Technical University of Crete

Master of Science in Technology & Innovation Management

Master Thesis

"Assessment Metrics for Clustering Algorithms"

20/5/2023

Angelopoulos Vasileios

Contents

Brief Curriculum Vitae

My name is Vasileios Angelopoulos and I am a student of the Master in Technology and Innovation Management offered by the Technical University of Crete. I have obtained my Diploma in Mechanical Engineering from the Aristotle University of Thessaloniki with specialization in Energy, Aeronautics and Engines. The subject of my Diploma Thesis was diesel engine simulation and exhaust aftertreatment technologies evaluation and I concluded my studies with a 6-month internship as a junior engineer at the Toyota Motor Europe Technical Center in Brussels. My interests include internal combustion engines, eco-friendly vehicles, renewable energy sources and computer programming. I am currently working as a software engineer in the field of embedded software for automotive applications.

Summary

In the first part of the Thesis, the theoretical background will be provided for Artificial Intelligence, Machine Learning, Supervised and Unsupervised Learning and Clustering in order to provide the basis for understanding the next chapters. In the second chapter, the theoretical and mathematical background is provided for each clustering algorithm that will be implemented. In the third chapter of the Thesis the different clustering assessment metrics are discussed and their mathematical background is presented. In the fourth chapter the basic steps of the implementation that will be carried out in order to provide the necessary outputs will be described. The intended output/result of the Thesis is the development of a software script in Python capable to take different datasets as an input, implement different clustering methods and export the clustering performance metrics. The correct operation of the developed script is shown in the final chapter where an example dataset is used to showcase the capabilities of the script by presenting the results/outputs of the script along with commentary. A manual covering the basic elements of the script can be found in the same chapter.

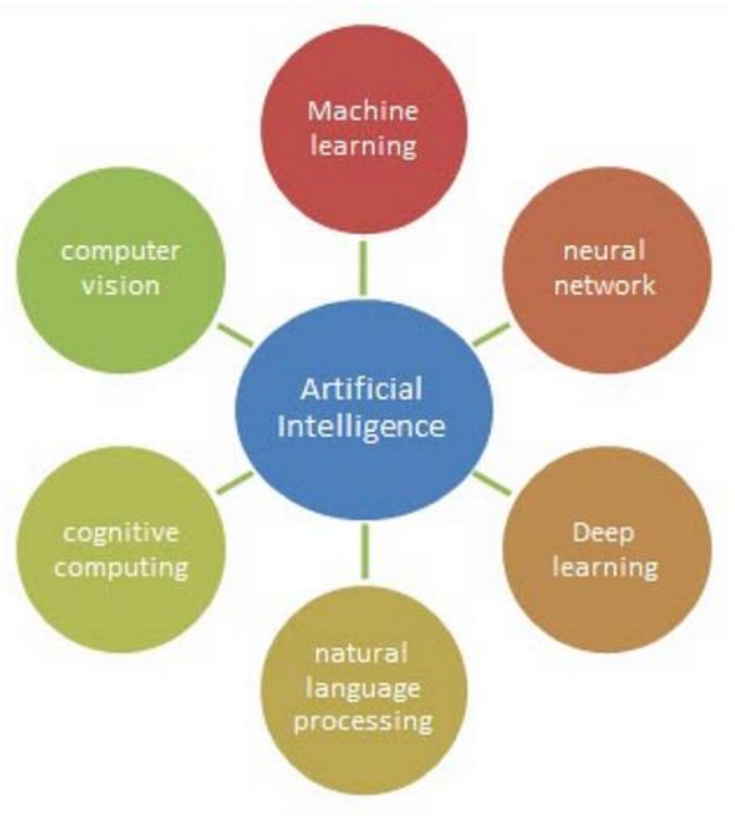
Chapter 1 - Theoretical Background

Artificial Intelligence (AI)

Artificial Intelligence is a developing technology which tries to simulate human intellectual and how the human mind functions. In the year 1950, John McCarthy was the first to conceive the term Artificial Intelligence. He stated that every single part of learning or any other element of intelligence can be precisely described in a way that a machine can simulate it. He also stated that an effort will be made to explore ways to make machines improve themselves and solve the kind of problems that are typically reserved only for humans.

(Source: https://towardsdatascience.com/advantages-and-disadvantages-of-artificial-intelligence-182a5ef6588c)

Artificial Intelligence is the ability of a computer program to perform tasks that would normally require human intelligence. Such tasks are language translation, speech recognition, decision-making, and visual awareness and perception. AI combines a very extensive range of technologies that include machine learning, natural language processing, computer vision, cognitive computing, deep learning and neural networks.



Branches of AI Based on Capabilities

Source: https://rancholabs.medium.com/6-major-sub-fields-of-artificial-intelligence-77f6a5b28109

The six primary subfields of Artificial Intelligence are:

- Machine Learning is a subfield of artificial intelligence that enables computers to learn from various data, be able to recognize patterns and also improve their performance by themselves without human interference and input. Machine learning is able to classify, interpret and estimate data from even a complex data set.

- Natural language processing (NLP) is a subfield of artificial intelligence that allows computers and humans to effectively communicate using natural language. This is accomplished through various methods such as text translation, speech recognition and sentiment analysis.

- Neural Network is a subfield of artificial intelligence that integrates cognitive science into machines in order to perform tasks. In more simple words, a Neural Network imitates how the human brain works by identifying existing connections among large amounts of data. Neural Networks are currently used for risk analysis, fraud detection, sales forecast etc.

- Computer vision is a subfield of artificial intelligence that enables the computer to interpret visual input from real images. Computer vision is extensively applied in the healthcare sector in order to evaluate a patient's status and existing condition by interpreting MRI scans, X-rays and other available imaging methods.

- Deep Learning is a method of learning in which the machine analyses the input data until it identifies a sole satisfactory output. This method is also known as self-learning of machines/computers. The main concept of Deep Learning is that a machine/computer is be able to find an acceptable solution by performing a loop of repetitive takes and self-analyzing.

- Cognitive Computing is a subfield of artificial intelligence that is capable of simulating human intelligence in machines/computers. In order to do that it uses self-learning algorithms that are in turn making use of a range of technologies like data mining, pattern recognition, and natural language processing. The aim is to manage to effectively simulate the human thought process in a computer model. A currently popular example of cognitive computing is Google Assistant.

Sources:

https://www.softwaretestinghelp.com/what-is-artificial-intelligence/

https://www.dataversity.net/what-is-cognitive-computing/

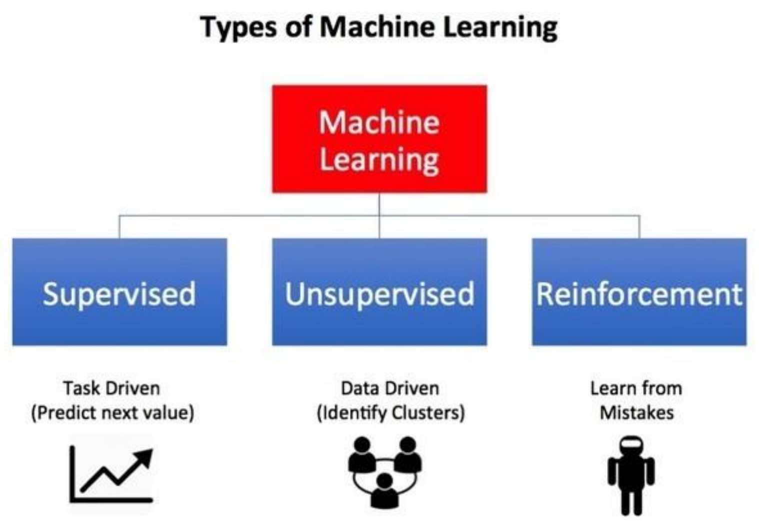
Machine Learning (ML)

Machine learning is a subfield of artificial intelligence, which is defined as the ability of a machine to emulate intelligent human thinking and solve problems like humans do. Machine learning was defined in the 1950s by Arthur Samuel as the area of study that gives the ability to computers to learn without the need of a human input to program them. The main approach of Machine learning is to let computers program themselves through previous human inputs and experience.

There are three main types of machine learning:

- supervised learning
- unsupervised learning
- reinforcement learning

Supervised machine learning models are trained with data that has labels, which allows the models to learn and become more precise in their future predictions. For example, an algorithm would be trained with images of fruits and other random things, that are already labeled by a human. The machine would then learn different ways to identify labels of new and unseen pictures of fruits on its own without human interference and input. Unsupervised machine learning models on the other hand are able to work with data that has no available labels. Unsupervised machine learning algorithms are capable to discover patterns that people aren't intentionally looking for and/or are unable to pinpoint. As an example, an unsupervised machine learning program is able to explore and analyze sales data and then classify different categories of the existing customers that are making purchases. Lastly, Reinforcement machine learning models are capable of learning through their mistakes by trial and error and then get feedback in the form of a reward or a penalty.


Types of Machine Learning
Machine Learning
Supervised
Unsupervised
Reinforcement
Task Driven (Predict next value)
Data Driven (Identify Clusters)
Learn from Mistakes

Main types of machine learning

Source: https://techlech.com/what-are-the-three-types-of-machine-learning-supervised-learning-unsupervised-learning-and-reinforcement-learning/

More details about supervised and unsupervised learning are presented below. This project will focus on clustering which is an unsupervised machine learning method.

Sources:

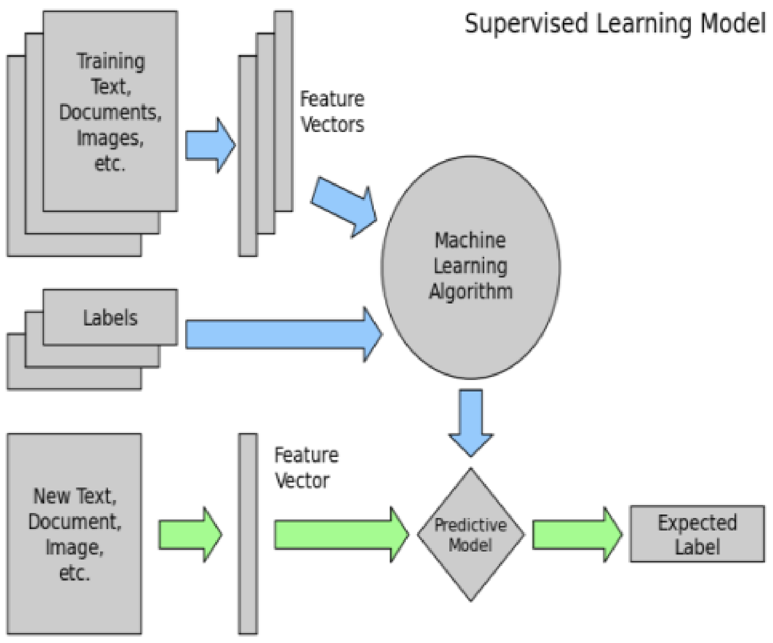
https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

https://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects.

Supervised learning

Supervised learning refers to a specific category of machine learning algorithms in which a model is trained on a dataset where each data point has an available matching label. The aim of supervised learning is to learn a function that connects the input data points to the correct output labels therefore allowing the function to forecast and predict the labels of unseen and unknown data. Supervised learning plays an important role in numerous tasks like email spam filtering, image classification, natural language processing and also speech recognition.


Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Supervised Learning Model

Source:
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=deb44af9d65763af055ec66b29e9b12f9b80f564

The input data in supervised learning is a set of features/ attributes which are the characteristics of the data using numerical or categorical variables or mixed. The output labels can be categorical or continuous or have binary labels like true/false and multi-class labels like "A", "B", "C". Supervised learning algorithms can be split into two distinct categories/types: classification and regression. Classification algorithms forecast and predict categorical labels based on the input data. Regression algorithms are able to predict continuous values.

Supervised learning can be accurately presented mathematically as follows:

With a training dataset D,

$$D = \{(x1, y1), (x2, y2), ..., (xn, yn)\},$$

Where:

xi is an input feature vector,

yi is its matching output/label

The goal of the algorithm is to construct a function f(x) that connects the inputs x to the outputs y. This function can be constructed by minimizing a loss function L(f(x), y), which measures the difference between the predicted output f(x) and the true/real output y. This loss function can be defined depending on the specific problem that we have to tackle, such as mean squared error for the regression problems and cross-entropy loss for the classification problems. Various optimization techniques can be used to minimize the loss. When using the gradients of the loss function the model parameters are constatly changing in each iteration/loop until the loss function is minimized.

Some very well-known supervised learning algorithms include support vector machines (SVMs), decision trees and neural networks. These algorithms have a different model complexity and make use of different optimization methods to fit the model. In order to train a supervised learning algorithm, we use a dataset with available matching labels to fit a model to the data. The performance of the model is assessed on a distinct test dataset and the model is then used to forecast and predict the labels of unseen data.

