IMPERIAL COLLEGE LONDON

Department of Earth Science and Engineering

Machine Learning for Automatic Facies Classification from Tomography Models

By

Nitchakul Pipitvej

Supervised by Dr. Michele Paulatto

An independent research project report submitted for the MSc in Applied Computational Science and Engineering, Module ACSE-9

August 2019

Github Alias: wafflescore
Email: np2618@ic.ac.uk
Date: 30 August 29, 2019

Abstract

Identifying lithology from geophysical data is important in the study of geological processes. I investigate how lithological classification can be aided by using machine learning automatic clustering methods. There are many available methods and various combinations of rock properties that can be used to train each model. In this report, Self-Organization Maps (SOMs), HDBSCAN, K-Means, and Fuzzy-C-Mean classification were tested. The physical parameters available are compressional wave velocity (V_p), shear wave velocity (V_s), density, V_p/V_s ratio, compressional wave attenuation (Q_p^{-1}), and shear wave attenuation (Q_s^{-1}). A total of five different sets of physical parameters were used, which are 1) all the available physical properties 2) all the available physical properties excluding the topological location 3) V_p, V_s, V_p/V_s 4) V_p, Q_p^{-1}, density and 5) V_p, V_s. The result shows that HDBSCAN and SOMs work well for the first and second case, with SOMs being more susceptible to noises. FCM and K-means behave similarly and work better when there is less physical parameter present. Combining SOMs with other classification methods can introduce small improvements but greatly increases the computational time.

Keywords: Machine Learning, Clustering method, Facies Clustering, Self-Organizing map

Introduction

In geology, identifying the type of rocks in the subsurface can be important for understanding the tectonic evolution and the possible presence of mineral resources. Often, a site region will harbour multiple rock types but not all of them are the object of interest. The conventional way of determining the lithology by laboratory studies of rock cores retrieved from drilling boreholes, while yielding an accurate result, is very costly and time-consuming. Another method of retrieving rock data is to use the well logs, which is the 1-dimensional data retrieved from drilling boreholes. This is faster and cheaper than laboratory measurements. These data are relatively high-resolution (1m – 10m resolution) and can then be fed into machine learning methods to determine the lithology [1].

Apart from well logs, there are other means to retrieve the lithology properties as well, an example is seismological measurements to captures the V_p and V_s of an area. These data can be 2- or 3-dimensional however will have a lower resolution (500m to 4km resolution) compared to 1d borehole measurements.

The objective of the project is to try to determine lithology from physical properties measured in 2D with geophysical methods. Various methods of machine learning techniques have been used in this field and have proved promising. It is the aim of this project to test several classification methods and see how well they perform on different sets of available properties and determine the best method and parameters to classify the lithology.

Based on multiple studies, the use of Self-Organizing map with watershed segmentation can accurately cluster similar rock types together while using only a small number of available physical properties[2][3][4]. An implementation of SOMs using resistivity, V_p, and V_p gradient [2] can identify additional clusters compared to the previous study that uses only two properties [3]. Similarly, SOMs application on V_p, V_s, and V_p/V_s ratio is also able to achieve a good result [4]. Using real data inputs, these studies were bound by the limitation of available properties. In our study, the data will be a synthetic model of the Earth, thus making it possible to generate and make use of a higher number of properties.

Many studies focused on the magnetotelluric and seismic data [3][4][5][6], though there may be other notable relations in other unexplored parameters that may be able to derive a more accurate model for lithology classification.

The tested were performed with the following sets of physical parameters:

1. V_p, V_s, density, V_p/V_sratio, Q_p^{-1}, Q_s^{-1}, location, and depth.
2. V_p, V_s, density, V_p/V_s ratio, Q_p^{-1}, and Q_s^{-1}
3. V_p, V_s, and V_p/V_s
4. V_p, Q_p^{-1}, and density
5. V_p and V_s.

Speculation on this was by adding more properties, the clustering method should be able to identify clusters more accurately. The returned result should be able to help the user identify clusters more easily and should require only a small amount of intervention from the user.

Unsupervised learning methods are incorporated into this project. The convention of unsupervised learning is that there is no prior knowledge about the label of each datapoint [7]. The task to utilize unsupervised learning is challenging in the aspect that there is no known information to minimize the error by and will usually use the data density and distance as its main reference [8].

Methods

Classification Method: K-Mean

K-Mean is an unsupervised classification method, the only parameter that was needed is the number of clusters. It was preferable as it is an easy method to be implemented and does not require a long computational time. It can classify a multivariant input into the user-specified number of clusters [9].

The algorithm performs as follow:

1. Randomly assign K datapoints to be the cluster centres.
2. Computes the distance of all the other datapoints to each cluster centres.
3. Assign each datapoints to the nearest cluster.
4. Update the cluster centres by computing the mean of all the datapoints in that cluster.
5. Repeat until the cluster centres converge.

One drawback of this method was that we need to know the initial number of clusters to correctly classifies the inputs. A popular approach to find this number is the elbow method, which iterates the number of cluster and plots it to find the point where the gradient of the mean distance to the centroid changes the most. That point demonstrates the number of optimal clusters in the observed dataset. Still, [10] also states that the elbow may not always be easily observed. This thus leads to another verification method named Silhouette Score, which will be discussed later in the verification section.

Classification Method: Fuzzy C Mean Clustering

Fuzzy C Mean (FCM) is similar to K-mean clustering but with the capability that each datapoint can belong to multiple clusters [11]. The membership degree that a datapoint can belong to a cluster varies between 0 and 1, in which the sum of all membership degrees of that datapoint is 1.

The algorithm performs as follow:

1. Randomly assign the membership degree of each datapoint.
2. Computes the centroid defined as [12]:

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}}$$

   Where i is the centroid index, u is the membership degree of being in cluster i, and j is the datapoint index. The dimension of the datapoint is m dimension.
3. Update the membership degree defined as:

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} (\frac{d_{ij}}{d_{kj}})^{\frac{2}{(m-1)}}}$$

   Where d_{ij} is the distance between datapoint j and centroid i.
4. Repeat until the cluster centroid converges.

