has been presented in Section 2.3 and we evaluate our approach on a benchmark dataset of Sentinel 2 images with ground-truth annotation.

4.2.1 Related work

As it has been mentioned in Section 2, over twenty years ago, (Ester et al., 1996) introduced the now popular DBSCAN, a density-based algorithm for discovering clusters in large spatial databases with noise. DBSCAN is still being used both in research and real-world applications, while it has inspired numerous extensions. Some of the main improvements that the scientific community has focused on is the optimisation of DBSCAN's parameters, i.e. MinPts density threshold and Eps neighbourhood radius, and the performance time, proposing faster versions of the algorithm.

Regarding the parameters optimisation, the algorithm in (Diao et al., 2018) divides the data set into multiple data regions, sets the appropriate parameters for each data region for local clustering, and finally merges the data regions. (Kim et al., 2019) focus on minimizing the additional computation required to determine the parameters by using the approximate adaptive ε-distance for each density while finding the clusters with varying densities that original DBSCAN cannot find. (Lu, 2019) proposes a self-adaption grey DBSCAN algorithm that automatically selects its parameters, whereas (Li et al, 2020b) calculate the parameters MinPts and Eps based on the optimal k-value, selected after multiple iterations of k-clustering. Furthermore, in the works of (Hou et al., 2019) and (Li & Chen, 2018), the problem of proper parameterisation is solved by applying the density peak algorithm and the natural neighbour algorithm respectively.

Focusing on the Eps parameter, (Lai et al., 2019) designed a new mechanism for the variable update of the multi-verse optimiser algorithm in order to search the range of Eps, while (Starczewski & Cader, 2019) determine its value based on an analysis of the sorted values of the distance function and (Zhu et al., 2018) based on Gauss kernel density. On the contrary, (Mu et al., 2020) propose a parameter-free algorithm to perform DBSCAN with different-level parameters.

As far as it concerns the performance, several recent works present revisions of DBSCAN, aiming to reduce the computation time. (Pandey et al., 2020) target the time spent in the input/outputs (reading/writing data), (Luchi et al., 2019) reduce the number of patterns presented to DBSCAN, and (Chen et al., 2018) prune unnecessary distance computations for high-dimensional data. Work in (Galán, 2019) inspects the neighbourhoods of only a subset of the objects in the dataset, similarly to the modification of DBSCAN presented in (Jang & Jiang, 2018) that requires computing only the densities for a chosen subset of points. Likewise, the approaches of (Shiqiu & Qingsheng, 2019) and (Chen et al., 2019b) run on selected core points, the first based on locality sensitive hashing and the latter on k-nearest neighbours.

In addition, (Ding & Yang, 2020) apply a novel randomized k-centre clustering idea to decrease the complexity of range query, which is a time-consuming step in DBSCAN, while (Ekseth & Hvasshovd, 2019) present a method and a software with filtered computation of the dense similarity matrix, which significantly boosts the performance. (Cai et al., 2020)

improve the efficiency with Ada-DBSCAN, an extension that consists of a data block splitter and a data block merger, coordinated by local clustering and global clustering, and (Li et al., 2020a) with ADBSCAN, an approach for identifying local high-density samples utilizing the inherent properties of the nearest neighbour graph. Other extensions that present good performance are the 3W-DBSCAN (Yu et al., 2019), which represents a cluster by a pair of nested sets called lower bound and upper bound, the Bisecting Min Max DBSCAN (Johnson et al., 2018), and the method in (Lin et al., 2019) that only computes the distances between the object and its nearby neighbours.

Towards decreasing the complexity of DBSCAN, (de Berg et al., 2019) avoid spending time on every edge in the neighbourhood graph by working with box graphs, while (Huang et al., 2020) exploit the Warshall algorithm to mitigate its complexity. In (Bryant & Cios, 2018), problem complexity reduces by using a single parameter (choice of k nearest neighbours) and by handling large variations in cluster density (heterogeneous density). Moreover, (Mathur et al., 2019) use a combination of distance-based aggregation by overlaying the data with customized grids and (Boonchoo et al., 2019) utilize bitmap indexing to support efficient neighbour grid queries.

The upsurge of distributed and high-performance computing technology has motivated scientists to propose various parallel implementations of DBSCAN. (Yang et al., 2019) present a framework that divides data into partitions, calculates local DBSCAN results, and merges local results based on a merging graph. (Deng et al., 2018) enable parallel computing by a divide-and-conquer method that includes a simplified k-mean partitioning process and a reachable partition index, and (Gong et al., 2018) present a new approach that supports real-time clustering of data based on continuous cluster checkpointing. Additional parallelised versions of DBSCAN are found in the works of (Kumari et al., 2019), (Hu et al., 2018) applying MapReduce, (Zhou, 2018) with a parallel grid clustering algorithm, and (Sarma et al., 2019), who developed a scalable distributed implementation of their own μDBSCAN.

(Wang et al., 2019) and (Ibrahim & Shafiq, 2018) manage to parallelise DBSCAN by using Quadtree data structure. In the latter, the solution distributes the dataset into smaller chunks and then utilizes the parallel programming frameworks such as Map-Reduce to provide an infrastructure to store and process these small chunks of data. (Li, 2020) utilizes Cover Tree to retrieve neighbours for each point in parallel and the triangle inequality to filter many unnecessary distance computations. Alternatively, AnyDBC by (Mai et al., 2018), instead of performing range queries for all objects, iteratively learns the current cluster structure of the data and selects a few most promising objects for refining clusters at each iteration.

A further step to distributed implementations of DBSCAN is the optimisation of the involved stages. (Chen et al., 2019a) propose improvements both in data partitioning stage and in merging stage. (Han et al., 2018) adopt a partitioning strategy based on kdtree, similarly to (Shibla & Kumar, 2018), and a new merging technique by mapping the relationship between the local points and their bordering neighbours. (Tyercha et al., 2019) use a Hilbert curve to identify the centres for initial partitioning, (Liyang et al., 2019) optimise it with distance matrix and R-Tree based methods, and (Song & Lee, 2018) suggest a cell-based data partitioning scheme, which randomly distributes small cells rather than the points themselves.

Finally, in a more technical level, (Hahsler et al., 2019) describe the implementation of DBSCAN as an R package and (Shahriar et al., 2019) compare the performance of the algorithm on four different platforms, including R.

4.2.2 Methodology for an automated workflow on HPC

In Section 2.2 we have presented the background and notation on DBSCAN-Martingale, and Algorithm 1 followed with the estimation of the number of clusters $\hat{k}$. The presented DBSCAN-Martingale algorithm requires T iterations of the DBSCAN algorithm each one of

Algorithm 2: Parallel-DBSCAN-Martingale(T, N, w) **return** $\hat{k}$
1: Allocate T cores and$N$ nodes
2: Set the number w of different values of$minPts$
3: Extract feature vectors per satellite image patch using any feature extraction method
4: Find ϵ_{max} using the maximum reachability distance from an OPTICS reachability plot
5: For $minPts \in \{minPts_1, minPts_2, \dots, minPts_w\}$:
6: Generate a random sample of T values in $[0, \epsilon_{max}]$
7: Sort the generated sample $\epsilon_t, t = 1,2, \dots, T$
8: **for** $t = 1$ to T
9: compute $C_{DBSCAN(\epsilon_t)}$ distributed in each one of the T cores
10: compute $C^{(t)}$ distributed in each one of the T cores
11: update the cluster IDs
12: update the vector C
13: update $\hat{k} = \max_j C[j]$
14: end **for**
15: end **for**
16: compute $\delta_i = \left|\frac{\hat{k}_{i+1} - \hat{k}_i}{minPts_{i+1} - minPts_i}\right|, i = 1,2, \dots, w-1$
17: compute $minPts$ such as: $minPts = \arg\min_i \delta_i$
18: find index i such that $minPts = minPts_i$
19: **return** $\hat{k}_i$

the T iterations and number of realisations are allocated in multiple nodes and cores so the algorithm can scale up using an HPC infrastructure. The method is tailored on satellite image patches and the notation ($C_{DBSCAN(\epsilon_t)}, C^{(t)}, C[j]$, etc.) is defined in Section 2.2

In Figure 9 we present our proposed framework for the implementation of Algorithm 2.