It must be highlighted that acquiring labeled data to use for supervised learning can be difficult and also expensive. Also, the quality and precision of the labels can have an impact on the performance of the algorithm and special attention must be given to the selection process of the training data.

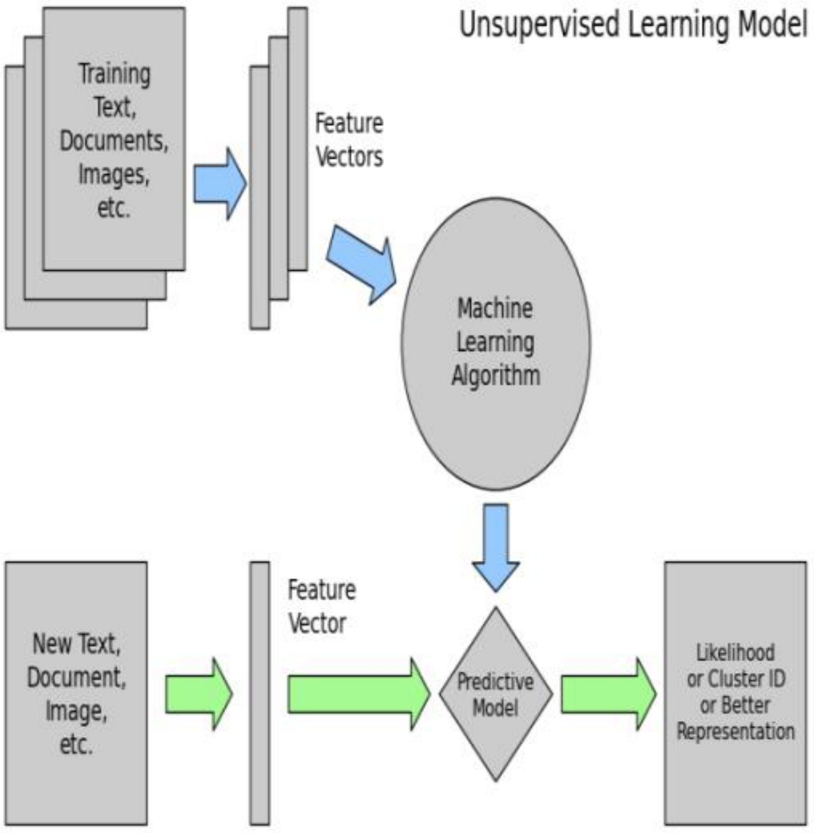
Sources:

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. Springer Science & Business Media.

Unsupervised learning

Unsupervised learning refers to a specific category of machine learning algorithms in which an algorithm is trained using a dataset without supervision. In unsupervised learning the algorithm is provided with an dataset without matching labels and it must identify relationships present within the data, unlike supervised learning where the algorithm is provided with examples with matching labels (training data set) and is prepared in order to forecast and predict the labels of new data.


Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

Unsupervised Learning Model

Source:

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=deb44af9d65763af055ec66b29e9b12f9b80f564

Unsupervised learning can be accurately presented mathematically as follows:

With a training dataset D,

$$D = \{x1, x2, ..., xn\},$$

Where:

xi is an input feature vector

The aim of the algorithm is to to construct a function f(x) that matches inputs x to outputs y without any previous knowledge or experience of y. Clustering is a popular and common use of unsupervised learning, where the algorithm clusters data points that are similar into distinct groups. Clustering can be implemented for a variety of different tasks and applications like customer and image segmentation.

Dimensionality reduction is another well-known application of unsupervised learning where the algorithm effectively reduces the number of features/attributes in the dataset while keeping as much information as possible. This is very useful for visualization purposes, feature selection tasks, and cases where speeding up computation speed and/or reducing computational load is needed. Unsupervised learning is also commonly used as a pre-processing phase before working with supervised learning methods. The unsupervised machine learning algorithm is used to discover different hidden patterns in the data before finally training the supervised model.

Some of the most well-known unsupervised learning algorithms include k-means clustering, hierarchical clustering, Gaussian mixture models (GMM) and principal component analysis (PCA). These algorithms can be used to solve problems in a broad range of sectors like finance, healthcare and marketing by discovering hidden structures/connections and recognizing patterns in the various datasets.

Sources:

https://www.javatpoint.com/difference-between-supervised-and-unsupervised-learning

https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

Ng, A. (2012). Unsupervised feature learning and deep learning. In Proceedings of the 2012 IEEE conference on computer vision and pattern recognition

Clustering

Clustering is an unsupervised machine learning method that has the capability to split a dataset into distinct groups (known as clusters) based on the patterns that can be recognized in the different datasets. Clustering has a very wide range of real-world applications in different fields like customer segmentation, image processing and segmentation and anomaly detection.

The main families/groups of clustering algorithms are as follows:

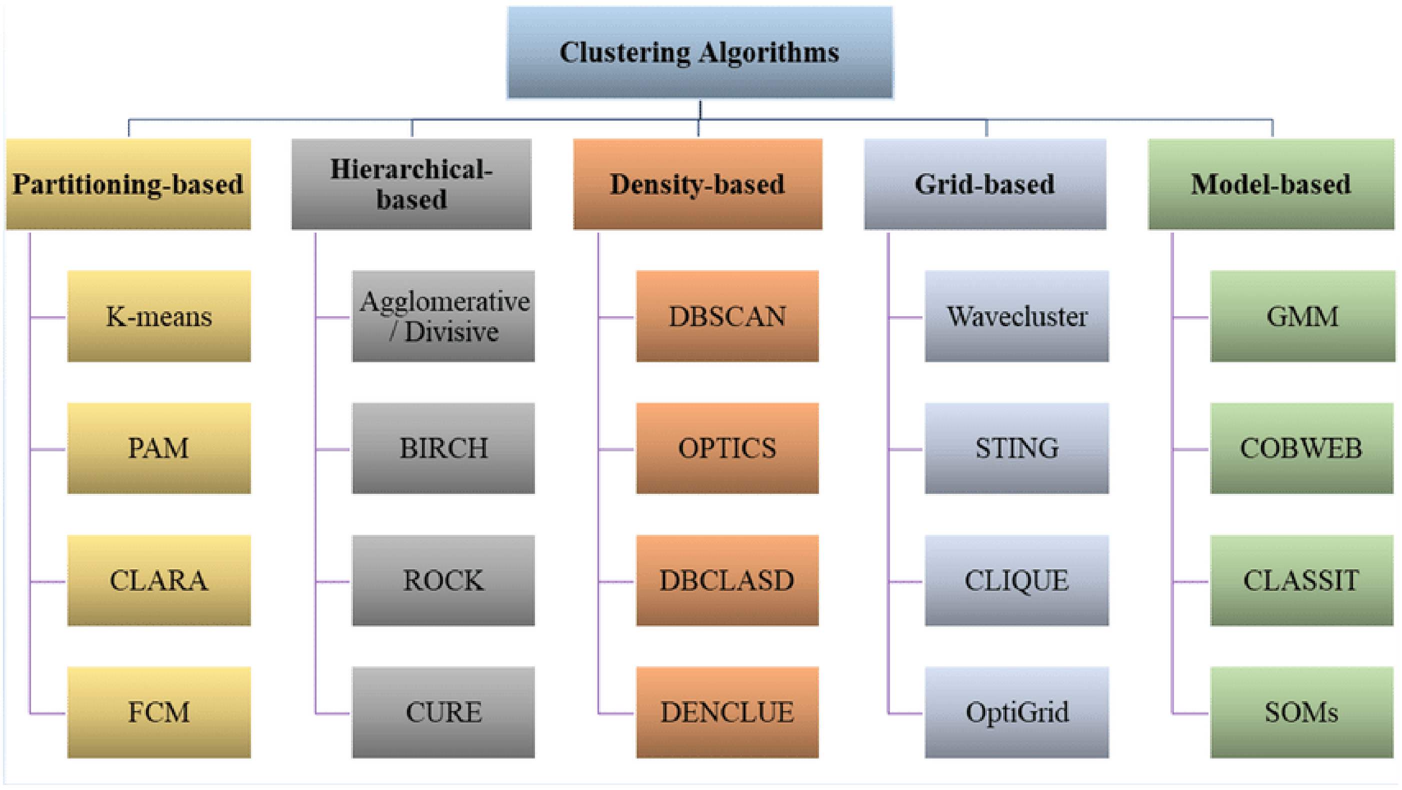
- **Partitioning-based clustering:** This type of clustering algorithm/method splits the dataset into a pre-determined number of partitions/clusters. The goal of partitioning-based clustering is to group similar data points together while separating the non-similar data points into different clusters. In partitioning-based clustering, the number of clusters is defined before the clustering begins and the algorithm splits the dataset into the chosen number of pre-determined clusters. The partitions/clusters are constructed by optimizing an objective function that aims to minimize the distance between data points that are in the same cluster while also maximizing the distance between data points that are in different clusters. The most well-known examples of partitioning-based clustering algorithms are K-means and K-medoids.

- **Hierarchical clustering:** This clustering method creates a hierarchy of clusters by repeating the splitting of the data into smaller groups. Two well-known types of hierarchical clustering methods are agglomerative and divisive clustering are.

- **Density-based clustering:** This type of clustering method groups together points that are close to each other and are of high density (compact), while seeing outliers as noise. A popular density-based clustering algorithm is DBSCAN.

- **Grid-based clustering:** This type of clustering method involves splitting the data into a grid of cells, where each cell is representing a possible cluster. Grid-based clustering algorithms merge adjacent cells that meet specific similarity criteria to form clusters. Examples of grid-based clustering algorithms include WaveCluster, STING and CLIQUE.

- **Model-based clustering:** This clustering algorithm assumes that the data is formed from a mixture of probability distributions and tries to approximate the parameters of these distributions. An example of a model-based clustering algorithm is Gaussian mixture models (GMM)



Broad classification of clustering algorithms

Source: https://www.researchgate.net/figure/Broad-classification-of-clustering-algorithms_fig2_341111980

The above graph shows the main categories of clustering methods. The choice of algorithm depends on the characteristics of the data and the problem being solved as each of these families of clustering algorithms has its own strengths and weaknesses when compared to each other. Note that this graph is not exhaustive as it does not contain all algorithms that exist and there are many other clustering methods that do not clearly fit into one of these categories.

Sources:

Ng, A. (2014). CS229 Lecture notes: Unsupervised learning. Stanford University.

https://scikit-learn.org/stable/modules/clustering.html

Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques

Jain, A. K., & Dubes, R. C. (1988). Algorithms for clustering data. Prentice-Hall.

Chapter 2 - Clustering algorithms

The clustering methods that will be implemented in later stages of the thesis are presented thoroughly in this chapter.

K-means clustering

K-means clustering is a clustering method capable to partition a set of data points into K clusters, where K is the selected number of clusters that is predefined. The algorithm works by iteratively assigning each data point to the closest available cluster centroid and then updating the centroids to be the mean of the points that are currently assigned to them. The algorithm proceeds to do the same for another iteration until either the centroids stop moving/changing or the assignments of data points to the different clusters stop changing. The algorithm intents to reduce the sum of squared distances between the various data points and their respective cluster centroids.

The discrete steps of the K-means algorithm are as follows:

1. K cluster centroids are initialized. This can be done by randomly selecting K data points to act as the initial centroids.
2. Each data point is assigned to the closest cluster centroid by calculating the distance (using a distance metric like the Euclidean distance) between each data point and each centroid.
3. The cluster centroids are then updated in order to to be the mean of the points that are assigned to them.

1. Steps 2 and 3 are repeated until the centroids are staying the same or the assignments of points to clusters stay the same between two runs of the algorithm.

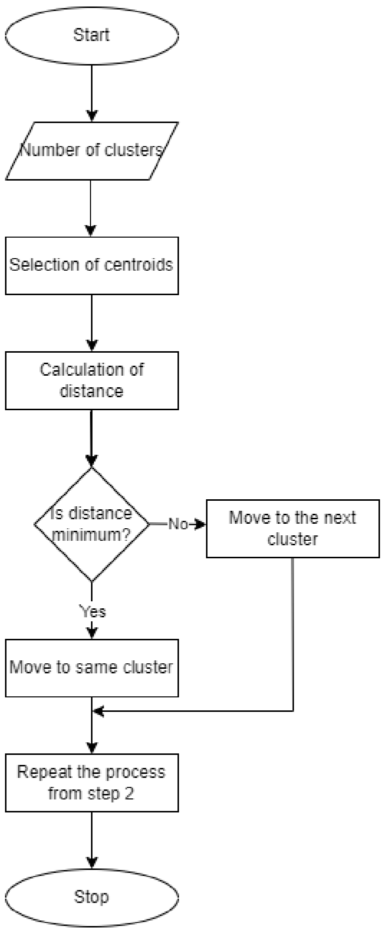
The K-means algorithm has one main advantage which is that it is fast and very easy to implement, especially when working with large datasets. On the other hand, it can be sensitive to the initialization of the centroids and it also may not always lead to the clustering solution that is the most optimal. Also, a non-optimal choice of K can lead to suboptimal clustering which is why it is very important to carefully choose the number of clusters that is given as an input to the algorithm. Finally, a disadvantage of the K-means algorithm is that it assumes that the clusters have a spherical shape which may not always be true. The algorithm can also be affected by outliers and noise

Sources:

https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a).