This clustering method could help identify portions of the observed data that contain noise and anomalies, allowing the user an opportunity to look into each possibility instead of just one strict classification.

Classification Method: HDBSCAN

HDBSCAN is a clustering method based on DBSCAN [13]. This algorithm looks at the density of the available datapoints. The area that has a high density of datapoints will be clustered together. While the concept seems similar to K-mean clustering, it is more robust since the shape of the cluster is not limited to a convex shape. This is due to how the distance between each cluster member is calculated, while K-mean uses the mean of all the datapoints in the cluster, DBSCAN computes the distance between each point in the cluster and expands the cluster size depending on the user's defined distance. DBSCAN takes in two important parameters, the minimum number of points (*min_sample*) and epsilon (ε). The minimum number of points defines when a datapoint can be considered as a cluster's core, whereas the ε is the distance in which the core expands. In **Figure 1**, point A was originally the cluster core point, within a distance of epsilon, four more datapoints can be found. From those datapoints are then found the neighbours within **ε** distance and so on. A point will be considered a cluster core when there is more than *min_sample* of datapoints within the distance of ε. Datapoints that lie outside radius of the core points are then labelled as noise.

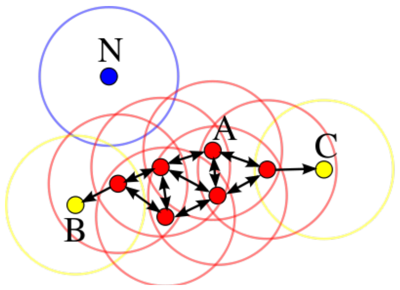

Figure 1 DBSCAN cluster core identification

Ignoring the noise, the cluster cores and neighbours are then used to compute the minimum spanning tree and the single linkage dendrogram. The HDBSCAN build up from this by adding a parameter, minimum cluster size (*min_cluster_size*), the dendrogram that has fewer datapoints than the minimum cluster size will be joined together to form a cluster.

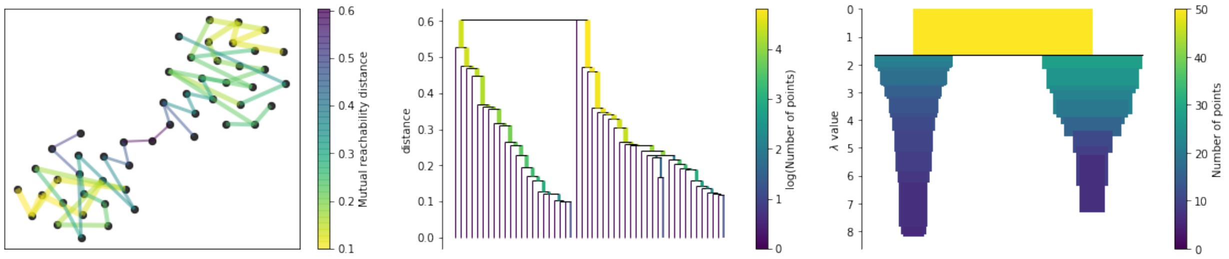

Figure 2, DBSCAN's (left)minimum spanning, (middle) single linkage dendrogram, (right) condensed dendrogram.

The *HDBScan* notebook focuses on this clustering method and tries to optimize for the ideal parameter suitable for data model M5a and M5b.

Classification Method: Self-Organizing Maps (SOMs)

SOMs is an unsupervised learning artificial neural network introduced by Kohonen [14]. Its notable feature is its ability to take in an input of unlimited dimension and represent them as a map of lower dimension, which corresponds to the project's input of having multiple dimensions in the input. This method can be considered as a dimensional reduction approach. This output can then be processed and analyzed to either manually or automatically assign the lithological class of each observed data cluster via some clustering method, then mapped back to the original topography.

The algorithm performs as follow:

1. Initialize the Kohonen layer (hidden layer) of the user's input size with random neuron weights. While the size of the Kohonen layer can be manually input, it is advised as a rule of thumb to use $5 * \sqrt{N}$, where N is the number of total datapoints [15].
2. Compute each datapoint's Best Matching Unit (BMU) using the Euclidian distance between the data point and the neurons' weight. The neuron with the least distance will be that datapoint's BMU or the winning neuron.
3. Update the Kohonen layer's weight using:

$$m_k(t+1) = m_k(t) + \epsilon\alpha(t)[x(t) - m_k(t)]$$

   where $m_k(t)$ is the neuron weight of node k at iteration t, α is the learning rate, and ϵ is the neighbourhood function. This is done for each of the input datapoints.
   Note, there are many neighbourhood functions available, though in this implementation, the gaussian neighbourhood function is used.

$$\epsilon(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-m_x}{\sigma})^2}$$

   where σ is the spread of the neighborhood function, and m_x is the winning neuron of datapoint x.
4. Repeat from step 2 until the Kohonen layer converges.

With this, the weight map of the SOMs is complete. To use SOMs as a clustering method, a further implementation must be done. Using the Kohonen layer, a distance map can be calculated by finding the normalized sum of the distance between each neuron's weight and its neighbour's.

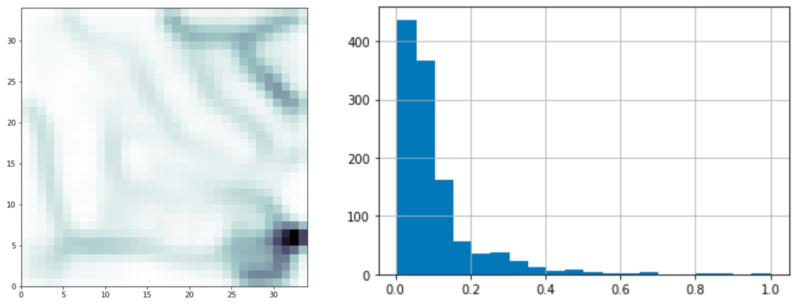
Figure 3 (left) SOMs' distance map and (right) its histogram.