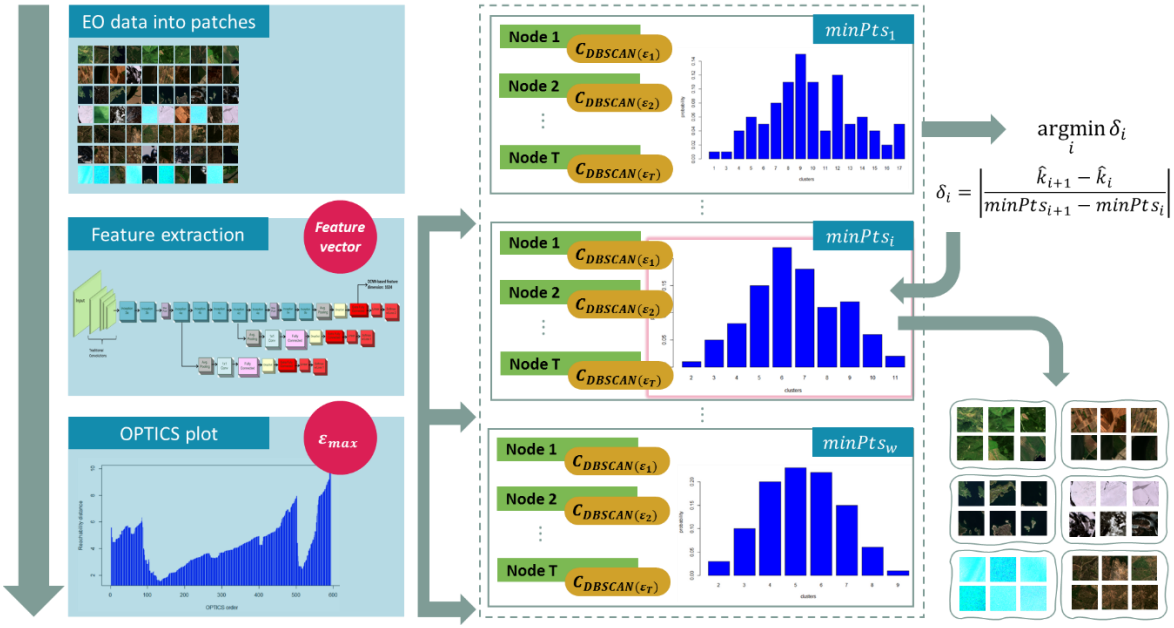


Figure 9: Parallel-DBSCAN-Martingale

4.2.3 Evaluation

Dataset description

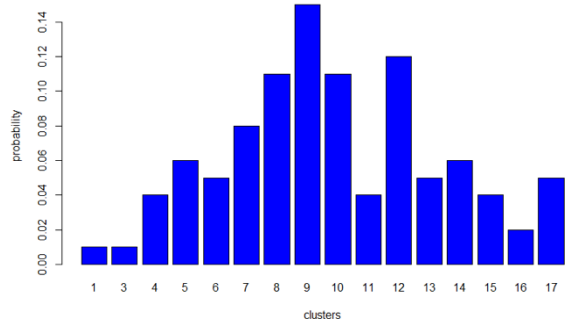
For the evaluation of our approach, we have selected the BigEarthNet (http://bigearth.net/) dataset, which is also referred in D4.3, since it has been used in the experiments for retrieving most relative EO content. The dataset contains ground-truth annotation about Sentinel 2 satellite images and counts 590,326 patches of size 120x120. Each patch may contain one or more of the following labels: water, rice, urban, vineyards, forest, bare rock, and snow.

By definition, clustering cannot handle a multi-labelling problem (each patch will be assigned to a single cluster), thus we have excluded the patches whose ground truth is more than one label. The modified dataset presented noticeable imbalance in the number of patches for each class (e.g. 220k forest, 1k rice, 52 rock), so we kept only 1,000 patches of each label, while "rock" had to be completely eliminated, leading to a sum of 6,000 patches.

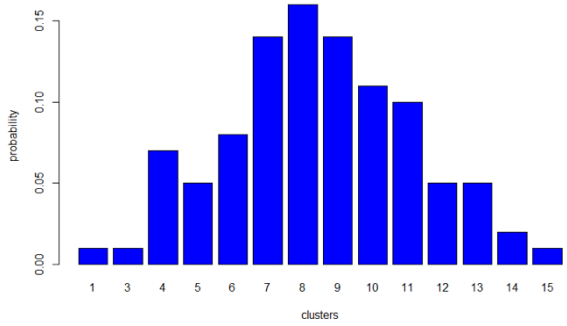
Finally, since the Parallel-DBSCAN-Martingale requires multiple iterations and is a time demanding technique, we have decided to take a normally distributed sample of 600 patches.

Results

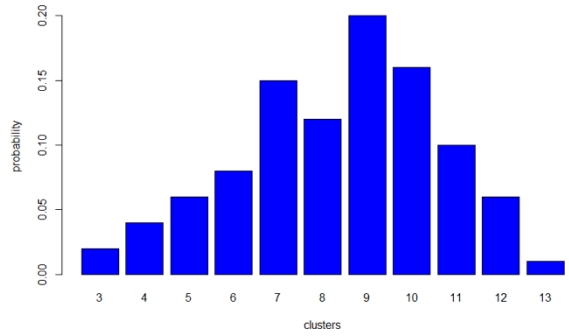
In our experiment, T is set to 5, ϵ_{max} to 10, and w to 12, with $minPts$ varying from 5 to 16. For each$minPts$, the following table contains the plot with the probability of the number of clusters and the OPTICS reachability plot.