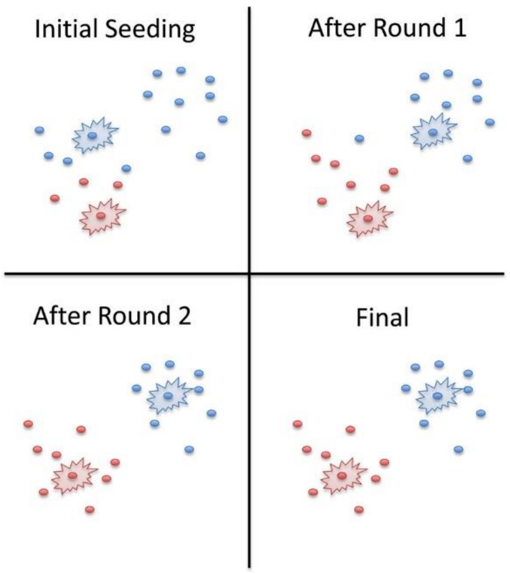
https://devopedia.org/k-means-clustering#Singh-2018

https://scikit-learn.org/stable/modules/clustering.html#k-means



Flowchart of k-means algorithm.

Source: Singh, Seema. 2018. "K-Means Clustering." Data Driven Investor



Example of K-means algorithm iterations until final clustering result.

Source: Page, Justin T, Zachary S Liechty, Mark D Huynh, and Joshua A Udall. 2014. "BamBam: genome sequence analysis tools for biologists." BMC Research Notes, November.

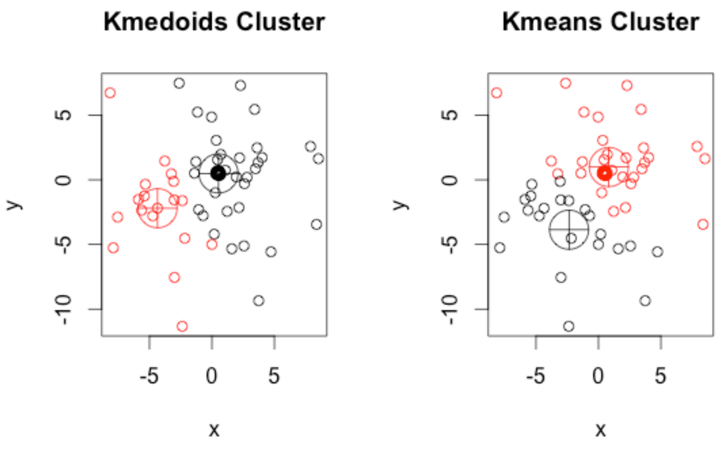
K-medoids clustering

K-medoids clustering is a slightly modified version of the K-means algorithm that is more robust to outliers and noise. K-medoids clustering uses an actual point in the cluster to represent it instead of using the mean point as the center of a cluster. The basic principle of this algorithm is that it aims to minimize the sum of dissimilarities (sum of distances) between each object/data point and its corresponding reference point. The most centrally located object/data point of the cluster has the minimum sum of distances to other points and is called a medoid. A drawback of K-Medoids clustering in that is does not always generate the same result with each run, as the resulting clusters are subject to the initial assignments that are random.

The discrete steps of the K-medoids algorithm are as follows:

2. K cluster medoids are initialized. This can be done by randomly selecting K data points to act as the initial medoids.
3. The sum of the squared distance between data points and all medoids is calculated and then each data point is assigned to the closest cluster/medoid.
4. A non-medoid object Orandom is selected.
5. The total points S of swaping object Oj with Orandom are computed
6. If S<0 then Oj is swapped with Orandom (the medoid is updated)
7. The algorithm continues iterating until the medoids stay the same between two runs of the algorithm.

Source: https://www.academia.edu/8446443/K_Medoid_Clustering_Algorithm_A_Review



Example where the output of the k-medoid algorithm is different than the output of the k-means algorithm

Source: https://stats.stackexchange.com/questions/156210/an-example-where-the-output-of-the-k-medoid-algorithm-is-different-than-the-outp

Partitioning K-means

Partitioning K-Means is a modified version of the K-means algorithm that is designed to beeter work with larger datasets. In the standard K-Means algorithm each iteration computes the distances between all data points and all centroids. This can be computationally heavy especially for large datasets. In Partitioning K-Means instead of updating all existing centroids in every iteration, a subgroup of data points is selected randomly so that only the centroids of the clusters that contain these data points are then updated. This subgroup of data points is mentioned as a mini-batch. On the other hand, using mini-batches can also lead to some noise into the clustering procedure because only a subgroup of the data is used to determine if the centroids must be updated. The mini-batch size and the frequency of updating the centroids can have an impact on the quality of the clustering results as smaller mini-batch sizes can lead to faster convergence while also leading to lower quality clustering outputs.

The discrete steps of the Partitioning K-Means algorithm are as follows:

1. In the beginning, the centroids are initialized by randomly selecting k centroids from the data points. These centroids will act as the initial cluster centers.

2. The data is then divided into mini-batches which are subgroups of the original dataset so that each mini-batch/subgroup contains only a portion of the entire data.

3. Then for each mini-batch/subgroup of the data:

    a. Each data point is assigned to the nearest centroid: More specifically, each data point in the mini-batch/subgroup is assigned to the nearest centroid using the same distance metric as in the original K-Means algorithm (e.g., Euclidean distance). Therefore, a set of mini-clusters is produced.

    b. The mini-centroids are updated as for each mini-cluster, the Partitioning K-Means algorithm calculates the mean of all the data points in the mini-cluster and it then updates the mini-centroid.

    c. The mini-clusters are merged after all the mini-batches have been processed to form the final clusters. This can be achieved by assigning each data point to the nearest centroid from all the existing mini-centroids.

4. The same steps are repeated until convergence, as the algorithm uses mini-batches to update the centroids in each iteration until convergence. When the centroids no longer move/change between different iterations or when a maximum number of iterations is reached, then convergence is achieved and the algorithm stops iterating.

One key variable in Partitioning K-Means is the size of the mini-batches as a larger mini-batch size can improve the clustering accuracy while also requiring more computational power. A smaller mini-batch size on the other hand can reduce computer memory requirements and computational cost but it leads to inferior clustering accuracy and also slower convergence of the Partitioning K-Means algorithm. Another important variable is the number of iterations that the algorithm is meant to run until a solution is reached. Similar to the original K-Means algorithm, Partitioning K-Means can converge to a local minimum. By running the Partitioning K-Means algorithm several times while choosing different initial centroids, the likelihood of finding the global minimum is increased.

Sources:

https://www.geeksforgeeks.org/partitioning-method-k-mean-in-data-mining/

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html

https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that can find groups of data points and create clusters that are compactly and closely grouped together. The algorithm also has the capability to recognize the points that do not belong to any existing cluster. Those points are called noise.

The DBSCAN algorithm works by firstly defining a distance (which is referred to as Eps) and a minimum number of points (which is referred to as MinPts) that need to be within that distance of a point in order for it to be considered to be a part of a cluster. After that the DBSCAN algorithm starts at a point that is chosen randomly and in the case that there are enough points within Eps distance of that point, it creates a cluster around it. After that, the algorithm then moves to the next point and if it is within Eps distance of a previously identified cluster, it is also added to that cluster. If it is not within Eps distance of any existing identified cluster but there are enough points within Eps distance of that point, it becomes the seed for a new cluster. In case that there are not enough points within Eps distance of that point, it is then labeled as noise.

The DBSCAN algorithm has several advantages when directly compared to other widely used clustering algorithms:

- Firstly, it does not require the user to provide the number of clusters existing in the dataset beforehand.
- Secondly, it is capable to distinguish points that do not belong to any existing cluster (that are referred to as noise).
- Lastly, it is able to discover and form clusters of any shape as long as they are dense (data points are compactly and closely grouped together).

On the other hand, one disadvantage of the DBSCAN algorithm is that it can be sensitive to the values chosen for Eps and MinPts (these values are given from the user as an input to the algorithm). It is also not suitable to be used with high-dimensional datasets, because the distance between points is less meaningful as the number of dimensions increases.

Source: https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556

Gaussian Mixture Model

GMM (Gaussian Mixture Model) is a probabilistic model that makes the assumption/hypothesis that all the data points/observations of the given dataset are a mix of multi-dimensional Gaussian probability distributions with unknown parameters. The GMM algorithm uses the Expectation-Maximization (EM) algorithm to estimate the parameters of the Gaussian distributions and also the number of existing clusters. The EM algorithm iteratively estimates the probability of each data point belonging to each cluster (this step is referred to as the E-step) and then updates the parameters of each cluster using the estimated probabilities (this step is referred to as the M-step), until finally convergence is reached.

The objective function of GMM is to maximize the probability p(X) for the data X, or the log-likelihood value L. By approximating that a mixture of K Gaussian distributions have produced the data, p(X) is written as marginalized probability summed over all K clusters for all available data points.

$$p(x_i) = \sum_{k=1}^{K} p(x_i|c_k)p(c_k)$$

$$p(X) = \prod_{i=1}^{N} p(x_i) = \prod_{i=1}^{N} \sum_{k=1}^{K} p(x_i|c_k)p(c_k)$$

$$L = log(p(X)) = \sum_{i=1}^{N} log(\sum_{k=1}^{K} p(x_i|c_k)p(c_k))$$

Because we do not have the ability to get an analytical solution for the summation inside the log function above, we make use of the Expectation Maximization (EM) algorithm. The Expectation Maximization algorithm is an iterative algorithm that can be used to find maximum likelihood estimates (MLE) of models where parameters cannot be calculated in a direct way. The EM algorithm consists of two stages which are the expectation step and the maximization step.

In the Expectation Step the membership values r_{ic} are computed. This value represents the probability that data point x_i belongs to the cluster c.

$$r_{ic} = p(y_c|x_i) = \frac{p(y_c, x_i)}{p(x_i)} = \frac{p(x_i|y_c)p(y_c)}{\sum_{c=1}^{K} p(x_i|y_c)p(y_c)}$$

$$p(y_c) := \pi_c, p(x|c_k) = N(x|\mu_k, \sum_k) \implies r_{ic} = \frac{\pi_c N(x_i|\mu_c, \sum_c)}{\sum_{c=1}^{K} N(x_i|\mu_c, \sum_c)}$$

In the Maximization Step a new parameter m_c is calculated, which controls the portion of points assigned to different clusters. By calculating Maximum Likelihood Estimators (MLE's) for every cluster c, the parameters μ, π, Σ are updated.

$$m_c = \sum_{i=1}^{N} r_{ic}$$
$$\pi_c = \frac{m_c}{N}$$
$$\mu_c = \frac{1}{m_c} \sum_{i=1}^{N} r_{ic} x_i$$
$$\sum\nolimits_c = \frac{1}{m_c} \sum_{i=1}^{N} r_{ic}(x_i - \mu_c)^T (x_i - \mu_c)$$

The E-M steps are repeated until the log-likelihood value L finally converges.

One of the key advantages of the Gaussian Mixture Model algorithm compared to other clustering algorithms is that it can perform better when working with more complex cluster shapes because it does not make the assumption/hypothesis that the clusters have a spherical shape so it does not have a bias for circular clusters.

Sources:

https://towardsdatascience.com/gaussian-mixture-models-gmm-6e95cbc38e6e

https://scikit-learn.org/stable/modules/mixture.html

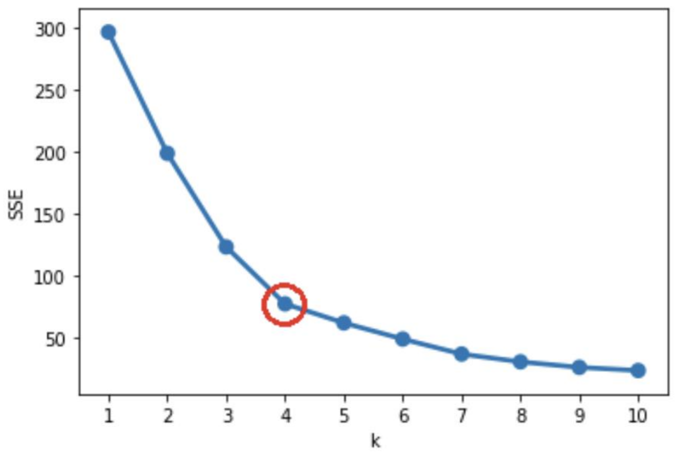
https://www.sciencedirect.com/topics/engineering/gaussian-mixture-model

Determination of the number of clusters

The elbow method is a widely used method for determining the optimal number of clusters in a dataset. This is a method used when working with unsupervised learning algorithms like clustering.