Once the distance map is acquired, a watershed segmentation, an algorithm widely used for image segmentation [16], is performed to identify cluster label for each part of the SOMs neuron map [17]. There are two methods used here:

a. Dynamically compute the threshold mask of the map. By identifying the size of the neighbour to consider, a gaussian filter is applied.
b. Fixed threshold based on the distance map's histogram bin values.

Once the watershed segmentation is done, the distance map will be divided into segments or clusters. There is still a problem since the masking area are not populated with a class. To populate them, K-nearest neighbour (where K = 1 and K = 5) and random walk were applied.

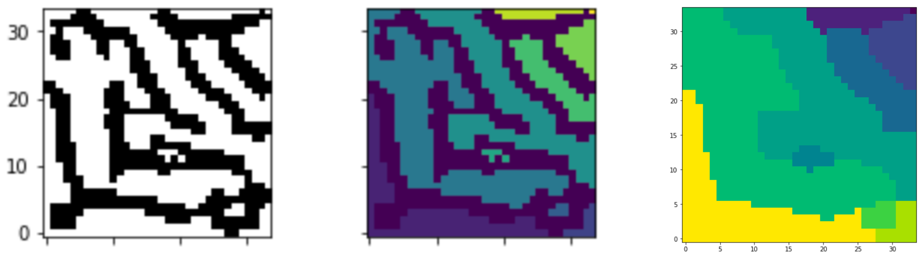
Figure 4 Distance map's (left) mask, (mid) cluster labelled, and (right) populated mask label.

There are many variations of segmentation done in the backend of this algorithm, though will only return the result(s) that harbours the best Calinski Harabaz Score and Silhouette Score. If needed, the user can specify for the function *eval_ws* to return all variation by setting the flag in *re_all* to *True.*

Since SOMs produced a map of a lower dimension of the input data, this map could then be fed into other methods of classification.

Verification Methods

There are many ways to find the validity of a cluster. These can be classified into two groups: the internal indexes and the external indexes. Internal index methods are evaluation methods that do not need any knowledge of the actual cluster label. This can be used to optimize each classification method's parameters. The ones used in this project are:

1. Sum squared error (SSE): computes the distance of each datapoint to its cluster centre and sums them up. While lower SSE shows that the data belongs to its correct cluster, it may result from having too many clusters. To correctly find the optimized number of clusters, an Elbow plot method may be used. This was done by computing the SSE for an incrementing number of clusters. The ideal number of clusters should lie where the plot of SSE's gradient changes the most. Still, the elbow point may not easily be observed [10].

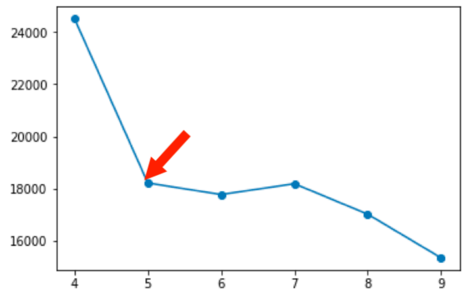

Figure 5 Example of Elbow plot from SSE computation and its ideal number of clusters.

2. Silhouette Score: Computes two main distances, one is the distance a of a datapoint to all the other datapoints in the same cluster. Another distance to compute is the distance b of the datapoint to all the datapoint that is in another cluster, in which this cluster should be the one nearest to its cluster. The formulas are as follow [18]:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \epsilon C_i, i \neq j} d(i,j) \qquad b(i) = \frac{min}{k \neq i} \frac{1}{|C_k|} \sum_{j \epsilon C_k} d(i,j)$$

where a is the sum of the distance from datapoint i to all the other datapoint j in the same cluster. b is the sum of distance from datapoint i to datapoint j in the nearest cluster. The silhouette score is then calculated by the following formula:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}\text{, if } |C_i| > 1; s(i) = 0, \text{if } |C_i| = 1$$

This means that the silhouette score lies between the range of -1 and 1, this number shows the degree of belonging of a datapoint in its cluster. The score closes to 1 means that the datapoint lies far from the cluster boundary, meaning that it is likely in the correct cluster. Score near 0 means that the datapoint is close to the boundary and may or may not be classified in the wrong cluster. Score near -1 means it may be more suitable to be classified in another cluster instead.

3. Calinski Harabaz Score (CH score): The score represents the ratio of dispersion between-clusters (W_k) and dispersion within-cluster (B_k)[19]. Higher CH score signifies that the clusters are more well defined since the dispersion in the same cluster is less than the dispersion between each cluster. The score is computed from the following formula:

$$s(k) = \frac{Tr(B_k)}{Tr(W_k)} \times \frac{N-k}{k-1}$$

where N is the total number of datapoints, and k is the number of clusters.

As for external index, the actual label of the clusters is needed, we can use this to check the result of each clustering algorithm. The method used in this project are:

1. Entropy: measures the uncertainty of the data. It is calculated by comparing the predicted label with the actual label. The formula is:

$$E_j = \sum_i p_{ij} \log(p_{ij})$$

in which E_j is the entropy of class j, p_{ij} is the probability that the predicted cluster j belongs to class i calculated from $p_{ij} = \frac{m_{ij}}{m_i}$ (m_{ij} = number of datapoint in cluster j that belongs in class i, m_j = number of total datapoints in cluster j). The entire dataset's entropy value is then computed by summing each cluster's entropy, weighted with the size of cluster j (n_j) over the total number of data points (n):

$$E = \sum_{j=1}^{m} \frac{n_j}{n} E_j$$

2. Purity: shows how well a cluster is compared to the actual label. A cluster will represent a class (i) that its member belongs to the most. The cluster's purity (P_j) is then calculated from its member that belongs to class i over its total member (n_j).

$$P_j = \frac{1}{n_j} Max_i(n_j^i)$$

The purity of the entire data set is then calculated:

$$P = \sum_{j=1}^{m} \frac{n_j}{n} P_j$$

The best possible purity is 1, being the percentage of datapoint that correctly belong to a cluster. It should be noted that a high value of purity may not always be the best result, cases when there are too many clusters, will have very high purity but cannot represent any meaningful cluster.