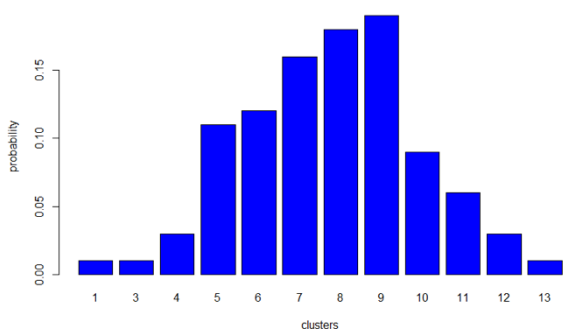
(a) $minPts$ = 5



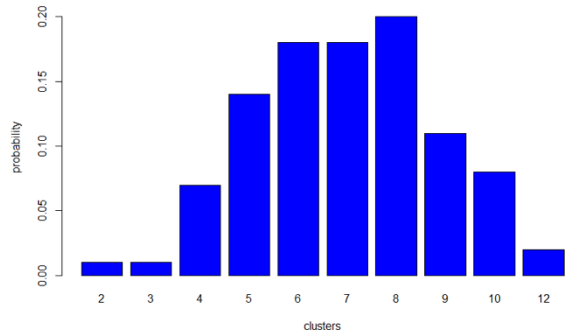
(b) $minPts$ = 6



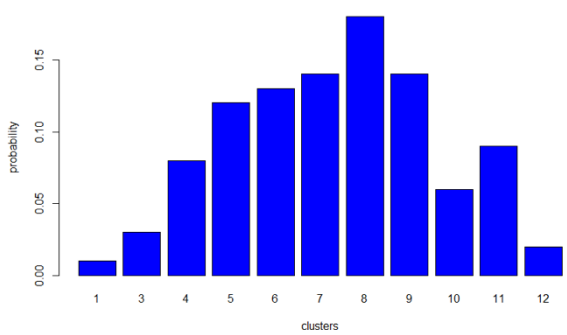
(c) $minPts$ = 7



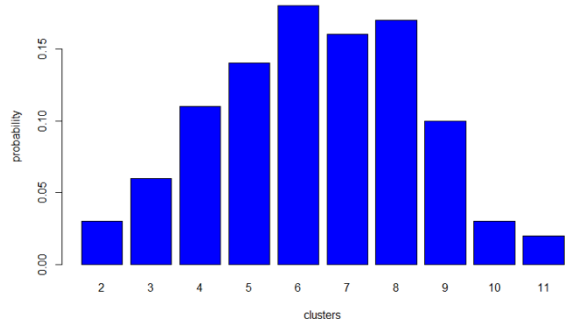
(d) $minPts$ = 8



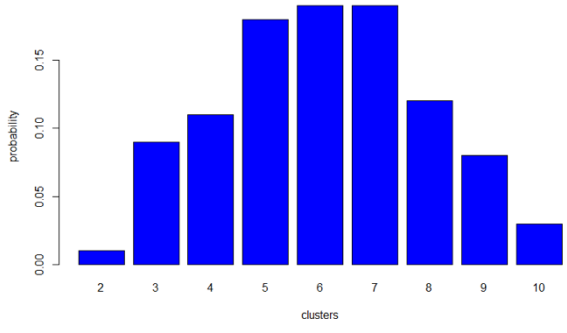
(e) $minPts$ = 9



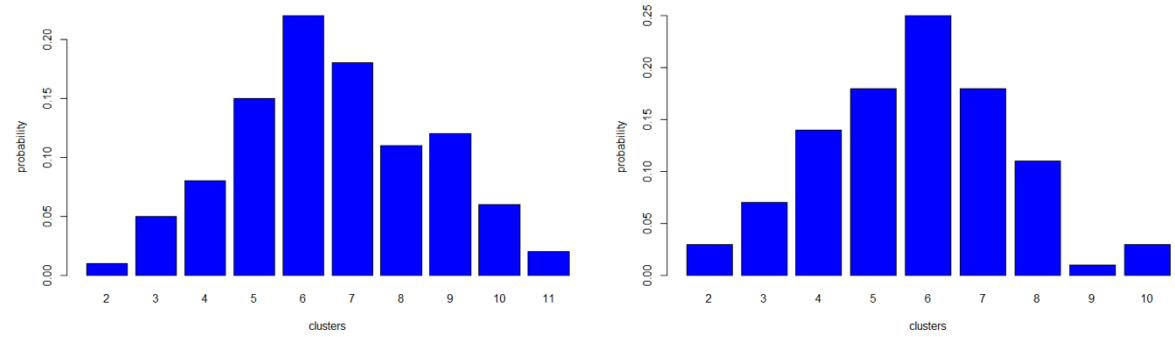
(f) $minPts$ = 10



(g) $minPts$ = 11

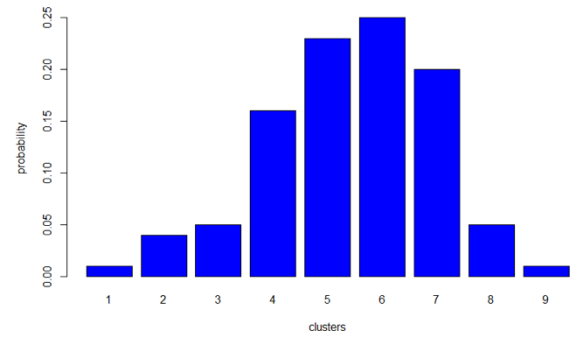


(h) $minPts$ = 12

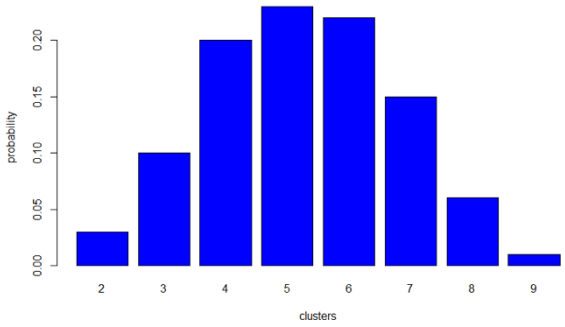


(i) $minPts$ = 13

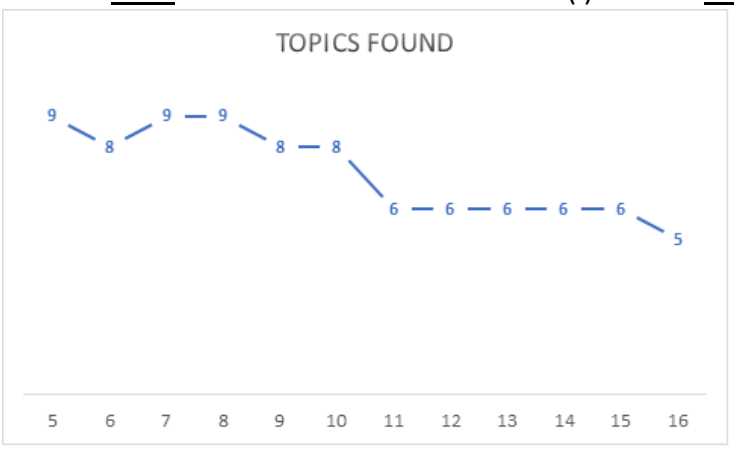
(g) $minPts$ = 14



(k) $minPts$ = 15



(l) $minPts$ = 16



(m) Most probable number of clusters per $minPts$

Figure 10: Probability for the estimated number of clusters (a)-(l) and a summary (m).

The proposed framework of Parallel-DBSCAN-Martingale generates a multitude of results for $minPts = 5,6,\ldots,16$ and then gets the final result in an unsupervised way, searching for the most "stable" number of clusters as an optimal solution (Figure 10). In case such an optimal value does not exist, we assume that the clusters are very mixed into one and the dataset is not separable. The feature extraction part generates one vector per satellite image patch using a Deep Convolutional Neural Network layer. However, the vector representation can be of any type such as color histograms or other feature vectors. Estimating the number of clusters with the highest probability for every $minPts$ requires computational effort that needs to be distributed in multiple processing nodes for scalability. The maximum reachability distance is an indication of ϵ_{max} and it is sufficient to be computed only for one value of $minPts$.

4.2.4 Concluding remarks and future work

Clustering satellite image patches allows a fast grouping of Earth Observation data into clusters of similar semantic content, i.e. concepts such as water, snow, rock, urban, rice, etc. This process is unsupervised and requires no label information or number of clusters a priori. In an operational level, once a cluster of satellite image patches is obtained and one of the patches gets a label (e.g. this is an urban surface) then this label is propagated to all other members (patches) of the same cluster.

In this Section we proposed a parallel version of a state-of-the-art clustering algorithm, namely DBSCAN-Martingale which estimates the number of clusters in an automatic way. The use of HPC allows us to share the processing task into several processing nodes, offering a scalable solution to the EO Big Data community. The parameters of the proposed density-based approach are also learned in an unsupervised way and a stable solution is searched with many executions and realisations of the process, to get the optimal value for the number of clusters.

Once the optimal number of clusters is obtained, then any traditional clustering algorithm (e.g. k-means) can be applied, having as input the estimated number of clusters and the vector representation of the satellite image patches. The evaluation has taken place on the correct estimation of the number of clusters as a Key Performance Indicator. The final version of this framework and additional experiments will be part of the final deliverable of WP4, namely D4.4 - Change, event, and community detection techniques, which is due on month M33.

5 INTEGRATION

5.1 Implementation on HPC infrastructure

In order to achieve the best performance of clustering, the algorithms will be run on the infrastructure provided by HLRS, including a dedicated Big Data system, a Cray Urika-GX (Cray Inc., 2019) and an HPC system, a Cray XC40 (Cray Inc., 2019). On these two clusters, the environments are setup so that machine learning (ML) and deep learning (DL) algorithms can be processed effectively. The accessible infrastructure, software stack and the strategy for code immigration will be described briefly below.

5.1.1 Hardware Infrastructure

The HPDA system Cray Urika-GX and HPC system Cray XC40 are provided by HLRS in order to accelerate the computation, shorten the running time and improve the performance.

Cray Urika-GX: The Cray Urika-GX is a dedicated Big Data analytics system, specifically configured with a HPC interconnect (G. Faanes et al., 2012), local SSD storage for fast data transfers, and 512 GB of RAM per compute node. Especially, near real-time data analytics are enabled to be performed in memory of HPDA system equipped with the high amount of RAM. The HPDA system naturally enables different kinds of machine learning and data analysis tasks, including the clustering task (imagery, textual and pixel). The technical details of HPDA system can be found in **Table 3**.

Table 3: Technical specification of the two Cray Urika-GX systems installed at HLRS.

	System 1 (Gilgamesch)	**System 2 (Enkidu)**
Total number of nodes including management and service nodes	48	16
Number of compute nodes	41	9
Processors per node	2 x Intel Broadwell 18-Core, 2.1 GHz	
RAM per node	512 GB	
Disk storage per node	2 TB HDD; Intel DC P3608 SSD (1.6TB)	
External parallel file system	Sonexion 900 with 240 TB (shared), throughput of 4.0 GB/s	
Operating system	CentOS 7	
Available Standard Software	• **Resource managers:** Apache Mesos, Apache YARN • **Data Analytics Frameworks and Tools:** Apache Hadoop, Apache Spark, Cray Graph Engine, GNU R	
Main Programming Languages	Scala, Java, Python, GNU R	

Cray XC40: The High-Performance Computing system “Hazel Hen” is a Cray XC40 machine with more than 185,000 CPU cores and a total of 964 TB of RAM. The system is predestined for compute-intensive simulations and scientific applications, however, as training deep learning algorithms on a huge set of data (especially images) is quite compute-intensive, thus HPC also plays an important role in deep learning, especially clustering here. The technical specification of the Cray XC40 at HLRS is listed in **Table 4**.

Table 4: Technical specification of the Cray XC40 at HLRS.

	Cray XC40 (Hazel Hen)
Cabinets	41 Cabinets à 1.5t
Number of compute nodes	7712

Processors per node	2 x Intel Haswell 12-Core, 2.5 GHz
RAM per node	128 GB
Disk storage per node	None
External parallel file system	Sonexion 1600 with more than 20 PBs (shared)
Operating system	Cray Linux Environment (CLE) and Compute Node Linux (CNL)
Available Standard Software	• **Resource manager:** PBS/Torque • Cray Developer Toolkit (MPI, PAPI, GNU compilers, debugger, …) • Cray Programming Environment including math and scientific libraries such as optimized BLAS, LAPACK
Main Programming Languages	FORTRAN, C, C++

For more details about the hardware, please refer to D6.2. From beginning of 2020, we have now the successor system installed, a HPE Apollo named “Hawk”, which has more compute power.

5.1.2 Software Stack

The processing and storage of big data, especially running machine learning and deep learning algorithms on large-scale distributed computing system often require specific frameworks. The Cray Urika-GX builds upon the Apache software stack for data analytics, for example, Apache Hadoop, Apache Spark, and HDFS, which are the essential fundamentals for clustering of large amount of textual and imagery data on HPDA.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models (Apache Hadoop, 2019). The core of Apache Hadoop consists of two parts: one is storage part which is the well-known Hadoop Distributed File System (HDFS) (Apache Hadoop, 2019) and the other one is the MapReduce programming model (Apache Hadoop, 2019). The data will be split into small blocks and distributed across nodes. Then the packaged code will be transferred to the node and process the small block data in parallel. Therefore, the dataset is processed faster and more efficiently.

In order to simplify the process of implementation of algorithms, Apache Spark as a cluster-computing framework is also provided. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. MLlib, as Spark’s machine learning library provides tools including common machine learning algorithms such as classification, regression, clustering, and collaborative filtering, data pre-processing, such as feature extraction, transformation, dimensionality reduction, and selection. Besides, Spark also introduces the concept of ML Pipelines, which provides a uniform set of high-level APIs built on top of DataFrames (Apache Hadoop, 2019) that help users create and tune practical machine learning pipelines. For example, for the textual clustering, LDA algorithm could be implemented through:

```
from pyspark.ml.clustering import LDA
```

and for the imagery clustering, K-means could be implemented through:

```
from pyspark.ml.clustering import KMeans
```

Besides, Spark also provides the evaluation metrics in `Evaluator` which can evaluate models for given parameters, especially the `ClusteringEvaluator` could be used to evaluate the performance of clustering models.

As mentioned above, Apache Spark can cover most common machine learning algorithms. However, for the processing of imagery data, traditional machine learning algorithms are sometimes not powerful enough, thus deep learning is introduced. To train a deep learning model and apply this model to the imagery data, TensorFlow is employed. TensorFlow is an open source software library for numerical computation using data flow graphs (TensorFlow, 2019). With its high-level API, Keras, common use cases can be solved by building deep learning models with barely no restriction. For example, in the clustering of imagery data, AutoEncoder could be simply built through connecting configurable building blocks.

For more information about the software stack in HPDA and HPC, please refer to D6.2.