The Sum of Squared Errors (SSE) is a measure of how capable the clustering algorithm is to form clusters with similar data points. In the case of k-means clustering, the SSE is calculated as the sum of the squared distances between each data point and its closest centroid. A lower SSE value therefore indicates that the clusters are more closely grouped together and that the data points that belong to each cluster are more similar.

The Bayesian Information Criterion (BIC) is a metric of the quality of fit of the model to the data in GMM (Gaussian Mixture Model) clustering, with a lower BIC representing a better fit. The BIC value is calculated as the log-likelihood of the data under the model plus a penalty term for the number of parameters in the model. The BIC metric penalizes the model for its complexity so that more complex models will have a worse score.



Elbow method example

Source: https://s3.amazonaws.com/assets.datacamp.com/production/course_10628/slides/chapter4.pdf

In order to find the optimal number of clusters to provide as an input to a clustering algorithm, the SSE/BIC is plotted as a function of k and the elbow point can be recognized as the point where the SSE/BIC begins to decrease at a slower rate, meaning that the higher number of clusters are not significantly improving the clustering performance.

Sources:

https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/

https://www.researchgate.net/publication/339670247_Determining_The_Appropiate_Cluster_Number_Using_Elbow_Method_for_K-Means_Algorithm

https://www.researchgate.net/figure/Elbow-plots-of-the-Bayesian-information-criterion-BIC-and-consistent-Akaike-information_fig2_347946003

https://machinelearningmastery.com/probabilistic-model-selection-measures/

Chapter 3 - Assessment metrics for clustering algorithms

Evaluating the results of a clustering algorithm is a crucial part of the process of clustering data. Assessment metrics are used to assess the performance of clustering algorithms by comparing the clustering results to a known truth (true cluster labels) or by measuring the quality of the clustering results based on different criteria. There are numerous types of assessment metrics that are used to evaluate the performance of clustering algorithms and each clustering assessment metric contains a mix of different criteria to measure the quality of the clustering results. These metrics can be split into three distinct categories:

- **External assessment metrics:** These metrics evaluate the performance of a clustering algorithm by comparing the clusters formed by the algorithm to the actual/real clustering labels. As an example, the Adjusted Rand Index (ARI) measures the similarity between the clustering produced by the algorithm and a reference clustering result (real clustering labels). These metrics are not applicable to typical clustering problems in which the real class labels are not available.

- **Internal assessment metrics:** These metrics evaluate the performance of a clustering algorithm based only on the information provided by the dataset itself and the clustering results. In this case there is no use of external information. For example, the Silhouette Coefficient measures the compactness and separation of the clusters formed by the algorithm, while the Calinski-Harabasz Index measures the ratio of between-cluster variance to within-cluster variance.

- **Relative assessment metrics:** These metrics compare the performance of different clustering algorithms that are applied on the same dataset. For example, the Normalized Mutual Information (NMI) measures the mutual information that is shared by the clustering result produced by two different clustering algorithms, while the Adjusted Mutual Information (AMI) corrects for coincidental agreements between the clustering result of two algorithms.

Some common assessment metrics that are used are:

- Adjusted Rand Index (ARI)
- Adjusted Mutual Information (AMI)
- Normalized Mutual Information (NMI)
- Silhouette Coefficient
- Calinski-Harabasz Index (CHI)
- Davies-Bouldin Index (DBI)
- Dunn Index (DI)

It must be highlighted that the choice of the appropriate assessment metric will be subject to each individual problem and also the type of clustering algorithm being assessed. Making use of multiple clustering performance metrics is advised in order to get a more complete picture of the clustering algorithm's performance.

Sources:

https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods/

http://www.sthda.com/english/wiki/wiki.php?id_contents=7952

https://blog.paperspace.com/ml-evaluation-metrics-part-2/

Adjusted Rand Index (ARI)

The Adjusted Rand Index is used to measure whether two cluster results are similar to each other. It takes into consideration all pairs of samples and measures how often they are assigned to the same or different clusters in the two distinct clustering results. The ARI ranges from -1 to 1, where 1 indicates perfect agreement between the two cluster results (the two cluster results are same), 0 indicates that the points are assigned into clusters randomly, and -1 indicates complete disagreement between the two clustering results. The ARI formula is as follows:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}$$

where RI stands for the Rand Index and E[RI] is the expected value of the Rand Index. The Rand Index calculates the percentage of pairs of samples that are either in the same cluster in both clustering results or in different clusters in both clustering results. The Adjusted Rand Index is obtained by subtracting to the RI an estimator of its expected value obtained under the assumption of two independent clustering results.

Source: https://www.sciencedirect.com/topics/computer-science/adjusted-rand-index

Adjusted Mutual Information (AMI)

Adjusted Mutual Information (AMI) is a measure of the similarity between two clustering results and it is a normalized version of Mutual Information (MI), which measures the quantity of information that one clustering result offers about the other. It corrects the effect of coincidental agreement between clustering results with a factor that takes into consideration the expected mutual information with a random labeling of the data. This is comparable to the way the Adjusted Rand Index (ARI) corrects the Rand Index (RI). The range of values of AMI is from 0 to 1, with 0 representing that the two clustering results are independent and 1 representing that they are matching. The AMI formula is as follows:

$$AMI(X,Y) = \frac{MI(X,Y) - E[MI(X,Y)]}{max(H(X),H(Y)) - E[MI(X,Y)]}$$

Where:

X and Y are the two clustering results,

MI(X, Y) is their mutual/shared information,

H(X) and H(Y) are the entropies of X and Y,

E[MI(X, Y)] is the expected mutual information with a random labeling of the data

Sources:

https://jmlr.csail.mit.edu/papers/volume11/vinh10a/vinh10a.pdf

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html#sklearn.metrics.adjusted_mutual_info_score

Normalized Mutual Information (NMI)

The Normalized Mutual Information is a measure of the amount of information that is mutual/common between two clustering results. NMI is based on Mutual Information and includes a normalization factor. The value of NMI ranges from 0 to 1, with a higher value indicating a better performance. NMI is not adjusted for chance which means that even a random clustering result would be given positive score. The NMI formula is as follows:

$$NMI = \frac{2 * (MI(C,K))}{H(C) + H(K)}$$

Where:

MI(C, K) is the mutual information between the two clustering results,

H(C) is the entropy of the true class labels,

H(K) is the entropy of the predicted cluster labels

Sources:

https://luisdrita.com/normalized-mutual-information-a10785ba4898

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html

Silhouette Coefficient

The Silhouette Coefficient is a clustering assessment metric that measures of the compactness of a single cluster and the separation between distnct clusters. The metric takes values in the range [-1,1], with a value closer to 1 representing superior clustering performance. The silhouette coefficient is calculated using the following formula:

$$SC = \frac{b - a}{max(a, b)}$$

Where:

a is the average distance between the sample/data point and all other samples/data points in the same cluster,

b is the minimum average distance between the sample/data point and all samples/data points in a different cluster.

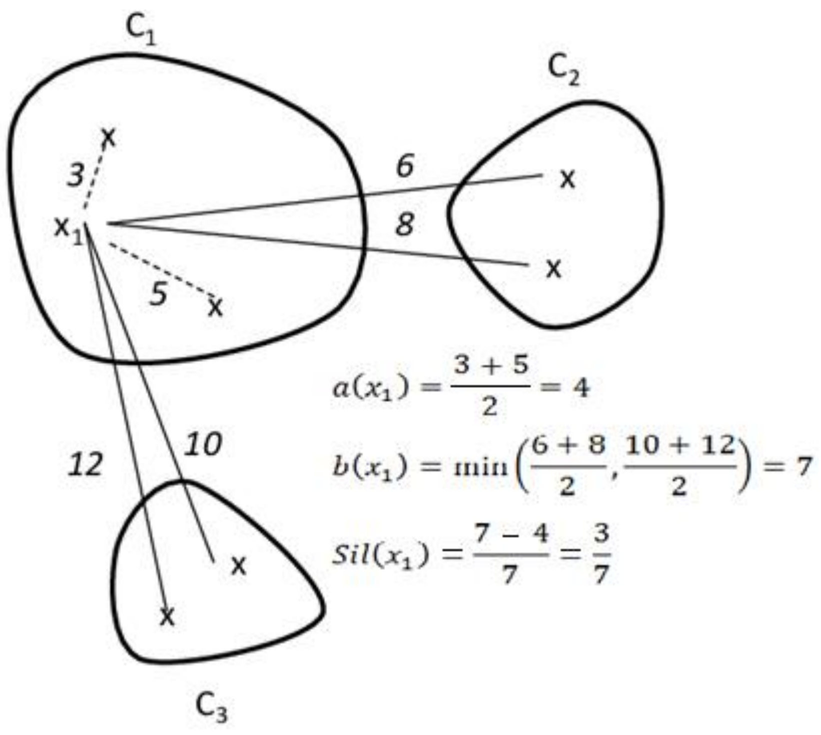
Clustering algorithms like K-means, K-medoids etc. expect the number of clusters k the data has to be predefined as an input from the user to the algorithm. This is the primary reason why the the silhouette score method is implemented. By implementing the silhouette score, the user chooses a range of clusters k and the silhouette method calculates a score for each one by applying the clustering algorithm to the dataset. The silhouette value/score represents how similar a data point is to the cluster that it assigned to compared to other existing clusters.

The silhouette score of a specific datapoint i is defined as:

$$s_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

Where:

- a_i is defined as the mean distance between the datapoint i and all other data points the exist in the same cluster (a smaller value is better)
- b_i is defined as the smallest mean distance of the datapoint i to all points in any other existing cluster, of which i is not a member of (a large value shows that the datapoint is not well matched to the neighboring cluster)


C1
C2
C3
x1
3
5
6
8
10
12
$a(x_1) = \frac{3+5}{2} = 4$
$b(x_1) = \min\left(\frac{6+8}{2}, \frac{10+12}{2}\right) = 7$
$Sil(x_1) = \frac{7-4}{7} = \frac{3}{7}$

Example: Calculation of the silhouette coefficient $Sil(x_1)$ for the datapoint $x_1$that is assigned to the cluster C_1.

Source: https://www.researchgate.net/figure/Derivation-of-the-Overall-Silhouette-Coefficient-OverallSil_fig1_221570710

The Euclidean distance or the Manhattan distance can be used as a distance metric for the silhouette score calculation. A high value of the silhouette metric means that each datapoint is well matched to the cluster that it is assigned to and badly matched to its neighboring clusters. The mean s_i of all datapoints existing in a dataset is a measure of how appropriate/correct the clustering result is.

Source: https://www.sciencedirect.com/science/article/pii/0377042787901257?via%3Dihub

Calinski-Harabasz Index

The Calinski-Harabasz Index is a clustering evaluation measure that is used to evaluate the quality of a clustering solution and is also known as the Variance Ratio Criterion. It is calculated as a ratio of the sum of inter-cluster dispersion (between-cluster variance) and the sum of intra-cluster dispersion (within-cluster variance) for a given clustering output. It calculates the ratio of the sum of squares between clusters to the sum of squares within clusters, multiplied by the ratio of the number of data points to the number of clusters minus one. The Calinski-Harabasz Index outputs a score for each clustering result, with higher scores representing a superior clustering solution. This score can be used to compare different clustering methods or to find the optimal number of clusters in a given data set.