Mainly, the internal indexes will be used in the process of parameter tuning and the external indexes for verifying the result.

Software Development

The project was developed using Python 3.6.5 and uses Jupyter Notebook as a preliminary interface due to how easily configurable presentation. The code was initially developed using Spyder version 3.2.8 then moved to Visual Studio Code which enables easy version control via Git. The machine that was used to test and implement the code is a Macbook Pro (13-inch, 2017) running on macOS Mojave, 2.3 GHz Intel Core i5 with 8GB memory.

The methodology used in this project is the Agile model. While the Waterfall model was initially adopted during the project planning, it was quickly seen as an inappropriate model for the project. The waterfall model assumed that once each method was implemented, there will be no further need to revisit them until the deployment phase. As the project grows, many problems and testing opportunities were found, thus resulting in small incrementation of individual features instead of going through one big phase to the project's completion. Each sprint of the code implementation last 1 – 2 weeks, in which the progress will be reported to the supervisor and proceed accordingly to the results.

Since the code was purely developed in Python, the architectural design diagram will represent the packaging structure of the python code. There are four .py files in the package, each with different core functions in which were structured as follow:

acse_9_irp_wafflescore		package of all the functions
	__init__.py	
	MiscHelpers.py	Functions that were used across multiple files
	SOMsHelpers.py	Functions that mainly deals with SOMs
	FCMHelpers.py	Functions related to Fuzzy C mean clustering method
	dataPreprocessing.py	Functions that deals with data preprocessing
	minisom.py	Contain class Minisom, Self-organizing maps method
	fuzzy_clustering.py	Contain class FCM, a fuzzy clustering method
example		Package of an example script
	__init__.py	
	full_script.py	Example script to run each clustering method
	Test run.ipynb	Notebook interface of to run *full_script.py*
Notebooks		All the notebook used to experiment and record results
	Model plotting - Data Exploration	
	Data Pre-processing	
	K-Means	
	Fuzzy C means	
	HDBScan	
	SOMs	
	SOMs - M5a	
	SOMs - M5b v1.0	
	SOMs - M5b v2.0 - with XZ	
	SOMs - M5b v2.0 - no XZ	
	Physical Parameter Test	

Additional details of each function are documented in the code as docstring. Similarly, the results clustering of each method is saved in its respective Jupyter Notebooks.

With the main focus of exploring the capability of each classification method, most of the helper python files written contain scripts that evaluate, validate, and perform parameter tuning on each of the classification methods. The two object classes that were utilizes are the minisom and fcm class.

In addition to the python files, Jupyter Notebooks are used to call the functions and scripts from the project package. These notebooks serve as both the front end and a record of all the findings.

The project mainly utilizes numpy as the main data container. Tensorflow was looked upon initially but since the program was developed on one CPU, using Tensorflow will result to a slower runtime. For future development that includes GPU, Tensorflow could be implemented as the data container instead of numpy.

The section below goes through the main functions presented in the project. To simplify, the flow of the program went in the following order.

1. Data preprocessing
2. Classification Method
    a. K-Means
    b. Fuzzy C means (FCM)
    c. HDBSCAN
    d. Self-Organizing Maps
        i. SOMs – K-means
        ii. SOMs – FCM
        iii. SOMs – HDBSCAN
3. Verification and Validation methods

Note that each classification method does not depend on each other except for SOMs sub-methods.

Data Preprocessing

The synthetic model created by Dr Michele Paulatto, who is the supervisor of this project, is the main input of this project. There are three Earth models available and each was tested at different stages of development. The models are as follow:

M1 – Initial model with 10 classes of lithology.
M5a – Simplified model with 5 classes of lithology.
M5b – Similar to M5a but with less anomaly in the temperature field.

These models were saved as .npz files and are generated from the scripts located in *Synthetic model*. The properties presented in the model are compressional wave velocity (V_s), shear wave velocity (V_s), density, V_p/V_s ratio, compressional wave attenuation (Q_p^{-1}), and shear wave attenuation (Q_s^{-1}). In each of the Earth model, some datapoints contains INF and NaN values. The INF is due to the property of water that does not have V_s, thus causing the value of V_p/V_s to reach INF. The value of $Q_p^{-1}$and Q_s^{-1} in the water layer greatly differs from rocks', making them not ideal for computation and thus replaced with NaN. In this scenario, we will assume that the position of water is known.