5.1.3 Execution of text clustering on HPDA

In this part, we report the execution of the text clustering methodology, descripted in Section 3.1 , as it has been conducted on the HLRS' HPDA system Urika-GX (Figure 11). The script that implements the method is written in R language and the main target of the experiment was to run it on a parallel way.

```
################################################################################
                              Welcome to

  _____________                                                  ____
 /   _____/|__|  |    _________    _____   ____   ______ ____ |  |__
/    \  ___|  |  |   / ___\__  \  /     \_/ __ \ /  ___// ___\|  |  \
\     \_\  \  |  |__/ /_/  > __ \|  Y Y  \  ___/ \___ \\  \___|   Y  \
 \_____  /__|____/\___  (____  /__|_|  /\___  >____  >\___  >___|  /
       \/        /_____/     \/      \/     \/     \/     \/     \/

This is a Cray Urika-GX analytics system. It is otimized for:

                       CGE (CrayGraph Engine)
                       Spark
                       Hadoop

In Gilgamesch we have installed 42 compute Nodes. Each compute node
has the following configuration:

                       512 GB of RAM
                       Two Intel E5-2695V4 2.1GHZ (Broadwell)
                       2 HDD with 2 TB each
                       1 SSD with 1,6 TB

The installed Software stack:

                       Urika-GX-2.2UP02


Regards,
Your Cray-Team
################################################################################

-bash-4.2$
```

Figure 11: Welcome message on the login node of the Cray Urika-GX, which contains hardware information

The parallelization was successfully performed across the cores of a single node, but not across multiple nodes, due to a limitation of R. The first step was to transfer to the system the R script and the folder with the 1,520 Wiki pages that served as the dataset of the

experiment. This was achieved via the SCP command and confirmed by checking the contents in the home directory, which exists in the HPDA's disk:

```
-bash-4.2$ ls
dbscan_martingale_mc.r  gsl-2.6  local  R  WikiRefPages
```

Figure 12: Home directory structure

Inside the execution environment, version 3.4.3 of R was already pre-installed on the HPDA system, but additional libraries, such as *dbscan* 1.1-5 and *doParallel* 1.0.15, had to be installed:

```
> sessionInfo()
R version 3.4.3 (2017-11-30)
Platform: x86_64-redhat-linux-gnu (64-bit)
Running under: CentOS Linux 7 (Core)

Matrix products: default
BLAS/LAPACK: /usr/lib64/R/lib/libRblas.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] doParallel_1.0.15 iterators_1.0.12  foreach_1.4.7     dbscan_1.1-5

loaded via a namespace (and not attached):
[1] compiler_3.4.3   Rcpp_1.0.3       codetools_0.2-15
```

Figure 13: Session information of R environment, adjusted for DBSCAN execution

The script automatically detects the number of available cores and suggests using the maximum number of cores minus one, in order to keep some resources for other tasks to run. Additionally, there is the option of a hard limit that allows us to set the limit of the maximum number of cores. The values that were tested are: 1, 2, 4, 8, 16, and 32 cores.

```
-bash-4.2$ cat dbscan_martingale_mc.r
library(dbscan)
library(doParallel)

#select the maximum cores to use (hard limit)
cores.limit <- 2 #1 2 4 8 16 32

#detect the system (current node's) cores
cat(paste('Stats: ', detectCores(logical = FALSE), detectCores(logical = TRUE), '\n'), sep=' ')

#suggested cores is the maximum number of cores minus one
suggested.cores <- detectCores(logical = TRUE) - 1
#The cores to be used cannot exceed the hard limit
num.logical.cores.use <- min(cores.limit, suggested.cores)
cat(paste("Num. of cores to be used:", num.logical.cores.use, "\n"))
#Register the number of cores to use
registerDoParallel(cores=num.logical.cores.use)
```

Figure 14: R code that registers cores for parallel execution

Regarding the DBSCAN parameters, explained in Section 2, the *minimum points* were set to 4, the *maximum eps value* to 0.2, *T* equals 5, and the realizations are 1,000.

```
#DBSCAN parameters
minpts = 4
eps_max = 0.4511
T = 5
realizations = 1000
```

Figure 15: R code with the DBSCAN core settings

The part that we decided to parallelise was the realizations of the algorithm, which are basically multiple iterations of DBSCAN for different values of the eps parameter. To this end, the original R script had to be modified, so that each DBSCAN run is performed on a separate core, in parallel. The “foreach” command allows the execution of the iterations on different cores at the same time and is activated with the *%dopar%* command. Each execution is performed on the whole dataset and returns an estimation of the number (an integer) of detected clusters. Eventually, the results of all the iterations are gathered to the “x1” variable.

```
x1 <- foreach(r = 1:realizations) %dopar% {

  number.of.clusters = c()

  ### generate 10 random numbers from the uniform distribution in [0, eps_max] ###
  random.epsilon = runif(T, min=0, max = eps_max)
  random.epsilon = sort(random.epsilon, decreasing = FALSE)
```

Figure 16: R code for activation of parallel iterations

After achieving the parallel execution, a further analysis has been realized, to investigate how time and cost range in relation to the number of used cores. As a reference for prices, we have selected the EC2 instances of Amazon Web Services (AWS), given that they support different number of vCores. The official prices are in US dollars, so we avoided converting them in euros.

The fluctuation of execution time and execution cost per number of cores is reported in Table 5 and also displayed as a line chart in Figure 17. Moreover, we define marginal cost m_i as:

$$m_i = \frac{v_{c_i} - v_{c_{i-1}}}{t_{c_i} - t_{c_{i-1}}}$$

where v_c is the price value, t_c is the processing time and c is the number of cores. The fluctuation of marginal cost m_i is illustrated in Figure 18 and shows that the selection of 4 cores seems an optimal selection, considering the amount of price increase with respect to the amount of the decrease in processing time.

Table 5: Time and cost analysis of multi-core execution of DBSCAN

EC2 Instance	Cost/hour	Hardware	Machine Orientation	Execution Time (hours)	Execution Cost
t2.small	$0,026	1 core, 2gb RAM	General purpose	85	$2,244
c5d.large	$0,115	2 cores, 4gb RAM, SSD	Compute Optimized	46,96	$5,400
c5d.xlarge	$0,230	4 cores, 8gb RAM, SSD	Compute Optimized	24,846	$5,715
c5d.2xlarge	$0,460	8 cores, 16gb RAM, SSD	Compute Optimized	13,18781	$6,066
c5d.4xlarge	$0,920	16 cores, 32gb RAM, SSD	Compute Optimized	7,083	$6,516
c5d.9xlarge	$2,070	32 cores, 72gb RAM, SSD	Compute Optimized	3,75	$7,763