The between-cluster sum of squares or between-cluster variance shows the degree to which the centroids of different clusters are separated from each other. The formula for BCSS is as follows:

$$BCSS = \sum_{k=1}^{K} n_k \, ||C_k - C||^2$$

Where:

n_k is the number of data points in the cluster k,

C_k is the centroid of cluster k,

C is the centroid of all the data points,

K is the number of clusters

The within-cluster sum of squares or within-cluster variance shows the compactness of the clusters themselves. The formula for WCSS is defined as follows:

$$WCSS = \sum_{k=1}^{K} WGSS_k = \sum_{k=1}^{K} \sum_{i=1}^{n_k} || X_{ik} - C_k||^2$$

Where:

n_k is the number of data points in the cluster k,

X_{ik} : the i^{th} observation of cluster k,

C_k is the centroid of cluster k,

K is the number of clusters

The Calinski-Harabasz Index is then calculated with the following formula:

$$CH(k) = \frac{\frac{BCSS}{K-1}}{\frac{WCSS}{N-K}} = \frac{BCSS}{WCSS}\,\frac{N-K}{K-1}$$

Where:

K is the number of clusters,

N is the total number of samples,

BCSS is the between-cluster sum of squares (between-cluster variance),

WCSS is the within-cluster sum of squares (within-cluster variance)

A higher value of CH(k) represents better clustering quality, since it shows that the centroids of different clusters are well-separated (between-cluster variance) and the data points within each cluster are compact (within-cluster variance). The Calinski-Harabasz Index has the advantage of being relatively easy and fast to compute and easy to interpret. One limitation is that the index is designed to work with convex clusters, and may not work as well with non-convex or irregularly shaped clusters. As the index is not normalized, the value is influenced by the number of clusters and the number of observations. The Calinski-Harabasz Index values can range from 0 to several hundred.

Sources:

https://pyshark.com/calinski-harabasz-index-for-k-means-clustering-evaluation-using-python/

Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. Communications in Statistics-theory and methods

Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2001). On clustering validation techniques. Journal of intelligent information systems

Davies-Bouldin Index

The Davies-Bouldin Index (DBI) is another clustering evaluation measure that is used to measure the quality of a clustering solution. The Davies-Bouldin Index measures the similarity between clusters and the dissimilarity between clusters for a specific clustering result. It calculates the average similarity between each cluster and a cluster that is most similar, divided by the average dissimilarity between each cluster and its least similar cluster.

The Davies-Bouldin Index outputs a score for each clustering result. Lower scores indicate better clustering results. It can be used to compare different clustering algorithms or to find the ideal number of clusters in a group of data. One main advantage of the Davies-Bouldin Index is that it does not assume any specific size or shape for the clusters and it does not give a higher score for clustering results with a greater number of clusters. Another advantage is that the index is easy to calculate and interpret. In order to calculate the DBI, we follow the steps described below:

Step 1: The intra-cluster dispersion is calculated

$$S_i = \left\{ \frac{1}{T_i} \sum_{j=1}^{T_i} |X_j - A_i|^q \right\}^{\frac{1}{q}}$$

where:

i is the particular identified cluster,

Ti is the number of vectors (observations) in cluster i,

Ti is the number of vectors (observations) in cluster i,

Xj is the j-th vector (observation) in cluster i,

Ai is the centroid of cluster i

The intra-cluster dispersion is the average distance between each observation within the cluster and its centroid. When the value q is set to 2 it calculates the Euclidean distance.

Step 2: The separation measure is calculated:

$$M_{ij} = \left\{ \sum_{k=1}^{N} |a_{ki} - a_{kj}|^p \right\}^{\frac{1}{p}} = ||A_i - A_j||_p$$

where:

a_{ki} is the k-th component of n-dimensional centroid Ai

a_{kj} is the k-th component of n-dimensional centroid Aj

N is the total number of clusters

When P is set equal to 2 so the above formula, it calculates the Euclidean distance between the centroids of clusters i and j.

Step 3: The similarity between clusters is calculated:

The similarity between clusters is calculated as a sum of two intra-cluster dispersions divided by the separation measure.

$$R_{ij} = \frac{S_i + S_j}{M_{ij}}$$

where:

S_i is the intra-cluster dispersion of cluster i

S_j is the intra-cluster dispersion of cluster j

M_{ij} is the distance between centroids of clusters i and j

Step 4: The most similar cluster for each cluster i is found:

$$R_i \equiv maximum(R_{ij})$$

having: i≠j

For each cluster i we find the highest ratio from all R_{ij} calculated

Finally, the mathematical formula for DBI is written as follows:

$$\bar{R} = \frac{1}{N}\sum_{i=1}^{N} R_i$$

which is the average of the similarity measures of each cluster with the cluster most similar to it.

To summarize the above steps, for each cluster i, the DBI calculates the average distance between each point in that cluster and the centroid of the cluster. It then calculates the same measure for all other clusters j, and takes the maximum value over all j. Then this quantity is divided by the distance between the centroids of clusters i and j, which measures how separated the clusters are. DBI gives an overall measure of the quality of the clustering result by averaging this quantity over all clusters i. It must be noted that different choices of distance metrics can give different results for the DBI. One restriction/limitation of the Davies-Bouldin Index is that it accepts that the clusters are spherical and have equal variance. The index may also not perform as well when the clusters have very different densities or sizes. The Davies-Bouldin Index ranges from 0 to infinity, with lower values representing superior clustering solutions.

Sources:

https://pyshark.com/davies-bouldin-index-for-k-means-clustering-evaluation-in-python/

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence

Dunn Index

The Dunn Index (DI) is another clustering evaluation metric that is used to evaluate the quality of a clustering results. The Dunn Index evaluates the compactness of clusters and the separation between clusters for a given clustering solution by calculating the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance. A high DI shows a better clustering since the observations in each cluster are closer together while the clusters are further away from each other. This means that the clusters are well separated. The DI can be used to compare different clustering algorithms or to find the best number of clusters in a given data set.

To calculate the Dunn Index for a given clustering solution, the minimum distance between any two data points belonging to different clusters is computed. After that, the maximum distance between any two data points belonging to the same cluster is computed. Finally, the minimum inter-cluster distance is divided by the maximum intra-cluster diameter in order to obtain the Dunn Index.

The formula for Dunn Index is written as follows:

$$DI = \frac{Min\ inter - cluster\ distance}{Max\ intra - cluster\ diameter}$$

where:

> Min inter-cluster distance is the minimum distance between any two data points belonging to different clusters,
>
> Max intra-cluster diameter is the maximum distance between any two data points belonging to the same cluster.

One disadvantage of the Dunn Index is that it can be sensitive to outliers because it is based on distances between data points. Also, the index may have not work as well when the clusters have significantly different densities and/or sizes. The range of the Dunn Index is theoretically without limits, but typically it can range between 0 and 1. The exact range of the Dunn Index can be different depending on the dataset and the number of clusters.

Sources:

https://pyshark.com/dunn-index-for-k-means-clustering-evaluation-using-python/

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. Journal of Cybernetics

Chapter 4 - Implementation strategy

The aim of this project is to develop a Python code from scratch that will demonstrate the above elements. The various tools that were used for the development of the script are presented below:

Tools

Python is a popular programming language used for a broad range of applications like data analysis, scientific computing and machine learning. Machine learning algorithms and tools for data preprocessing, model selection, data analysis and evaluation are provided in Python through libraries. Some of the most useful libraries for working with machine learning techniques in Python are:

- Scikit-learn is a machine learning library that supports supervised and unsupervised learning and gives access to a wide range of algorithms for clustering, classification, regression and dimensionality reduction while also including tools for data preprocessing, feature extraction, and model selection.

- Pandas is a library for data analysis and manipulation that gives access to tools for working with structured data. Pandas is used for preprocessing and cleaning data before using it as an input to machine learning models.

- NumPy is a library for numerical computing in Python that gives access to tools for working with arrays and matrices. NumPy is used in machine learning for data preprocessing and feature extraction.

- Matplotlib is the basic plotting library of the Python programming language and is used for creating static, animated, and interactive visualizations.

These libraries and tools make it easy to implement different machine learning algorithms and techniques in Python and create visualizations showing the output of the machine learning algorithms in a more presentable way.

Sources:

https://scikit-learn.org/stable/getting_started.html

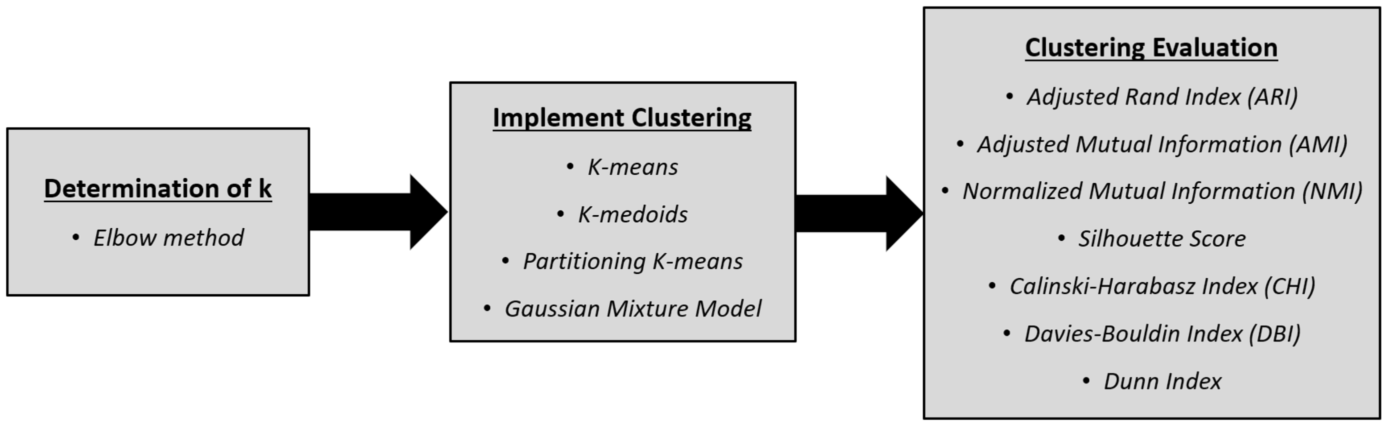
https://pandas.pydata.org/

https://numpy.org/

https://matplotlib.org/

Main implementation steps

The main part of the work concerns the development of software that will perform the following steps for a real dataset:



Steps list:

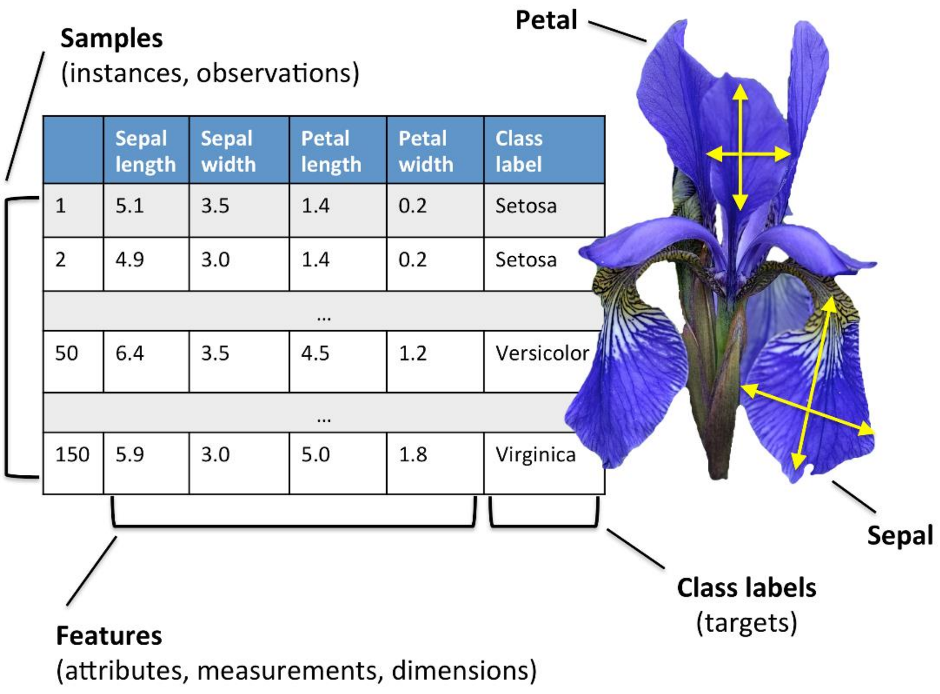
1. Load selected dataset, in this case the iris dataset.
2. Standardize the data by scaling it (to have zero mean and unit variance)
3. Perform clustering on the dataset using 4 different clustering algorithms. Each algorithm is either loaded from the scikit-learn library or created from scratch and defined in the start of the script. Although the DBSCAN algorithm was presented in Chapter 3, it won’t be used in this part of the analysis as it required to be previously tuned in order to ensure that we have the best possible clustering result.
4. Plot the SSE for the k-means family algorithms and BIC values for GMM for different number of clusters and use the Elbow Method to find the optimal number of clusters for all algorithms. The user will have to input the selected k number.
5. After finding the optimal number of clusters, the script fits the clustering algorithms on the data and plots the results. It also plots scatter matrix to visualize the results.
6. The various performance metrics like Adjusted Rand Index, Adjusted Mutual Information, Silhouette Score, etc. are calculated for all the algorithms. The different metrics are displayed in a bar plot for each clustering algorithm. The results will be accompanied by relevant commentary.