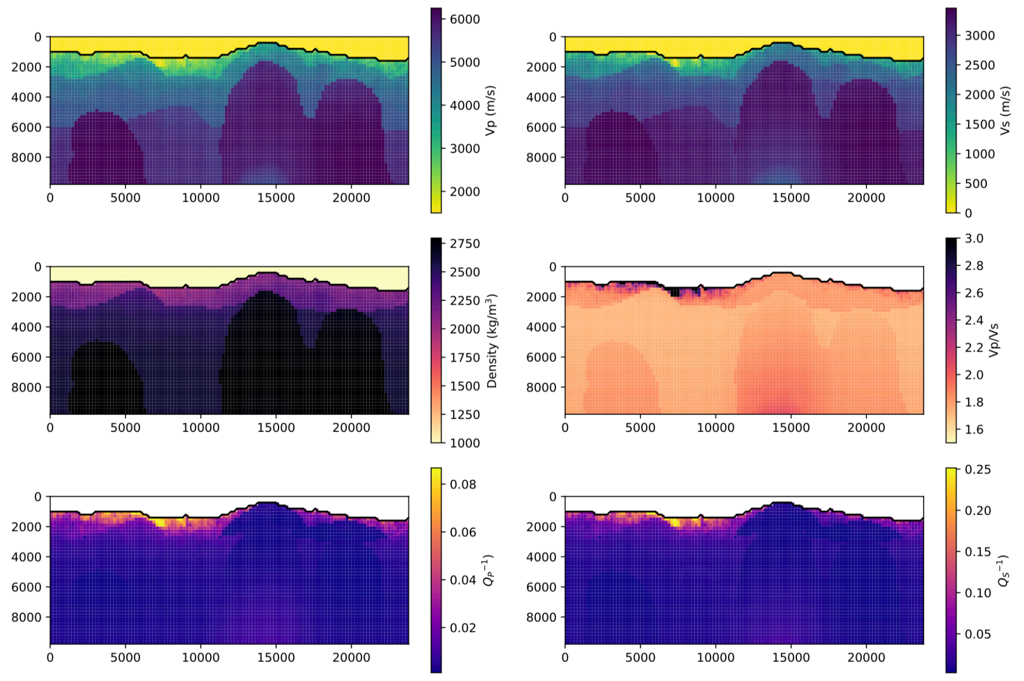

Figure 6 Property fields of Synthetic model M5b, Vp, Vs, Density, Vp/Vs Ratio,

The pre-processing scripts presented in notebook *Data Pre-processing* deals with those problems. It initially loads the .npz Earth model file to convert them into standard 2d numpy array in which each column represents each data properties. By inputting the water position, the program then removes all the water indexes in the earth model file.

The script then runs the function *data_cleanup*, which will transform the data input according to the input column property. The transformation applied to each property are as follow:

V_p, V_s, density	Divided by 1000
V_p/V_s ratio	Capped to 10
Q_p^{-1}, Q_s^{-1}	Applied log

After the transformation was done, the data can then be normalized and used in later stages.

Code Metadata

The latest version of the project is located on Github master repository, along with the user guide and license. The project should be able to run successfully on any platform that supports Python 3.6.5, though it is recommended to run on macOS platform as it was initially tested on. The libraries and its minimum version requirement that would be required to install are as follow:

scipy	1.1.0
scikit-image	0.13.1
numpy	1.14.3
scikit-learn	0.19.1
hdbscan	0.8.22
pandas	0.23.0
seaborn	0.8.1

all of which can be installed via pip for macOS and Linux platform though will require additional steps for Windows. It is recommended to use Anaconda Distribution to run the python files as it can manage libraries and dependency on Linus, MacOs, and Windows. The listed version may not be the latest release of each libraries and may result in some error when compiling with a different version.

Additional libraries that are modified and used are Minisom by JustGlowing and fuzzy-clustering by oeg-upm. Since both of the source codes' license allows modification, provided that the changes are listed in the source code, the changes and the license name were added as a comment to each of the source codes' python file. The files are minisom.py and fuzzy_clustering.py.

The package can be installed using *pip install acse-9-irp-wafflescore* command on the terminal. An additional instruction on how to install a package can be found here. The current version of the program is 1.0.5. Additional documents can be found on the Github master repository.

Implementation & code

The project was built so that the user can include any number of physical properties as an input array, though it was limited that the input should be in the form of numpy 2D array.

Additional input routines for specific input and output file formats can be easily added. Each row of the 2D numpy array should represent one datapoint, and the columns represent each of the physical properties. The total number of data-points is limited to the capability of the machine that runs the program. The system of the machine and the amount of RAM available is the limiting factors. Usually, for 32bit system, the maximum memory that can be allocated for the Python program will be around 2GB to 4GB. This information was stated on Microsoft Memory Limits for Windows [20].

Unit test and system test are located in the test folder. The unit tests are done for functions that were newly written, excluding the plot functions. As for the system test, it performs the whole process of getting the final cluster result of each method.

Tests of each clustering method were run using the provided synthetic Earth model. The result of each method is recorded in the following notebooks:

1. *Model plotting - Data Exploration*: Initial phase of the project, plots and explore the input synthetic data model M1.
2. *Data Pre-processing*: runs and pre-process the data
3. K-Means: Parameter tuning and result
4. Fuzzy C means: Parameter tuning and result
5. HDBScan: Parameter tuning and result
6. *SOMs*: Parameter tuning of the SOMs
7. *SOMs - M5a*: test SOMs on model M5a
8. *SOMs - M5b v1.0:* test SOMs with other classification methods
9. *SOMs - M5b v2.0 - with XZ* and *SOMs - M5b v2.0 - no XZ*: test SOMs with other classification method using a different set of SOMs hyperparameter.
10. *Physical Parameter Test*: test on different sets of available physical properties

All of which are located in the Notebook folder inside the GitHub repository. Notable observations from the results are mentioned in the next section, further detail can be found on each method's notebook.

The main Earth model that was used to test is model M5b, which contain a total of 5 classes and does not contain large anomalies in the temperature field. Note that the temperature is not part of the tested properties but serves as an input to generate the synthetic model and represents a source of noises. The classes presented are water, Andesite clastic sediments, Andesite lavas, Metasediments, Granitic crust, and Diorite Intrusions

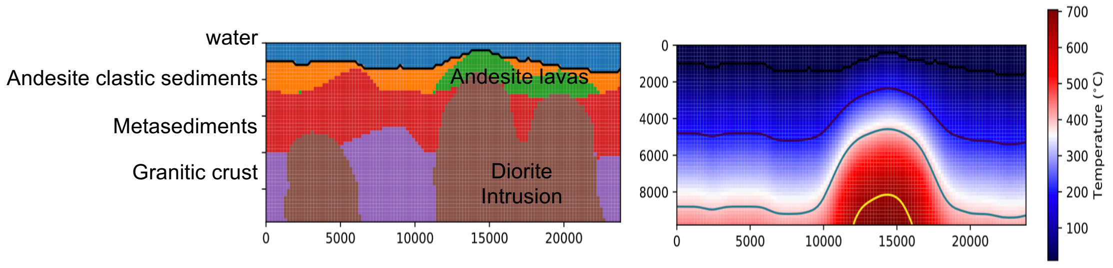

Figure 7 Synthetic model M5b lithology class (left) and temperature mapping (right).