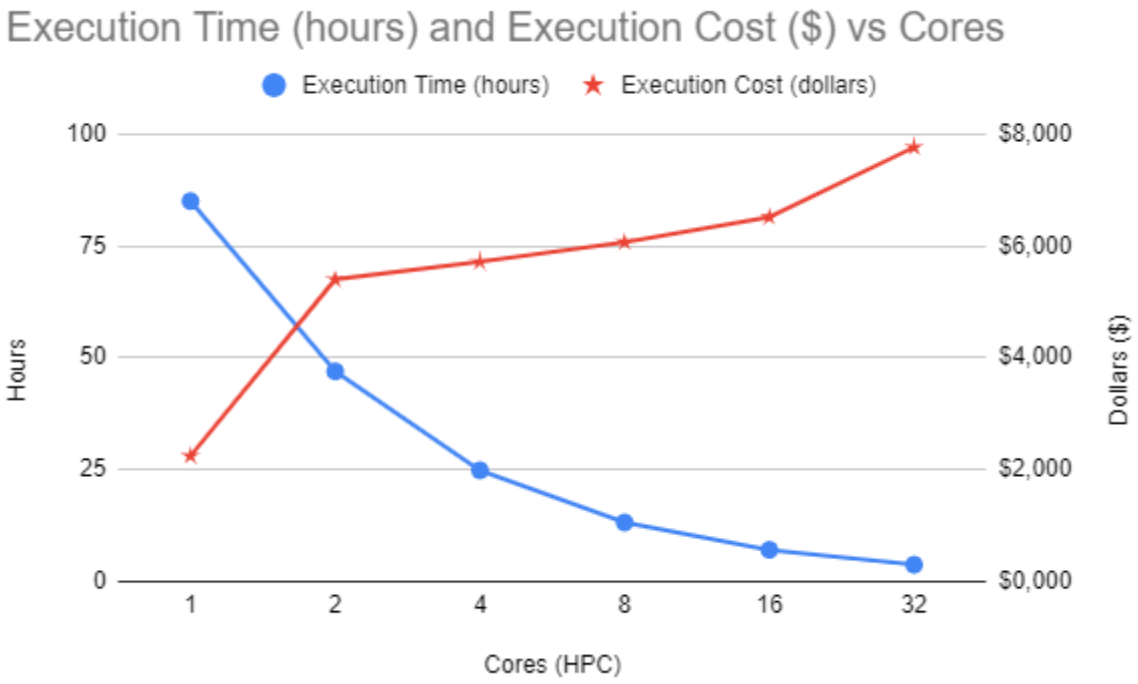


Figure 17: Execution Time (hours) and Execution Cost ($) vs Cores

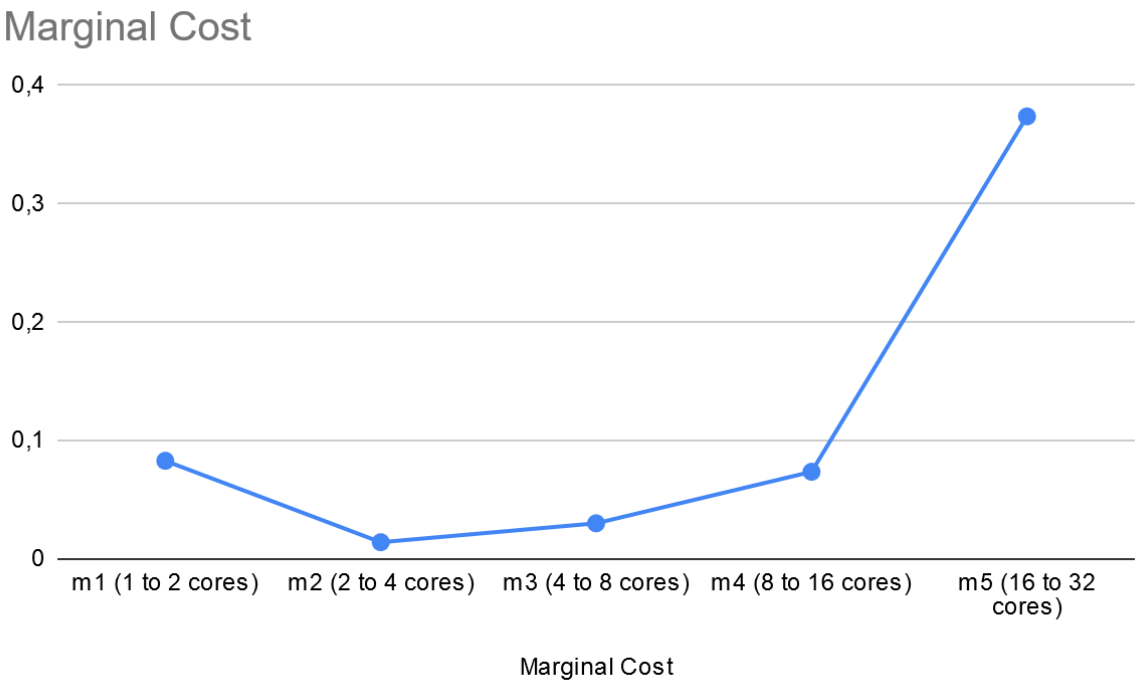


Figure 18: Marginal cost while increasing number of cores

5.1.4 Execution of EO clustering on HPDA

In this part, the execution of EO clustering methodology which is described in Section 3.2.2 will be reported. In order to achieve optimized performance, DBScan on Apache Spark was deployed on HLRS' HPDA system Urika-GX, and the script that implements the method is written in Python.

Implementation details:

Distribute implementation of DBScan works by casting a virtual fishnet on the data. DBScan computation is run separately on the data points residing in the same cell. User can define the cell size depending on the data. Points in the desired cell that are epsilon distance away from the points in all the other cells are broadcasted to the same cell. Finally, merging step would take into account both of these points. Local clusters are produced for each cell are then merged into one global cluster, which is achieved by relabeling all the points in cells from their local cluster to global cluster. In Figure 19, data is divided into two cells. Green circle defines the local cluster for Cell 1, whereas red circle defines the local cluster for Cell 2. Blue circle defines the global cluster which is achieved by merging the local clusters of Cell 1 and Cell 2.

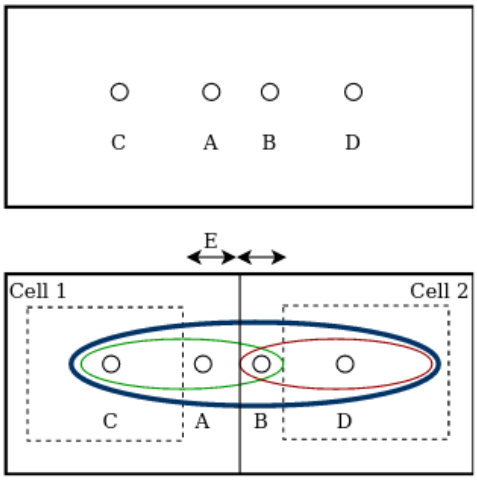


Figure 19:Distributed DBScan Implementaion[2]

Running DBScan Spark on Urika-GX:

Following command is used to execute DBScan on Apache Spark

```
spark-submit –total-executor-cores $core –executor-memory 450g
              $path_to_jar $path_to_properties
```

[2] https://github.com/mraad/dbscan-spark

Here $core defines the number of cores and *$path_to_jar* points to the implementation of DBScan Spark. *$path_to_properties* is the path to the properties file that contains the configuration information. Properties file contains the following parameter: path to input file, path to output file, minimum points value and epsilon value.

```
input.path=file://path_to_input
output.path=file://path_to_output
dbscan.eps=5
dbscan.min.points=5
```

Figure 20: Properties file example for running DBScan on Apache Spark

Performance Evaluation:

We performed an experiment on 500000 randomly generated 2D points to test the performance of distributed DBScan on multiple cores. We used epsilon and min points value as 5. The test is done with 1, 10, 36, 72 cores, and Figure 19 illustrates how execution time varies as the number of cores increase.

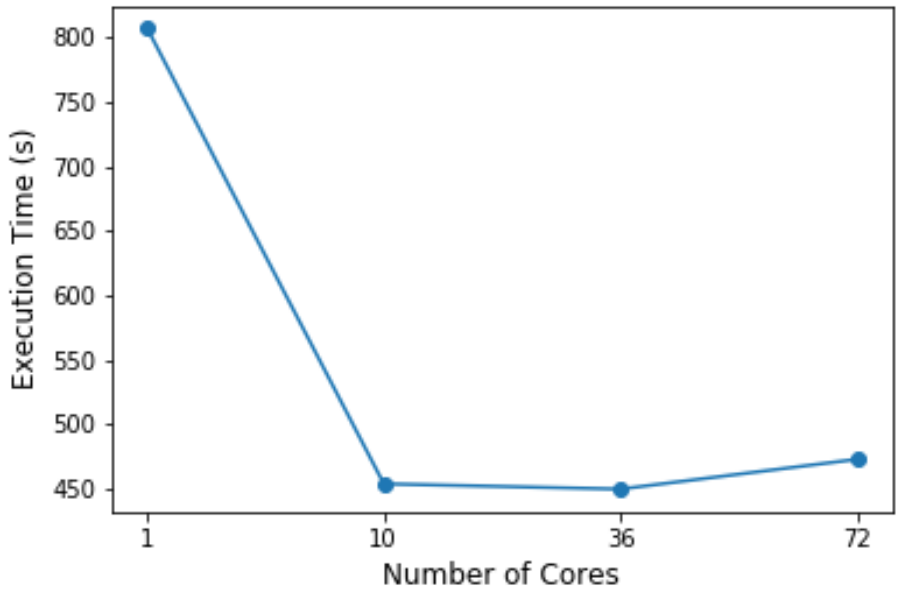


Figure 21: DBScan on Apache Spark: Execution time vs Number of cores

As the dataset size is small, after sometime increasing the number of cores doesn't improve the performance significantly. It could also increase the execution time, where communication time could become longer than whole processing time.

5.2 Integration with EOPEN Platform

Out of the clustering methods described in Section 3 and Section 4, the clustering of non-EO textual data, i.e. texts from Twitter posts, has been also integrated in the EOPEN system. First, a service that has been implemented to detect clusters of tweets (also referred as topics) is described, along with its input parameters, its output results and its variations. Secondly, follows the description of how topics are visualised in the EOPEN Dashboard by using the aforementioned service. Thirdly, we present an alternative workflow of this task, which is currently under development.

5.2.1 Text Clustering Service

The text clustering service is able to discover topics inside a set of tweets, by grouping them accordingly to the similarity of their content. The service is implemented in JAVA and offers three options: 1) find topics in x most recent tweets, 2) find topics in x previous days, 3) find topics in tweets that contain a specific keyword. The input parameters change respectively, but common parameters in all three variations are the language and the use case of interest. In detail, these are the different ways to call the service:

1. *http://.../api/topicDetection/usingRecent?language=<>&pilot=<>&num=<>*,
   e.g.
   http://.../api/topicDetection/usingRecent?language=English&pilot=Floods&num=400,
   where the service looks for topics in the 400 most recent English tweets about floods.
2. *http://.../api/topicDetection/usingDays?language=<>&pilot=<>&days=<>*,
   e.g.
   http://.../api/topicDetection/usingDays?language=English&pilot=Floods&days=10,
   where the service looks for topics in the English tweets about floods that were posted in the 10 last days.
3. *http://.../api/topicDetection/usingKeyword?language=<>&pilot=<>&keyword=<>*,
   e.g.
   http://.../api/topicDetection/usingKeyword?language=English&pilot=Floods&keyword=storm, where the service looks for topics in English tweets about floods that contain the keyword “storm”.

Then, the service communicates with the MongoDB database where collected social media is stored and the tweets that satisfy the input criteria are printed to text files. This set of files is passed as input to an R script, which performs a combination of density-based clustering with LDA (see subsection 3.1). The results of the R script are converted to JSON format by the service and involve: 1) the ID of each detected cluster, 2) the words most frequently mentioned in the tweets of each cluster, and 3) the IDs of the tweets that comprise each cluster. A part of the response of the service (for one cluster) can be seen in Figure 22.

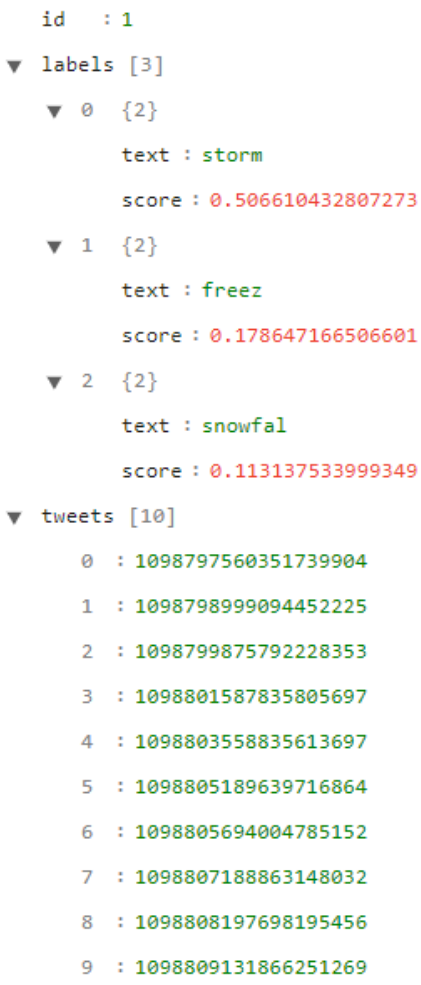


Figure 22: A detected topic as presented in the response of the text clustering service

The service has been dockerised and deployed in the EOPEN platform and it can be accessed by processes and the User Portal.