Dataset description

The script is designed to take different datasets as an input with minor adjustments. In order to demonstrate the capabilities of the software in a more visual and clear way, a simple dataset appropriate for clustering will be chosen.

The Iris flower dataset or Fisher's Iris dataset is a multivariate data set introduced by the British statistician and biologist Ronald Fisher. There are 4 attributes in this dataset (Four features were measured from each sample). These attributes are:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
…					
50	6.4	3.5	4.5	1.2	Versicolor
…					
150	5.9	3.0	5.0	1.8	Virginica

Samples
(instances, observations)
Petal
Sepal
Class labels
(targets)
Features
(attributes, measurements, dimensions)

Dataset preview (Source: https://www.mghassany.com/courses/MLcourse/pw-6.html)

The data set contains 3 classes (Iris-setosa, Iris-versicolor, Iris-virginica) of 50 instances each, where each class refers to a type of iris plant. This dataset was chosen as it simple enough so the clustering results can be plotted without having the need for dimensionality reduction techniques (like PCA). It must be noted that the class label data will not be used as an input to the clustering algorithms. This column will be used only for the calculation of the external and relative clustering metrics

Chapter 5 – Script presentation and results

Script presentation

The intended output/result of the Thesis is the development of a software script capable to take different datasets as an input, implement different clustering methods and export the clustering performance metrics. The correct operation of the developed script is shown in this chapter where the iris dataset is used as an example to showcase the capabilities of the script.

The python script was created using the web-based interactive computing platform Jupyter Notebook (https://jupyter.org/install). Jupyter Notebook that allows the user to create documents comprising of code visualizations and text. It is very useful for data analysis and machine learning tasks as it allows you run independent executable code cells in any order and observe the output immediately.

This code imports several libraries and modules from the scikit-learn package, as well as some other standard Python libraries like matplotlib and numpy. It also imports a specific function euclidean_distances from sklearn.metrics.pairwise module. Here's a brief description of what each imported module and function does:

- datasets: A module in scikit-learn that provides some standard datasets for testing and experimenting with machine learning algorithms.
- StandardScaler: A class in scikit-learn that scales input data to have zero mean and unit variance, which can be beneficial for some machine learning algorithms.
- GaussianMixture: A class in scikit-learn that fits a mixture of Gaussian distributions to a given dataset using the expectation-maximization algorithm.
- KMeans: A class in scikit-learn that performs K-Means clustering on a given dataset.
- KMedoids: A class in scikit-learn-extra that performs K-Medoids clustering on a given dataset. K-Medoids is similar to K-Means, but instead of computing the mean of each cluster, it selects a medoid, which is the data point that is closest to the center of the cluster.
- metrics: A module in scikit-learn that provides various evaluation metrics for machine learning models.
- plt: A sub-library in matplotlib used for plotting graphs and visualizations.
- numpy: A library for numerical computing in Python.
- pandas: A library for data manipulation and analysis in Python.

Source: https://scikit-learn.org/stable/user_guide.html

The euclidean_distances function calculates the pairwise Euclidean distances between the rows of a given matrix.

```python
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import euclidean_distances
import random
```

This code defines a function named plot_clustering_results that can be used to visualize the results of various clustering algorithms on a given dataset. It that takes in five arguments:

- algorithm: a string that indicates the name of the clustering algorithm used
- data: a 2D array of the data points being clustered
- labels: a 1D array that assigns each data point to a cluster
- centroids: a 2D array that represents the centroids of the clusters
- n_attributes: an integer that specifies the number of attributes of the data points.

The function first creates a Pandas DataFrame from the data and sets the column names using the feature names from the iris dataset. Next, the function creates a scatter matrix plot using the pd.plotting.scatter_matrix function from the Pandas plotting library. The color of each data point is set by its assigned cluster label. The plot is created using a subplot with size (n_attributes x n_attributes). The function then creates a Pandas DataFrame from the centroids and plots the centroids on the scatter matrix plot. For each subplot, the function plots a red 'x' at the position of each centroid. The position of the 'x' is determined by the value of the centroid along the x-axis and y-axis for that particular subplot. Finally, the function adds a title to the plot using the algorithm name and displays it.

```python
def plot_clustering_results(algorithm, data, labels, centroids, n_attributes):
    iris_df = pd.DataFrame(data,columns=iris.feature_names)

    fig, axs = plt.subplots(n_attributes, n_attributes, figsize=(15, 15))
    pd.plotting.scatter_matrix(iris_df, c=labels, ax=axs, marker='o',
                               hist_kwds={'bins': 20}, s=60, alpha=.8, diagonal='hist')

    centroids_df = pd.DataFrame(centroids, columns=iris.feature_names)
    for i, ax in enumerate(axs.flatten()):
        if i // n_attributes == i % n_attributes:
            for j in range(centroids_df.shape[0]):
                pass  # don't plot the vertical lines along the diagonal
        else:
            for j in range(centroids_df.shape[0]):
                ax.scatter(centroids_df.iloc[j,i%n_attributes], centroids_df.iloc[j,i//n_attributes], s=60, c='red', marker='x')

    plt.suptitle(algorithm, fontsize=20)
    plt.show()
```

This code defines a class called PartitioningKMeans which implements the Partitioning K-means algorithm for clustering data.

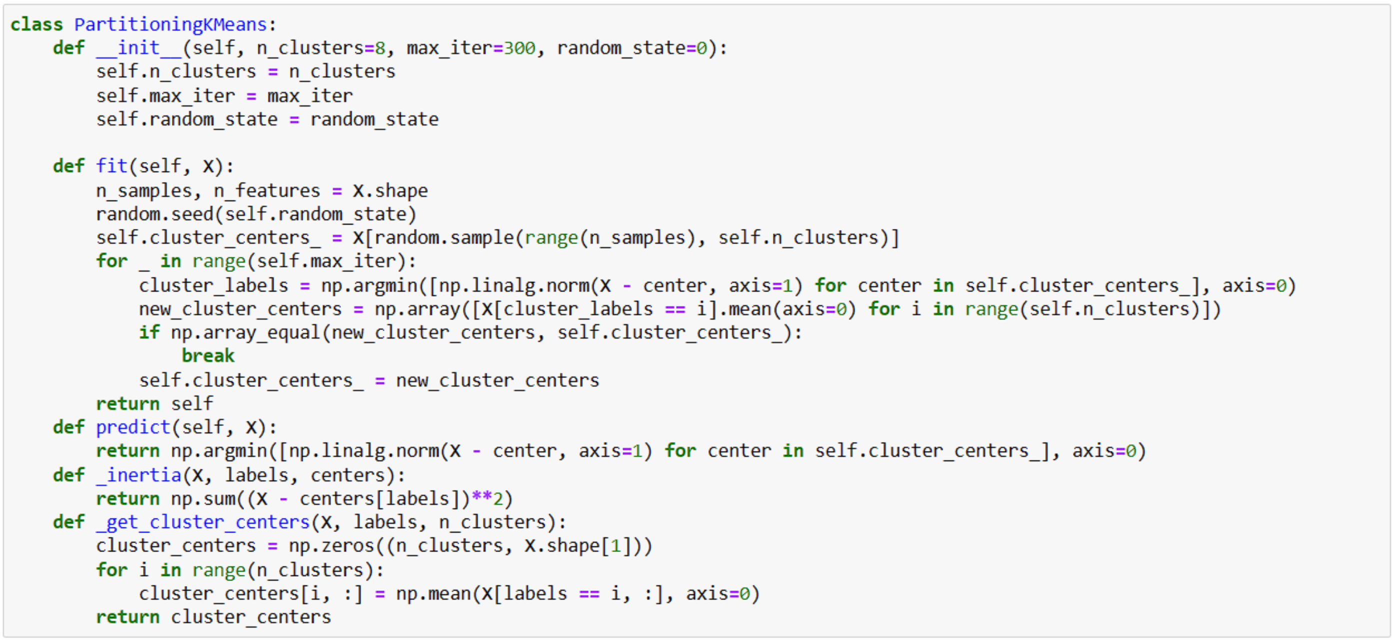
The __init__ method initializes the hyperparameters of the algorithm: n_clusters determines the number of clusters to form, max_iter determines the maximum number of iterations the algorithm should perform, and random_state sets the seed value for the random number generator, which ensures that the results are reproduceable.

The fit method takes a dataset X as input and performs K-means clustering. The algorithm starts by randomly initializing the cluster centers and then updates iteratively the cluster assignments and the cluster centers until convergence is reached. The cluster assignments are made based on the Euclidean distance between the data points and the cluster centers. The algorithm stops when either the maximum number of iterations is reached or the cluster centers stop changing between the last two iterations. The fit method returns the fitted PartitioningKMeans object.

The predict method takes a dataset X as input and returns the cluster assignments for each data point based on the cluster centers.

The _inertia method is a function that calculates the sum of squared distances between each data point and its assigned cluster center.

The _get_cluster_centers method is another function that calculates the coordinates of the cluster centers based on the assigned cluster labels.



```python
class PartitioningKMeans:
    def __init__(self, n_clusters=8, max_iter=300, random_state=0):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.random_state = random_state

    def fit(self, X):
        n_samples, n_features = X.shape
        random.seed(self.random_state)
        self.cluster_centers_ = X[random.sample(range(n_samples), self.n_clusters)]
        for _ in range(self.max_iter):
            cluster_labels = np.argmin([np.linalg.norm(X - center, axis=1) for center in self.cluster_centers_], axis=0)
            new_cluster_centers = np.array([X[cluster_labels == i].mean(axis=0) for i in range(self.n_clusters)])
            if np.array_equal(new_cluster_centers, self.cluster_centers_):
                break
            self.cluster_centers_ = new_cluster_centers
        return self
    def predict(self, X):
        return np.argmin([np.linalg.norm(X - center, axis=1) for center in self.cluster_centers_], axis=0)
    def _inertia(X, labels, centers):
        return np.sum((X - centers[labels])**2)
    def _get_cluster_centers(X, labels, n_clusters):
        cluster_centers = np.zeros((n_clusters, X.shape[1]))
        for i in range(n_clusters):
            cluster_centers[i, :] = np.mean(X[labels == i, :], axis=0)
        return cluster_centers
```

This code defines a function called davies_bouldin_index that computes the Davies-Bouldin Index for a clustering. It is computed as the average similarity between each cluster and its most similar cluster (similarity is defined as the ratio of the within-cluster distance to the between-cluster distance).

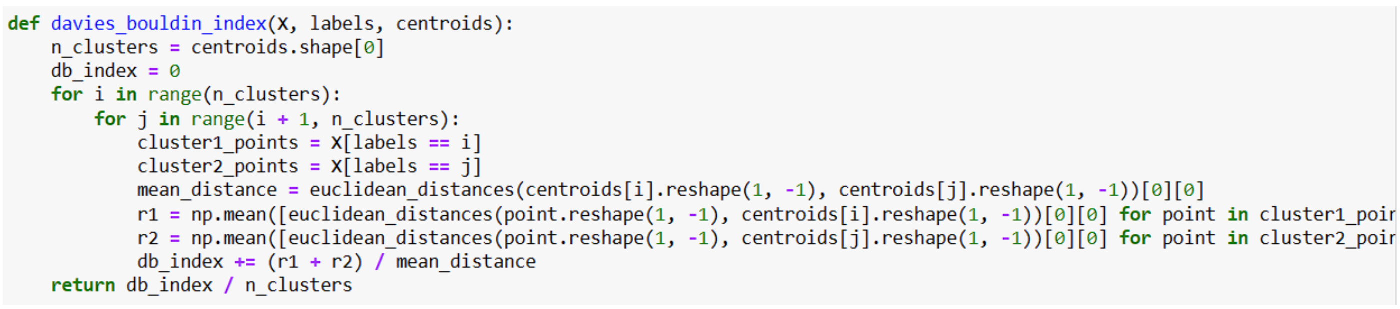
The function takes three inputs:

- X: a numpy array of shape (n_samples, n_features) representing the data points
- labels: a numpy array of shape (n_samples,) representing the cluster assignments for each data point
- centroids: a numpy array of shape (n_clusters, n_features) representing the coordinates of the cluster centers

The function first calculates the number of clusters n_clusters based on the number of rows in the centroids array. It then initializes the db_index variable to zero.

The function then iterates over all pairs of clusters and calculates the mean distance between their centroids. For each cluster, it calculates the mean distance from each data point in the cluster to its centroid. It then adds the ratio of the sum of these two distances to the mean distance between the two cluster centroids to db_index.