Initially, an assumption was made that using the topographic position (x and z) should be able to contribute to a better result as rocks of the same type are naturally together. Instead, it causes the clustering method to give too much attention to the position and results in too many separate clusters. Removing the topographic location lessens the number of clusters, correctly identifies the section with high temperature and shows quite an accurate recovery of the layer of sediments but causes the model to lose some precision of the intrusion parts. This behaviour is seen in both SOMs and HDBSCAN.

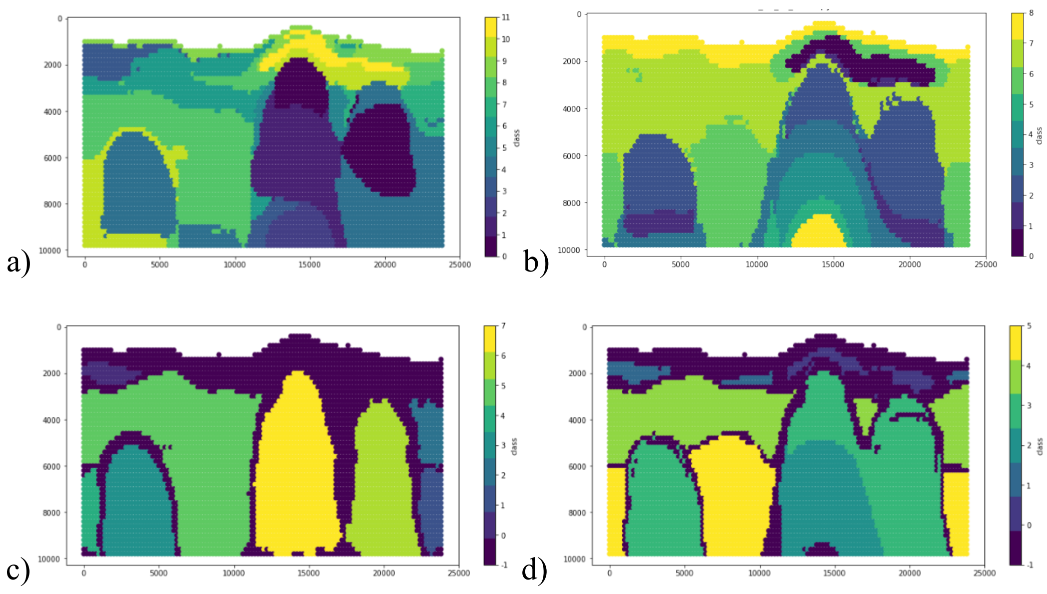

Figure 8 Model M5b: SOMs results from using all parameters with (a) and without topographic location(b). HDBSCAN results using all parameters with (c) and without topographic location (d).

A notable difference between these two methods is the ability to identify noises. HDBSCAN classifies all noise into its cluster, labelled with -1 and depicted in dark blue colour as seen in Figure 8-c,d. This is greatly beneficial in tomography as it can help identify the region that transit from one rock type to another.

Comparing to model M5a which contains more anomaly and noise, all clustering method seem to suffer greatly from trying to identify a unique class to each of the anomaly section. The only method that seems to be robust in this case is the HDBSCAN which correctly identify the noise from high temperature.

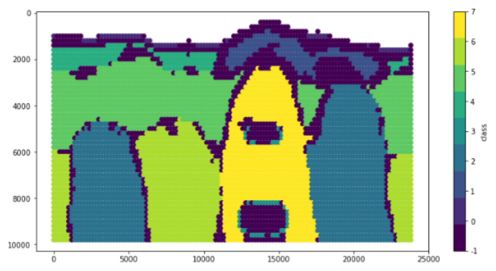

Figure 9 HDBSCAN result when tested with model M5a using all parameters.

As for K-means when tested with all parameters, the number of clusters that result to the best silhouette score and CH score ranges between 12 to 18 clusters, and the SSE plot does not show any elbow point observable elbow point. This means that the use of internal indexes does not help us to find the best number of clusters at all. The result with a lower number of clusters also does not show any significant changes.

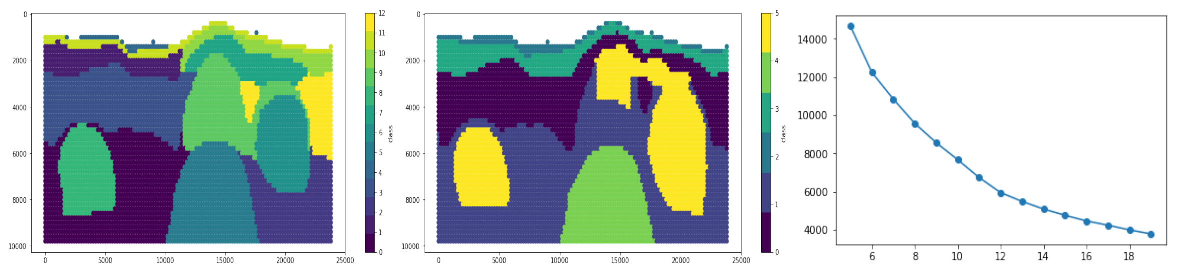
Figure 10 Model M5b: K-Means results with the best silhouette and CH score (left), with 6 clusters (middle), and SSE plot (right).

Similar behaviour is found in FCM, though it is notable that once we plot the degree of membership of each class, some characteristic emerges. The once unobservable Andesite lavas show up as an anomaly in all of the cluster's degree of membership.