5.2.2 Visualisation of topics in the EOPEN Dashboard

In order to offer to end users the ability to use the above service in a friendly way, it has been included in the EOPEN Dashboard. A core technical requirement of the EOPEN platform defines that the end users must be able to customise their own dashboard, so every EOPEN module is implemented in the Vue.js[3] framework as a separate component, which can be

[3] https://vuejs.org/

added, shaped, positioned and removed from the interface. Thus, three new components have been created to exploit the text clustering service, while trying to reuse what has been already created for displaying the collected tweets.

The first component (Figure 23), named “Topics Filter”, is a simple form where the user can select on which collection of tweets the text clustering will run. Each collection is defined by use case and language and corresponds to a different collection in the MongoDB database. It should be noted that the option to search for topics in the Korean tweets about food security is omitted, because their special characters cannot be handled yet. When clicking on the “Get recent topics” button, the text clustering service is called for the 400 most recent tweets of the selected collection. This number is not adjustable by the user and is set to 400, because the service achieves a good response time with such a size (approximately 3 seconds). This is a limitation can be addressed with an alternative implementation (see Section 5.2.3).

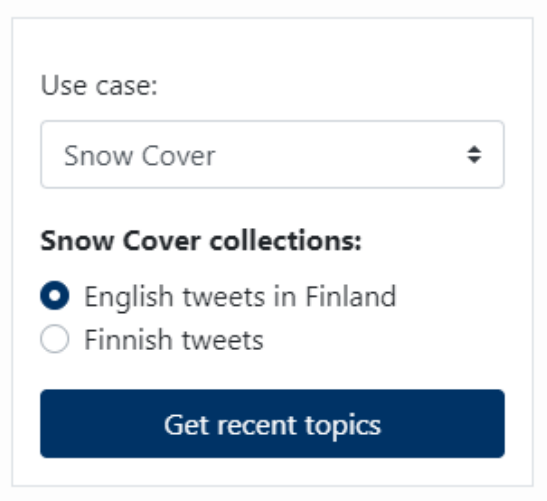


Figure 23: Topics Filters component

The results of the service are visualised to the user in a second component (Figure 24), named “Topics List”. Each of the detected clusters is presented as a word cloud, i.e. an illustration of single words where their importance is shown with font size. The words displayed derive from the attribute “labels” of the service’s response, while the word clouds are created with the “vue-wordcloud” component[4]. In Figure 24 an example of word clouds can be seen; the term “snow” is present in all topics, since all tweets from this collection refer to that, but then every cloud shows a different topic regarding snow. As far as it concerns the IDs of the tweets that belong to each cluster, they are saved in the background and are used for the following visualisation.

[4] https://github.com/feifang/vue-wordcloud

Figure 24: Topics List component

Every time a topic is clicked, the complete set of tweets that comprises it appears in a third component (Figure 25), named “Topic Tweets List”. This component reuses a ready Vue.js template to display a tweet, which has been previously created for presenting the collected tweets.

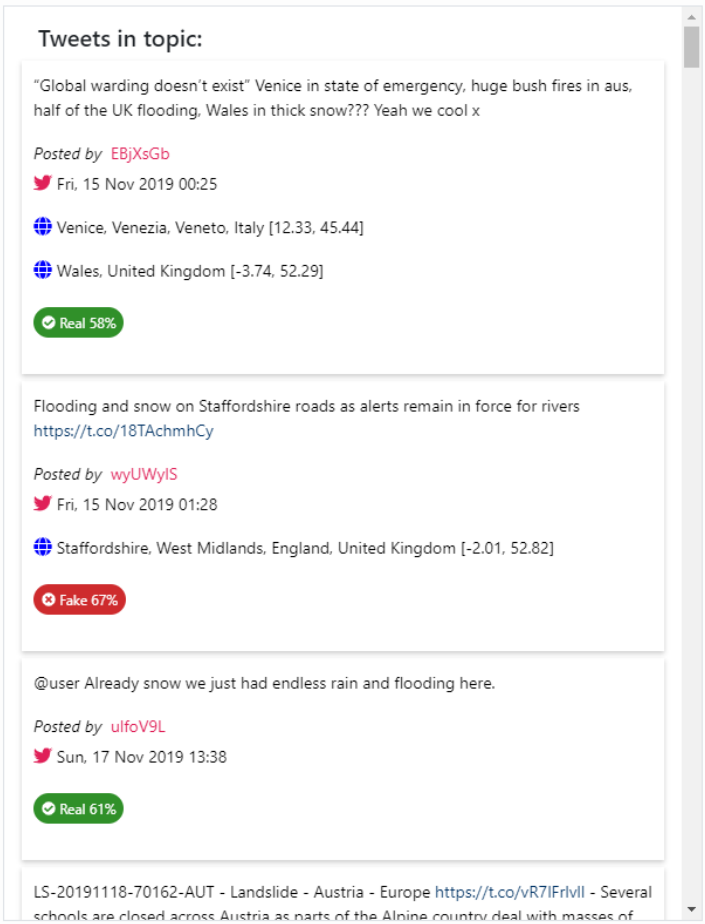


Figure 25: Topic Tweets List component

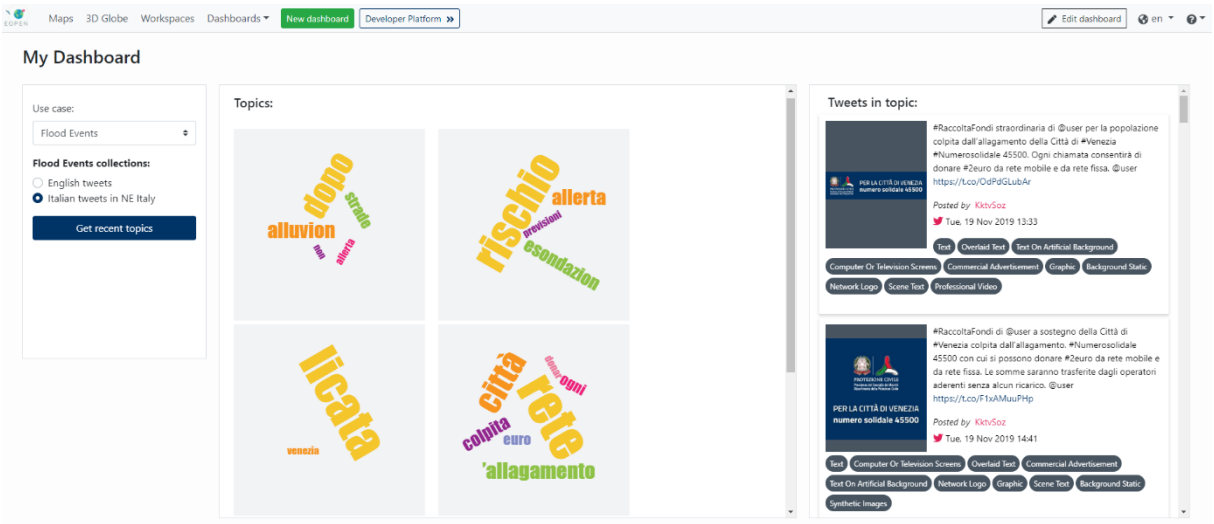


Figure 26: A custom dashboard that includes all the text clustering components

As already mentioned, the users are able to modify their dashboard as they wish, so in Figure 26 a custom dashboard can be seen, containing all three components that are relevant to the text clustering, i.e. the filters, the topics list, and the list of tweets per topic.

5.2.3 An alternative workflow for the text clustering

The current implementation, where the text clustering algorithm runs every time the service is called, shows a certain limitation: the end user always has to wait for the clustering to be completed in order to see the topics. For this reason, the input is also limited to 400 most recent tweets, so as to expect a good response time from the service.

Based on this limitation, a new workflow for the task of text clustering is proposed () and is currently under development. Instead of calling the text clustering service every time on the fly, a process is responsible to get the topics in the background (for more details on processes of the EOPEN platform, refer to D6.3). The process, scheduled to run every six hours, calls the text clustering service for each collection of tweets (e.g. Italian tweets about floods, Finnish tweets about snow, etc.) and stores the detected clusters in a database. From the perspective of the EOPEN Dashboard, when the user will click the button to get most recent topics, a simple API will query the database for the latest records and immediately show the results.

The final status of this alternative implementation will be reported in D4.4 (M33).

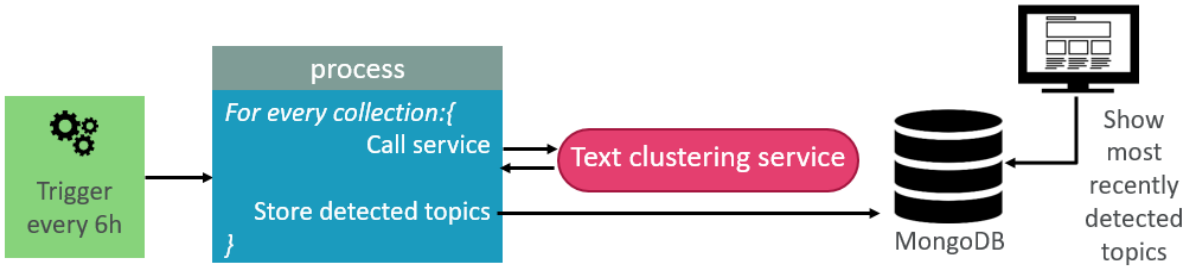


Figure 27: The future workflow of text clustering in EOPEN

6 CONCLUSIONS

In this deliverable we have presented the clustering techniques on both EO and non-EO data, which is finally offered as a service to end users.