Finally, the function returns value representing the Davies-Bouldin Index which is the average value of db_index across all clusters.



```
def davies_bouldin_index(X, labels, centroids):
    n_clusters = centroids.shape[0]
    db_index = 0
    for i in range(n_clusters):
        for j in range(i + 1, n_clusters):
            cluster1_points = X[labels == i]
            cluster2_points = X[labels == j]
            mean_distance = euclidean_distances(centroids[i].reshape(1, -1), centroids[j].reshape(1, -1))[0][0]
            r1 = np.mean([euclidean_distances(point.reshape(1, -1), centroids[i].reshape(1, -1))[0][0] for point in cluster1_poir
            r2 = np.mean([euclidean_distances(point.reshape(1, -1), centroids[j].reshape(1, -1))[0][0] for point in cluster2_poir
            db_index += (r1 + r2) / mean_distance
    return db_index / n_clusters
```

The dunns_index function calculates the Dunn's Index for a given clustering which is defined as the ratio between the minimum inter-cluster distance and the maximum intra-cluster distance.

The function takes three arguments:

- X is a numpy array of shape (n_samples, n_features) containing the data points,
- labels is a numpy array of shape (n_samples,) containing the cluster labels assigned to each data point
- centroids is a numpy array of shape (n_clusters, n_features) containing the cluster centroids.

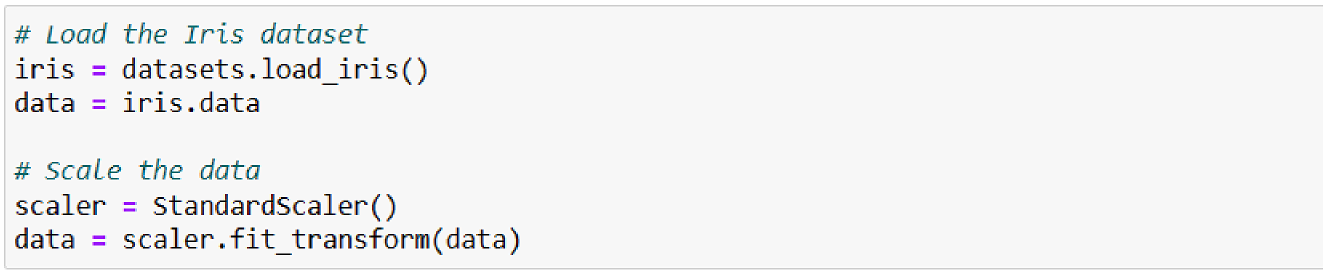
The function first initializes inter_cluster_distances to infinity, and then loops through all pairs of distinct clusters (all pairs of clusters with different indices) to compute the minimum distance between them. This minimum distance is stored in inter_cluster_distance, and the minimum of all such distances is stored in inter_cluster_distances.

Next, the function initializes max_intra_cluster_distance to zero, and then loops through all clusters to compute the maximum distance between any data point in the cluster and the cluster centroid. This maximum distance is stored in intra_cluster_distance, and the maximum of all such distances is stored in max_intra_cluster_distance.

The Dunn's Index is computed as the ratio between inter_cluster_distances and max_intra_cluster_distance. This value is returned as the output of the function.

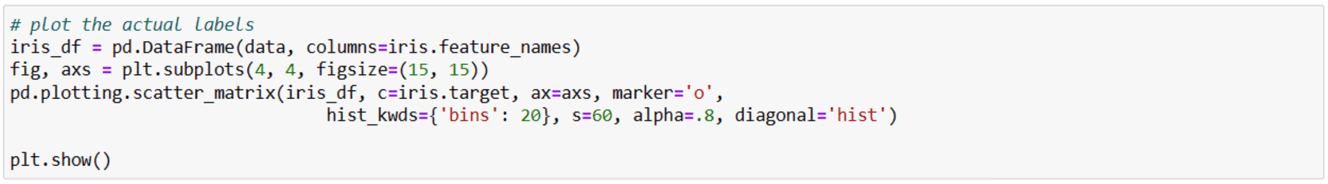
```python
def dunns_index(X, labels, centroids):
    n_clusters = centroids.shape[0]
    inter_cluster_distances = np.inf
    for i in range(n_clusters):
        for j in range(i + 1, n_clusters):
            cluster1_points = X[labels == i]
            cluster2_points = X[labels == j]
            inter_cluster_distance = np.min([euclidean_distances(point1.reshape(1, -1), point2.reshape(1, -1))[0][0]
                                              for point1 in cluster1_points for point2 in cluster2_points])
            inter_cluster_distances = min(inter_cluster_distances, inter_cluster_distance)
    max_intra_cluster_distance = 0
    for i in range(n_clusters):
        cluster_points = X[labels == i]
        intra_cluster_distance = np.max([euclidean_distances(point.reshape(1, -1), centroids[i].reshape(1, -1))[0][0]
                                          for point in cluster_points])
        max_intra_cluster_distance = max(max_intra_cluster_distance, intra_cluster_distance)
    dunns_index = inter_cluster_distances / max_intra_cluster_distance
    return dunns_index
```

The code loads the Iris dataset from the scikit-learn library and stores it in the iris variable. Then, it extracts the features from the dataset and standardizes the feature values by scaling them to have zero mean and unit variance using the StandardScaler class from the sklearn.preprocessing module. The scaled data is then stored in the data variable for further use.



```python
# Load the Iris dataset
iris = datasets.load_iris()
data = iris.data

# Scale the data
scaler = StandardScaler()
data = scaler.fit_transform(data)
```

It also plots the data, with the actual/real clustering labels.



```python
# plot the actual labels
iris_df = pd.DataFrame(data, columns=iris.feature_names)
fig, axs = plt.subplots(4, 4, figsize=(15, 15))
pd.plotting.scatter_matrix(iris_df, c=iris.target, ax=axs, marker='o',
                               hist_kwds={'bins': 20}, s=60, alpha=.8, diagonal='hist')

plt.show()
```

Clustering is performed on the dataset using four different methods after determining the optimal number of clusters to use for each method. The user is asked to input the optimal number of clusters, and the labels and centroids are printed for each method. The elbow method is used for K-Means, K-medoids and Partitioning K-Means clustering, while the BIC is used for GMM clustering to determine the optimal number of clusters.

The SSE list stores the SSE values for each number of clusters. The for loop iterates over 1 to 10 clusters and initializes a KMeans object with the corresponding number of clusters, random state, and other parameters. The fit() method of the KMeans object is called on the data and the SSE value is appended to the SSE list.

After computing the SSE for each number of clusters, the code generates a plot of SSE against number of clusters using the matplotlib library. The user is asked to input the optimal number of clusters, which is then used to perform K-Means clustering on the data again. The resulting labels and centroids are stored in kmeans_labels and centroids_kmeans respectively. The same is done for each one of the 4 algorithms.

```python
#Find best k and perform clustering

# K-Means Clustering-------------------------------------------------------------------------------------------
SSE = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init ='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(data)
    SSE.append(kmeans.inertia_)

plt.plot(range(1,11), SSE)
plt.title("Elbow Method for K-Means Clustering")
plt.xlabel("Number of clusters")
plt.ylabel("SSE")
plt.show()

best_k = input("Enter the best number of clusters for K-Means: ")
kmeans = KMeans(n_clusters=int(best_k), random_state=0)
kmeans.fit(data)
kmeans_labels = kmeans.labels_
centroids_kmeans = kmeans.cluster_centers_
print()

# partitioning K-Means Clustering----------------------------------------------------------------------------
SSE = []
for i in range(1,11):
    pkmeans = PartitioningKMeans(n_clusters = i, random_state=0)
    pkmeans.fit(data)
    SSE.append(PartitioningKMeans._inertia(data, pkmeans.predict(data), pkmeans.cluster_centers_))

plt.plot(range(1,11), SSE)
plt.title("Elbow Method for Partitioning KMeans Clustering")
plt.xlabel("Number of clusters")
plt.ylabel("SSE")
plt.show()

best_k = input("Enter the best number of clusters for Partitioning K-Means: ")
pkmeans = PartitioningKMeans(n_clusters=int(best_k), random_state=0)
pkmeans.fit(data)
pkmeans_labels = pkmeans.predict(data)
centroids_pkmeans = pkmeans.cluster_centers_
print()

# K-Medoids Clustering-----------------------------------------------------------------------------------------
SSE = []
for i in range(1,11):
    kmedoids = KMedoids(n_clusters = i, random_state=0)
    kmedoids.fit(data)
    SSE.append(kmedoids.inertia_)

plt.plot(range(1,11), SSE)
plt.title("Elbow Method for K-Medoids Clustering")
plt.xlabel("Number of clusters")
plt.ylabel("SSE")
plt.show()

best_k = input("Enter the best number of clusters for K-Medoids: ")
kmedoids = KMedoids(n_clusters=int(best_k), random_state=0)
kmedoids.fit(data)
kmedoids_labels = kmedoids.labels_
centroids_kmedoids = kmedoids.cluster_centers_

# GMM Clustering-----------------------------------------------------------------------------------------------
bic = []
for i in range(1,11):
    gmm = GaussianMixture(n_components=i, random_state=0)
    gmm.fit(data)
    bic.append(gmm.bic(data))

plt.plot(range(1,11), bic)
plt.title("Elbow Method for GMM Clustering")
plt.xlabel("Number of clusters")
plt.ylabel("BIC")
plt.show()

best_g = input("Enter the best number of clusters for GMM: ")
gmm = GaussianMixture(n_components=int(best_g), random_state=0)
gmm.fit(data)
gmm_labels = gmm.predict(data)
centroids_g = gmm.means_
```

The code is plotting the results of the clustering algorithms applied to the Iris dataset. It calls the plot_clustering_results function for each algorithm, passing the algorithm name, the data, the cluster labels and the centroids. n_attributes is the number of attributes (columns) in the dataset, which is 4 in this case (sepal length, sepal width, petal length, petal width).

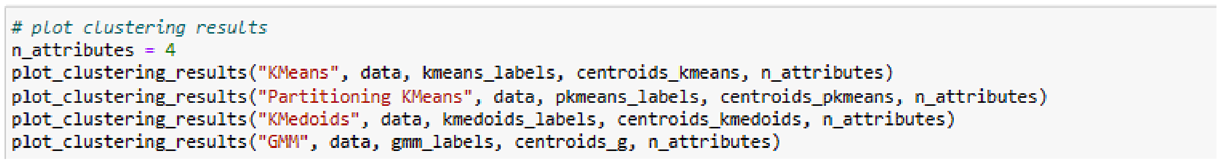
The plot_clustering_results function creates a scatter plot matrix of the data, with each pair of attributes represented in a scatter plot. The points are colored according to their cluster labels, and the centroids are marked with red crosses. The function takes five arguments:

- algorithm: a string with the name of the clustering algorithm
- data: a numpy array with the data points
- labels: a numpy array with the cluster labels for each point
- centroids: a numpy array with the centroid coordinates for each cluster
- n_attributes: an integer with the number of attributes in the dataset

The function first creates a pandas DataFrame from the data array, with the attribute names as column names. It then creates a scatter plot matrix using the scatter_matrix function from pandas, with the c argument set to the cluster labels, and the diagonal argument set to 'hist', which makes the diagonal plots histograms of each attribute.

The function then creates a pandas DataFrame from the centroid array, with the attribute names as column names. It loops over the subplots in the scatter plot matrix, and for each subplot, it checks if it is on the diagonal (where we have a histogram subplot) or not. If it is on the diagonal, it skips plotting the centroids. If it is not on the diagonal, it plots the centroids as red crosses at the centroid coordinates for each cluster.

Finally, the function adds a title to the plot with the name of the clustering algorithm, and shows the plot using the show function from matplotlib.pyplot.



```
# plot clustering results
n_attributes = 4
plot_clustering_results("KMeans", data, kmeans_labels, centroids_kmeans, n_attributes)
plot_clustering_results("Partitioning KMeans", data, pkmeans_labels, centroids_pkmeans, n_attributes)
plot_clustering_results("KMedoids", data, kmedoids_labels, centroids_kmedoids, n_attributes)
plot_clustering_results("GMM", data, gmm_labels, centroids_g, n_attributes)
```

Each block calculates the different metrics for each one of the four algorithms.

The first three lines of code calculate the Adjusted Rand Index, Adjusted Mutual Information, Normalized Mutual Information. These metrics are used to evaluate the quality of the clustering results by comparing the clusters obtained by the algorithm with the actual labels (in this case, the actual class labels of the iris dataset - iris.target).