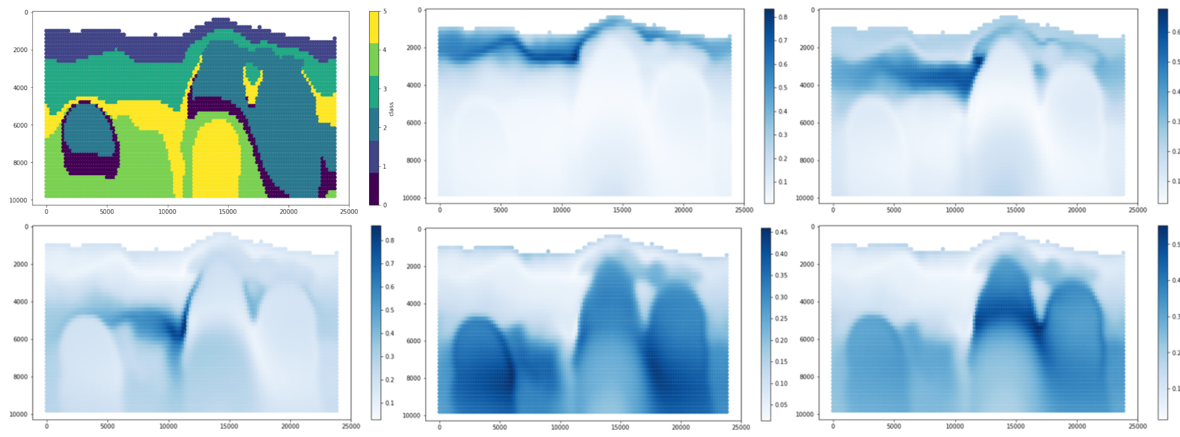
Figure 11 Model M5b, FCM result with 5 clusters and its degree of membership of each cluster.

While SOMs work quite well on its own, application of it with other clustering methods are not decent. For SOMs-FCM, the result does not introduce any new information apart from what was already observed SOMs. In SOMs-HDBSCAN as seen in Figure 12, there are fewer datapoints classify as noise but are more scattered. This cause the border between each facies to be less precise. SOMs is also considerably slower than HDBSCAN, with these two drawbacks, it is better to use purely HDBSCAN over SOMs-HDBSCAN.

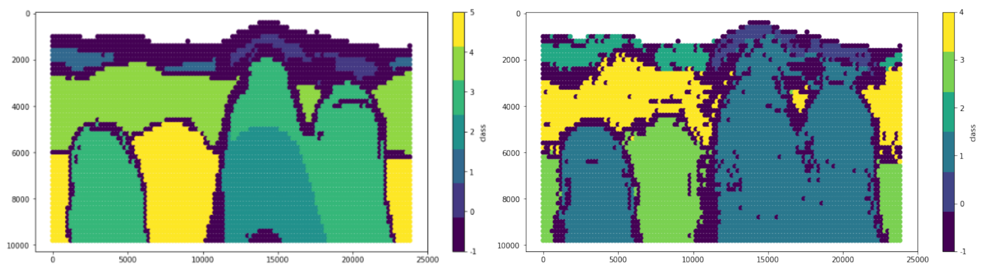
Figure 12 Model M5b without topographic location, HDBSCAN result (left) and SOMs-HDBSCAN result.

Moving on to alteration in physical parameter, while all the methods are tested in notebook *Physical Parameter Test*, each of the method's hyperparameter were not altered and may not represent the best possible outcome of pair of method and physical parameter. Due to the slow computation speed of FCM and SOMs shown in **Error! Reference source not found.**, performing hyperparameter search for each of the parameter pairs will be very costly, thus only the K-means and HDBSCAN are explored.

In K-means, using V_p, Q_p^{-1}, and density seems to work best, though may represent a few extra clusters, the result is considerably better than using all the physical parameters. FCM also returns a similar result where the result is better with less physical parameters.

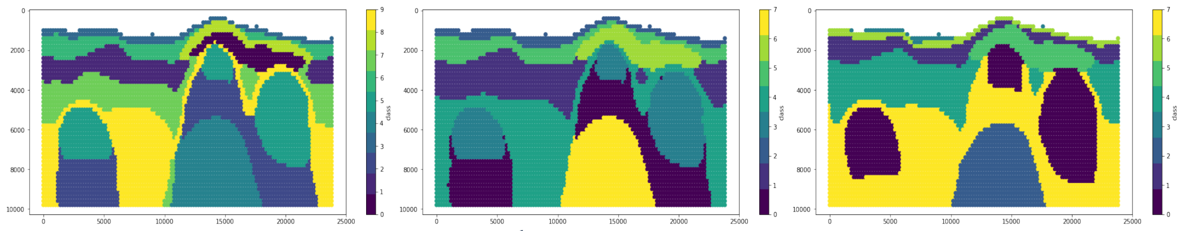

Figure 13 Model M5b: K-means paired with Vp, Qp^{-1}, and density with 10 (left) and 8 clusters (middle). K-means paired with all physical parameters (right).

Since HDBSCAN performs quite well with noise, the test was done with Model M5a. The result is not as good as the result of using all physical parameters.

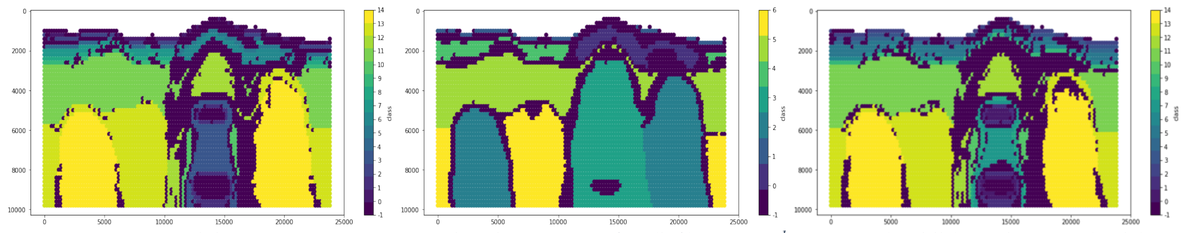

Figure 14 Model M5a, HDBSCAN paired with Vp, Vs, Vp/Vs (left), Vp, Qp^{-1}, Density (middle), and Vp, Vs (right).