In Section 2, we have defined the problem of estimating the number of clusters, presented solutions proposed by state of the art works and proposed a novel technique, namely DBSCAN-Martingale. Section 3 focused on the clustering of non-EO data, i.e. social media data and specifically Twitter posts. For the textual data, after a review of state of the art algorithms, a combination of density-based clustering and LDA has been proposed. With respect to clustering of images, we presented a proof-of-concept for unsupervised image clustering utilizing artificial neural networks, such as an AutoEncoder to efficiently perform clustering on a HPDA infrastructure using Keras with TensorFlow. Results are then mapped onto a two-dimensional plane, where the distance between images represents their similarity. Each identified group is then described through a representative thumbnail. In Section 4, we have presented the work done for segmentation of EO images, including studying the state of the art algorithms, implementing K-means and hierarchical K-means on pixels and, finally, getting the primary outcome. Section 4 also included a new approach of clustering satellite imagery patches in an unsupervised way. Section 5 described the procedures for implementing the algorithms on HPC and HPDA infrastructure, together with an evaluation of processing time and cost in dollars in comparison to the number of cores. Section 5 continued with the implementation of a text clustering service, its integration in the EOPEN platform and dashboard, and a description of its next version.

The main achievements can be summarized as:

- The novel methodology of DBSCAN-Martingale to estimate the number of clusters has been introduced.
- A text clustering technique that combines DBSCAN-Martingale and LDA has been proposed. Its implementation and integration to the EOPEN platform, as well as the visualisation of its results on the EOPEN dashboard have been described.
- A prototype of Twitter images clustering has been performed on a sample dataset consisted of 1000 images crawled from Twitter posts.
- A prototype of performing pixel clustering on EO images in an unsupervised way to do image segmentation has been provided.
- A new approach of clustering satellite image patches in an unsupervised way.
- The clustering algorithms have been implemented on HPC and HPDA infrastructure utilizing Apache Spark, TensorFlow and so on with faster outcome. An analysis of processing time and cost in relation to the number of cores has been carried out.

The work that remains to be done for this task until the end of the project will focus on the implementation of the clustering technique for EO patches in a parallel manner and the completion of the respective experiments.

7 REFERENCES

Aggarwal, C. C., Zhai, C., 2012. *"A survey of text clustering algorithms"*, Mining text data, Springer, Boston, MA, pp. 77-128.

Ankerst, M., Breunig, M. M., Kriegel, H. P., Sander, J., 1999. *"OPTICS: ordering points to identify the clustering structure"*, ACM Sigmod record, Vol. 28, No. 2, pp. 49-60.

Antoniou, I., Gialampoukidis, I., Ioannidis, E., 2015. *"Age and time operator of evolutionary processes"*, International Symposium on Quantum Interaction, Springer, pp. 51–75.

Apache Hadoop (2019), *Hadoop 1.2.1 Documentation.* [Online] Available: *https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html*

Blei, D.M., Jordan, M.I., 2003. *"Modeling annotated data"*, Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp. 127–134.

Boonchoo, T., Ao, X., Liu, Y., Zhao, W., Zhuang, F. and He, Q., 2019. Grid-based DBSCAN: Indexing and inference. Pattern Recognition, 90, pp.271-284.

Bryant, A. and Cios, K., 2018. RNN-DBSCAN: A density-based clustering algorithm using reverse nearest neighbor density estimates. IEEE Transactions on Knowledge and Data Engineering, 30(6), pp.1109-1121.

Cai, Z., Wang, J. and He, K., 2020. Adaptive Density-Based Spatial Clustering for Massive Data Analysis. IEEE Access, 8, pp.23346-23358.

Campello, R.J., Moulavi, D., Sander, J., 2013. *"Density-based clustering basedon hierarchical density estimates"*, Advances in Knowledge Discovery and Data Mining, Springer, pp. 160–172.

Chen, G., Cheng, Y. and Jing, W., 2019. DBSCAN-PSM: an improvement method of DBSCAN algorithm on Spark. International Journal of High Performance Computing and Networking, 13(4), pp.417-426.

Chen, Y., Tang, S., Bouguila, N., Wang, C., Du, J., Li, H., 2018. *"A fast clustering algorithm based on pruning unnecessary distance computations in DBSCAN for high-dimensional data"*, Pattern Recognition, 83, 375-387.

Chen, Y., Zhou, L., Pei, S., Yu, Z., Chen, Y., Liu, X., Du, J. and Xiong, N., 2019. KNN-BLOCK DBSCAN: Fast Clustering for Large-Scale Data. IEEE Transactions on Systems, Man, and Cybernetics: Systems.

Cray Inc., *Cray Urika-GX Agile Analytics Platform.* [Online] Available: https://www.cray.com/sites/default/files/Cray-Urika-GX-Product-Brief.pdf. Accessed on: Jan. 18 2019.

Cray Inc., *Cray XC40: Scaling Across the Supercomputer Performance Spectrum.* [Online] Available: https://www.cray.com/sites/default/files/resources/CrayXC40Brochure.pdf. Accessed on: Jan. 18 2019.

de Berg, M., Gunawan, A. and Roeloffzen, M., 2019. Faster DBScan and HDBScan in low-dimensional Euclidean spaces. International Journal of Computational Geometry & Applications, 29(01), pp.21-47.

Deng, C., Song, J., Cai, S., Sun, R., Shi, Y. and Hao, S., 2018. K-DBSCAN: an efficient density-based clustering algorithm supports parallel computing. International Journal of Simulation and Process Modelling, 13(5), pp.496-505.

Diao, K., Liang, Y. and Fan, J., 2018, August. An Improved DBSCAN Algorithm Using Local Parameters. In International CCF Conference on Artificial Intelligence (pp. 3-12). Springer, Singapore.

Ding, H. and Yang, F., 2020. On Metric DBSCAN with Low Doubling Dimension. arXiv preprint arXiv:2002.11933.

Dockhorn, A., Braune, C., Kruse, R., 2016. *"Variable density based clustering"*, Computational Intelligence (SSCI), 2016 IEEE Symposium Series on, IEEE, pp. 1–8.

Ekseth, O.K. and Hvasshovd, S.O., 2019. How an optimized DBSCAN implementation reduces execution-time and memory-requirements for large datasets. Proceedings of the Patterns.

Ester, M., Kriegel, H. P., Sander, J., Xu, X., 1996. *"A density-based algorithm for discovering clusters in large spatial databases with noise"*, Kdd, Vol. 96, No. 34, pp. 226-231.

Faanes, G., Bataineh, A., Roweth, D., Court, T., Froese, E., Alverson, B., ... & Reinhard, J. (2012, November). Cray cascade: a scalable HPC system based on a Dragonfly network. In SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (pp. 1-9). IEEE.

Galán, S.F., 2019. Comparative evaluation of region query strategies for DBSCAN clustering. Information Sciences, 502, pp.76-90.

Gan, J., Tao, Y., 2015. *"Dbscan revisited: mis-claim, un-fixability, and approximation"*, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, pp. 519–530.

Getz, G., Levine, E., Domany, E., 2000. *"Coupled two-way clustering analysis of gene microarray data"*, Proceedings of the National Academy of Sciences 97, 12079–12084.

Gialampoukidis, I., Gustafson, K., Antoniou, I., 2013. *"Financial Time Operator for random walk markets"*, Chaos, Solitons & Fractals, 57, 62-72.

Gialampoukidis, I., Vrochidis, S., Kompatsiaris, I., 2016. *"A hybrid framework for news clustering based on the DBSCAN-Martingale and LDA"*, International Conference on Machine Learning and Data Mining in Pattern Recognition, pp. 170-184, Springer, Cham.

Gialampoukidis, I., Vrochidis, S., Kompatsiaris, I., Antoniou, I., 2017. *"Topic detection using the dbscan-martingale and the time operator"*, Skiadas CH, editor, Proceedings of the 17th Applied Stochastic Models and Data Analysis International Conference with the 6th Demographics Workshop ASMDA2017, 2017 June 6-9, London, UK, ISAST, p. 387-95.

Gialampoukidis, I., Vrochidis, S., Kompatsiaris, I., Antoniou, I., 2019. *"Probabilistic density-based estimation of the number of clusters using the DBSCAN-martingale process"*, Pattern Recognition Letters, 123, 23-30.

Girvan, M., Newman, M.E., 2002. *"Community structure in social and biological networks"*, Proceedings of the national academy of sciences 99, 7821–7826.

Gong, Y., Sinnott, R.O. and Rimba, P., 2018, June. Rt-dbscan: Real-time parallel clustering of spatio-temporal data using spark-streaming. In International Conference on Computational Science (pp. 524-539). Springer, Cham.

Hahsler, M., Piekenbrock, M. and Doran, D., 2019. dbscan: Fast density-based clustering with r. Journal of Statistical Software, 25, pp.409-416.

Han, D., Agrawal, A., Liao, W.K. and Choudhary, A., 2018, December. Parallel DBSCAN Algorithm Using a Data Partitioning Strategy with Spark Implementation. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 305-312). IEEE.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504-507.

Hou, J., Lv, C., Zhang, A. and Xu, E., 2019, September. Merging DBSCAN and Density Peak for Robust Clustering. In International Conference on Artificial Neural Networks (pp. 595-610). Springer, Cham.

Hu, X., Liu, L., Qiu, N., Yang, D. and Li, M., 2018. A MapReduce-based improvement algorithm for DBSCAN. Journal of Algorithms & Computational Technology, 12(1), pp.53-61.

Huang, M., Yan, Y., Xu, L. and Ye, L., 2020. Using Warshall to Solve the Density-linked Density Clustering Algorithm. American Journal of Applied Mathematics, 8(1), pp.11-16.

Ibrahim, R. and Shafiq, M.O., 2018, June. Towards a New Approach for Empowering the MR-DBSCAN Clustering for Massive Data Using Quadtree. In 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS) (pp. 91-98). IEEE.