The average computation time in seconds of each test are presented below:

	FCM	HDBSCAN	SOM	SOM-FCM	SOM-HDBSCAN	KMEANS
Full Test	89.980826	0.1418735	40.6964635	106.252	40.7966045	0.2290125
Full Test no X, Z	80.1082085	0.1320455	40.122523	106.3039935	40.220171	0.1543205
Vp, Vs, Vp / Vs	67.437797	0.0981345	40.174758	105.784623	40.277581	0.205754
Vp, Qp, density	68.204626	0.1044515	46.889637	117.174485	46.9936315	0.165201
Vp, Vs	61.966932	0.0996405	32.625169	99.7475845	32.721147	0.168955

Table 1 Average computational speed of each methods performing each set of physical parameters.

SOMs and FCM take up a huge computational time, though, in the case of SOMs-FCM, the application of SOMs reduces the number of dimensions and makes the FCM runs faster. Still, the cost of computing SOMs highly outweighs the speed up in FCM part.

Discussion and Conclusion

One of the issues in selecting a clustering method lies in the verification process. While several verification methods are explored in this project, it still is hard to fully articulate the result and quantitatively rank the performance of the different clustering methods. During hyperparameter search, often the model that has excellent external verification result suffer from having a very high number of clusters. The issue was countered by adding some intuition to the hyperparameter search by limiting the minimum and the maximum number of clusters allowed.

Parameter tuning also plays a big role in the unsupervised learning method. For SOMs, there are multiple parameters to fine tune, the ones that were looked upon in this project are:

1. Kohonen layer's dimension
2. Learning rate
3. Number of iterations
4. The spread of the neighbourhood function (sigma)

Grid search and random search were done to find the best parameter, the downside is that it must be done for each of the input parameter set. This process is very costly but needs to be done otherwise the watershed segmentation will not return a decent result.

In conclusion, HDBSCAN seems to work best when there are noise present in the input data and when there are multiple physical parameters to learn from. SOMs also works on those conditions but have a lower purity score due to its inability to recognize noises. In the case of FCM and K-means, the result became significantly better when there are not many physical parameters to deal with and works best with V_p, Q_p^{-1}, and density in the current study. Applying other clustering methods with SOMs is possible but not ideal, while it improves the result of FCM, the computational speed doubles up while the validation score changes very little.

Possible future development may include modifying the watershed segmentation in SOMs method. The current work fills up the masking area with K-nearest neighbour and random walk, thus neglecting the possibility that those portions in the neuron map may belong to noise cluster.

Bibliography

[1] A.A. Konaté, H. Pan, S. Fang, S. Asim, Y.Y. Ziggah, C. Deng, N. Khan, Journal of Applied Geophysics 118 (2015) 37–46.
[2] K. Bauer, G. Muñoz, I. Moeck, Geophysical Journal International 189 (2012) 984–998.
[3] G. Muñoz, K. Bauer, I. Moeck, A. Schulze, O. Ritter, Geothermics 39 (2010) 35–45.
[4] B. Braeuer, K. Bauer, Geophysical Research Letters 42 (2015) 9772–9780.
[5] P.A. Bedrosian, N. Maercklin, U. Weckmann, Y. Bartov, T. Ryberg, O. Ritter, Geophys J Int 170 (2007) 737–748.
[6] A. García-Yeguas, J. Ledo, P. Piña-Varas, J. Prudencio, P. Queralt, A. Marcuello, J.M. Ibañez, B. Benjumea, A. Sánchez-Alzola, N. Pérez, Computers & Geosciences 109 (2017) 95–105.
[7] G.E. Hinton, T.J. Sejnowski, H.H.M.I.C.N.L.T.J. Sejnowski, T.A. Poggio, Unsupervised Learning: Foundations of Neural Computation, MIT Press, 1999.
[8] I. Sutskever, R. Jozefowicz, K. Gregor, D. Rezende, T. Lillicrap, O. Vinyals, ArXiv:1511.06440 [Cs] (2015).
[9] J. Macqueen, in: In 5-Th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.
[10] D.J. Ketchen, C.L. Shook, Strategic Management Journal 17 (1996) 441–458.
[11] Z. Cebeci, F. Yildiz, in: 2015.
[12] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Kluwer Academic Publishers, Norwell, MA, USA, 1981.
[13] L. McInnes, J. Healy, 2017 IEEE International Conference on Data Mining Workshops (ICDMW) (2017) 33–42.
[14] T. Kohonen, Proceedings of the IEEE 78 (1990) 1464–1480.
[15] I. Rojas, G. Joya, A. Catala, Advances in Computational Intelligence: 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Springer, 2015.
[16] L. Vincent, P. Soille, IEEE Trans. Pattern Anal. Mach. Intell. 13 (1991) 583–598.

[17] J.A.F. Costa, M.L. de A. Netto, in: IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), 1999, pp. 367–372 vol.5.
[18] P.J. Rousseeuw, Journal of Computational and Applied Mathematics 20 (1987) 53–65.
[19] E. Rendón, I.M. Abundez, C. Gutierrez, S.D. Zagal, A. Arizmendi, E.M. Quiroz, H.E. Arzate, in: Proceedings of the 2011 American Conference on Applied Mathematics and the 5th WSEAS International Conference on Computer Engineering and Applications, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2011, pp. 158–163.
[20] mcleanbyron, (n.d.).