Jang, J. and Jiang, H., 2018. DBSCAN++: Towards fast and scalable density clustering. arXiv preprint arXiv:1810.13105.

Johnson, T., Prabhu, K., Parvatkar, S., Naik, A. and Temkar, P., 2018. The Bisecting Min Max DBSCAN Algorithm.

Khan, W. (2013). Image segmentation techniques: A survey. *Journal of Image and Graphics*, *1*(4), 166-170.

Kim, J.H., Choi, J.H., Yoo, K.H. and Nasridinov, A., 2019. AA-DBSCAN: an approximate adaptive DBSCAN for finding clusters with varying densities. The Journal of Supercomputing, 75(1), pp.142-169.

Kulis, B., Jordan, M.I., 2011. *"Revisiting K-means: new algorithms via Bayesian nonparametrics"*, arXiv preprint arXiv:1111.0352.

Kumari, A., Shrivastava, V. and Pandey, A., 2019. Reduction of DBSCAN Time Complexity for Data Mining Using Parallel Computing Techniques.

Lai, W., Zhou, M., Hu, F., Bian, K. and Song, Q., 2019. A new DBSCAN parameters determination method based on improved MVO. IEEE Access, 7, pp.104085-104095.

Lalitha, M., Kiruthiga, M., & Loganathan, C. (2013). A survey on image segmentation through clustering algorithm. *International Journal of Science and Research*, *2*(2), 348-358

Li, H., Liu, X., Li, T. and Gan, R., 2020. A novel density-based clustering algorithm using nearest neighbor graph. Pattern Recognition, 102, p.107206.

Li, J. and Chen, Y.B., 2018. Improved DBSCAN Algorithm Based on Natural Neighbors. Modern Computer, 2018(13), p.2.

Li, J.W., Han, X.Q., Jiang, J.W., Hu, Y. and Liu, L., 2020. An Efficient Clustering Method for Dbscan Geographic Spatio-Temporal Large Data with Improved Parameter Optimization. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 42, pp.581-584.

Li, S.S., 2020. An Improved DBSCAN Algorithm Based on the Neighbor Similarity and Fast Nearest Neighbor Query. IEEE Access, 8, pp.47468-47476.

Lin, P., Hong, Z., Feng, W., Li, Y. and Wu, L., 2019, October. Design and Implementation of an Improved DBSCAN Algorithm. In 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) (pp. 1834-1839). IEEE.

Liu, Q., Sun, Y., Wang, C., Liu, T., Tao, D., 2017. *"Elastic net hypergraph learning for image clustering and semi-supervised classification"*, IEEE Transactions on Image Processing 26, 452–463.

Liyang, L., Hongzhen, S.O.N.G., Shen, W.A.N.G. and Jinyu, L., 2019. Parallel Implementation of DBSCAN Algorithm Based on Spark.

Louhichi, S., Gzara, M., Abdallah, H.B., 2014. *"A density based algorithm for discovering clusters with varied density"*, Computer Applications and Information Systems (WCCAIS), 2014 World Congress on, IEEE, pp. 1–6.

Lu, S., 2019. Self-adaption grey DBSCAN clustering. arXiv preprint arXiv:1912.11477.

Luchi, D., Rodrigues, A.L. and Varejão, F.M., 2019. Sampling approaches for applying DBSCAN to large datasets. Pattern Recognition Letters, 117, pp.90-96.

Mai, S.T., Assent, I., Jacobsen, J. and Dieu, M.S., 2018. Anytime parallel density-based clustering. Data Mining and Knowledge Discovery, 32(4), pp.1121-1176.

Mai, S.T., Dieu, M.S., Assent, I., Jacobsen, J., Kristensen, J., Birk, M., 2017. *"Scalable and interactive graph clustering algorithm on multicore cpus"*, Data Engineering (ICDE), 2017 IEEE 33rd International Conference on, IEEE, pp. 349–360.

Mathur, V., Mehta, J. and Singh, S., 2019. HCA-DBSCAN: HyperCube Accelerated Density Based Spatial Clustering for Applications with Noise. arXiv preprint arXiv:1912.00323.

Moftah, H. M., Azar, A. T., Al-Shammari, E. T., Ghali, N. I., Hassanien, A. E., & Shoman, M. (2014). Adaptive K-means clustering algorithm for MR breast image segmentation. *Neural Computing and Applications*, *24*(7-8), 1917-1928.

Moumtzidou, A., Mironidis, T., Markatopoulou, F., Andreadis, S., Gialampoukidis, I., Galanopoulos, D., Ioannidou, A., Vrochidis, S., Mezaris, V.,Kompatsiaris, I., et al., 2017.

"Verge in vbs 2017", International Conference on Multimedia Modeling, Springer, pp. 486–492.

Mu, B., Dai, M. and Yuan, S., 2020, January. DBSCAN-KNN-GA: a multi Density-Level Parameter-Free clustering algorithm. In IOP Conference Series: Materials Science and Engineering (Vol. 715, No. 1, p. 012023). IOP Publishing.

Newman, M., Girvan, M., 2004. *"Finding and evaluating community structure in networks"*, Physical Review E 69, 026113.

Pandey, S., Samal, M. and Mohanty, S.K., 2020. An SNN-DBSCAN Based Clustering Algorithm for Big Data. In Advanced Computing and Intelligent Engineering (pp. 127-137). Springer, Singapore.

Petkos, G., Schinas, M., Papadopoulos, S., Kompatsiaris, Y., 2017. *"Graph-based multimodal clustering for social multimedia"*, Multimedia Tools and Applications, 76(6), 7897-7919.

Puzicha, J., Hofmann, T., & Buhmann, J. M. (1999, June). Histogram clustering for unsupervised image segmentation. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)* (Vol. 2, pp. 602-608). IEEE.

Romesburg, C. (2004). *Cluster analysis for researchers*. Lulu. com.

Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234-241). Springer, Cham.

Sarma, A., Goyal, P., Kumari, S., Wani, A., Challa, J.S., Islam, S. and Goyal, N., 2019, September. µDBSCAN: An Exact Scalable DBSCAN Algorithm for Big Data Exploiting Spatial Locality. In 2019 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 1-11). IEEE.

Schneider, J., Vlachos, M., 2013. *"Fast parameterless density-based clustering via random projections"*, Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, ACM, pp.861–866.

Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X., 2017. *"Dbscan revisited, revisited: why and how you should (still) use dbscan"*, ACM Transactions on Database Systems (TODS) 42, 19.

Shahriar, N., Al Faisal, S.A., Pinjor, M.M., Rafi, M.A.S.Z. and Sarkar, A.R., 2019, December. Comparative Performance Analysis of K-Means and DBSCAN Clustering algorithms on various platforms. In 2019 22nd International Conference on Computer and Information Technology (ICCIT) (pp. 1-6). IEEE.

Shibla, T.P. and Kumar, K.S., 2018, June. Improving Efficiency of DBSCAN by Parallelizing kd-Tree Using Spark. In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 1197-1203). IEEE.

Shiqiu, Y. and Qingsheng, Z., 2019, October. DBSCAN Clustering Algorithm Based on Locality Sensitive Hashing. In Journal of Physics: Conference Series (Vol. 1314, No. 1, p. 012177). IOP Publishing.

Song, H. and Lee, J.G., 2018, May. RP-DBSCAN: A superfast parallel DBSCAN algorithm based on random partitioning. In Proceedings of the 2018 International Conference on Management of Data (pp. 1173-1187).

Starczewski, A. and Cader, A., 2019, June. Determining the Eps Parameter of the DBSCAN Algorithm. In International Conference on Artificial Intelligence and Soft Computing (pp. 420-430). Springer, Cham.

Teh, Y.W., Jordan, M.I., Beal, M.J., Blei, 2004. *"Sharing clusters among related groups: Hierarchical Dirichlet processes"*, NIPS, pp. 1385–1392.

TensorFlow (2019), An end to end open source machine learning platform. [Online] Available: https://www.tensorflow.org

Tyercha, E.R., Kazmaier, G.S., Gildhoff, H., Pekel, I., Volker, L. and Grouisborn, T., SAP SE, 2019. Hilbert curve partitioning for parallelization of DBSCAN. U.S. Patent 10,318,557.

Wang, Y., Gu, Y. and Shun, J., 2019. Theoretically-Efficient and Practical Parallel DBSCAN. arXiv preprint arXiv:1912.06255.

Weng Lilian (2019): From AutoEncoder to Beta-VAE. [Online] Available: https://lilianweng.github.io/lil-log/2018/08/12/from-AutoEncoder-to-beta-vae.html

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37-52.

Xia, X., & Kulis, B. (2017). W-net: A deep model for fully unsupervised image segmentation. *arXiv preprint arXiv:1711.08506*.

Yang, K., Gao, Y., Ma, R., Chen, L., Wu, S. and Chen, G., 2019, April. DBSCAN-MS: Distributed Density-Based Clustering in Metric Spaces. In 2019 IEEE 35th International Conference on Data Engineering (ICDE) (pp. 1346-1357). IEEE.

Yu, H., Chen, L., Yao, J. and Wang, X., 2019. A three-way clustering method based on an improved DBSCAN algorithm. Physica A: Statistical Mechanics and its Applications, 535, p.122289.

Zhou, G.J., 2018. Research on Parallel Design of DBSCAN Clustering Algorithm in Spatial Data Mining. DEStech Transactions on Engineering and Technology Research, (ecar).

Zheng, X., Cai, D., He, X., Ma, W. Y., & Lin, X. (2004, October). Locality preserving clustering for image database. In *Proceedings of the 12th annual ACM international conference on Multimedia* (pp. 885-891). ACM.

Zhu, L., Zhu, J., Bao, C., Zhou, L., Wang, C. and Kong, B., 2018, December. Improvement of DBSCAN Algorithm Based on Adaptive Eps Parameter Estimation. In Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence (pp. 1-7).