The next lines of code calculate the internal evaluation metrics. Note that in order to calculate this metrics we do not require the actual labels (iris.target) as an input. The fourth line calculates the Silhouette Score, the fifth line calculates the Calinski-Harabasz Index, the sixth line calculates the Davies-Bouldin Index and the last line of code calculates the Dunn Index (for the last two metrics the centroids are used as input arguments)

```
#Calculate the performance metrics

#K-means performance
ari_kmeans = metrics.adjusted_rand_score(iris.target, kmeans_labels)
ami_kmeans = metrics.adjusted_mutual_info_score(iris.target, kmeans_labels)
nmi_kmeans = metrics.normalized_mutual_info_score(iris.target, kmeans_labels)
silhouette_kmeans = metrics.silhouette_score(data, kmeans_labels)
chi_kmeans = metrics.calinski_harabasz_score(data, kmeans_labels)
dbi_kmeans = davies_bouldin_index(data, kmeans_labels,centroids_kmeans)
di_kmeans = dunns_index(data, kmeans_labels,centroids_kmeans)

#Partitioning K-means performance
ari_pkmeans = metrics.adjusted_rand_score(iris.target, pkmeans_labels)
ami_pkmeans = metrics.adjusted_mutual_info_score(iris.target, pkmeans_labels)
nmi_pkmeans = metrics.normalized_mutual_info_score(iris.target, pkmeans_labels)
silhouette_pkmeans = metrics.silhouette_score(data, pkmeans_labels)
chi_pkmeans = metrics.calinski_harabasz_score(data, pkmeans_labels)
dbi_pkmeans = davies_bouldin_index(data, pkmeans_labels,centroids_pkmeans)
di_pkmeans = dunns_index(data, pkmeans_labels,centroids_pkmeans)

#K-medoids performance
ari_kmedoids = metrics.adjusted_rand_score(iris.target, kmedoids_labels)
ami_kmedoids = metrics.adjusted_mutual_info_score(iris.target, kmedoids_labels)
nmi_kmedoids = metrics.normalized_mutual_info_score(iris.target, kmedoids_labels)
silhouette_kmedoids = metrics.silhouette_score(data, kmedoids_labels)
chi_kmedoids = metrics.calinski_harabasz_score(data, kmedoids_labels)
dbi_kmedoids = davies_bouldin_index(data, kmedoids_labels,centroids_kmedoids)
di_kmedoids = dunns_index(data, kmedoids_labels,centroids_kmedoids)

#GMM performance
ari_gmm = metrics.adjusted_rand_score(iris.target, gmm_labels)
ami_gmm = metrics.adjusted_mutual_info_score(iris.target, gmm_labels)
nmi_gmm = metrics.normalized_mutual_info_score(iris.target, gmm_labels)
silhouette_gmm = metrics.silhouette_score(data, gmm_labels)
chi_gmm = metrics.calinski_harabasz_score(data, gmm_labels)
dbi_gmm = davies_bouldin_index(data, gmm_labels,centroids_g)
di_gmm = dunns_index(data, gmm_labels,centroids_g)
```

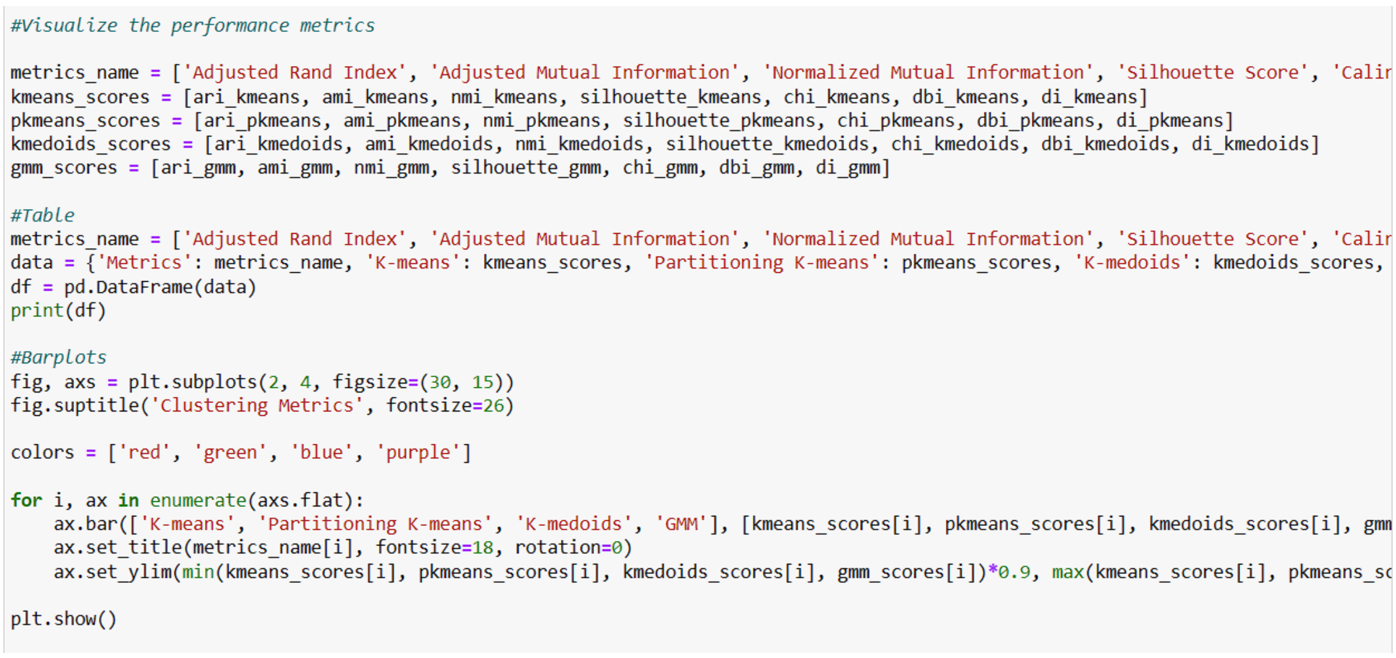
The provided code calculates the coordinates of the real centroids based on the true labels of the Iris dataset. The code uses the NumPy library to calculate the mean of each feature for each cluster. It then stores the coordinates of the centroids in a list with the name centroids. After calculating the centroids, the code calculates four internal performance metrics for the actual labels: Silhouette score, Calinski-Harabasz score, Davies-Bouldin index, and Dunn's index.

```
# Calculate the coordinates of the real centroids based on the true labels
centroids = []
for i in range(3):
    mask = (iris.target == i)
    centroid = np.mean(data[mask], axis=0)
    centroids.append(centroid)

# Print the coordinates of the real centroids
print("Coordinates of the real centroids:")
for i in range(3):
    print("Cluster {}: {}".format(i, centroids[i]))

centroids_actual = np.array(centroids)

#actual labels performance metrics
silhouette_actual = metrics.silhouette_score(data, iris.target)
chi_actual = metrics.calinski_harabasz_score(data, iris.target)
dbi_actual = davies_bouldin_index(data, iris.target,centroids_actual)
di_actual = dunns_index(data, iris.target,centroids_actual)
print(silhouette_actual,chi_actual,dbi_actual,di_actual)
```

The first block of code defines the names of the metrics and stores the scores obtained from each algorithm for each metric in separate lists.

The second part of the code creates a table using the pandas library, which displays the scores of each metric for each algorithm.

The third part of the code creates bar plots using the matplotlib library. There are 7 subplots, with each subplot corresponding to one metric. The x-axis of each subplot represents the different algorithms, and the y-axis represents the scores obtained for each metric. The color of each bar represents a different algorithm. The title of each subplot denotes the metric being displayed.



```
#Visualize the performance metrics

metrics_name = ['Adjusted Rand Index', 'Adjusted Mutual Information', 'Normalized Mutual Information', 'Silhouette Score', 'Calir
kmeans_scores = [ari_kmeans, ami_kmeans, nmi_kmeans, silhouette_kmeans, chi_kmeans, dbi_kmeans, di_kmeans]
pkmeans_scores = [ari_pkmeans, ami_pkmeans, nmi_pkmeans, silhouette_pkmeans, chi_pkmeans, dbi_pkmeans, di_pkmeans]
kmedoids_scores = [ari_kmedoids, ami_kmedoids, nmi_kmedoids, silhouette_kmedoids, chi_kmedoids, dbi_kmedoids, di_kmedoids]
gmm_scores = [ari_gmm, ami_gmm, nmi_gmm, silhouette_gmm, chi_gmm, dbi_gmm, di_gmm]

#Table
metrics_name = ['Adjusted Rand Index', 'Adjusted Mutual Information', 'Normalized Mutual Information', 'Silhouette Score', 'Calir
data = {'Metrics': metrics_name, 'K-means': kmeans_scores, 'Partitioning K-means': pkmeans_scores, 'K-medoids': kmedoids_scores,
df = pd.DataFrame(data)
print(df)

#Barplots
fig, axs = plt.subplots(2, 4, figsize=(30, 15))
fig.suptitle('Clustering Metrics', fontsize=26)

colors = ['red', 'green', 'blue', 'purple']

for i, ax in enumerate(axs.flat):
    ax.bar(['K-means', 'Partitioning K-means', 'K-medoids', 'GMM'], [kmeans_scores[i], pkmeans_scores[i], kmedoids_scores[i], gmm
    ax.set_title(metrics_name[i], fontsize=18, rotation=0)
    ax.set_ylim(min(kmeans_scores[i], pkmeans_scores[i], kmedoids_scores[i], gmm_scores[i])*0.9, max(kmeans_scores[i], pkmeans_sc

plt.show()
```

Overall, this script is performing clustering on the Iris dataset using K-Means, K-medoids, Partitioning K-Means and Gaussian Mixture Model (GMM) algorithms.

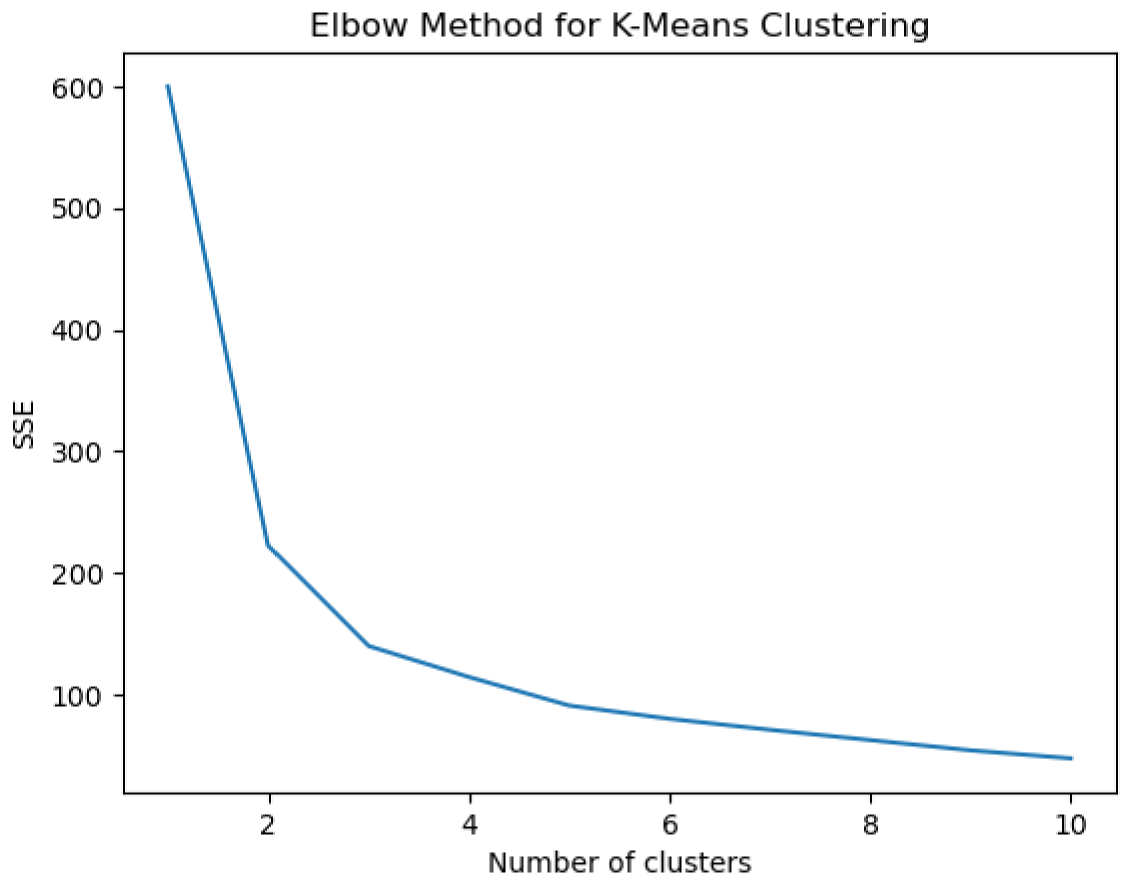
It first standardizes the data by scaling it. Then it uses the Elbow Method to find the optimal number of clusters for each algorithm. The script plots the SSE and BIC values for different number of clusters to help the user decide the best number of clusters.

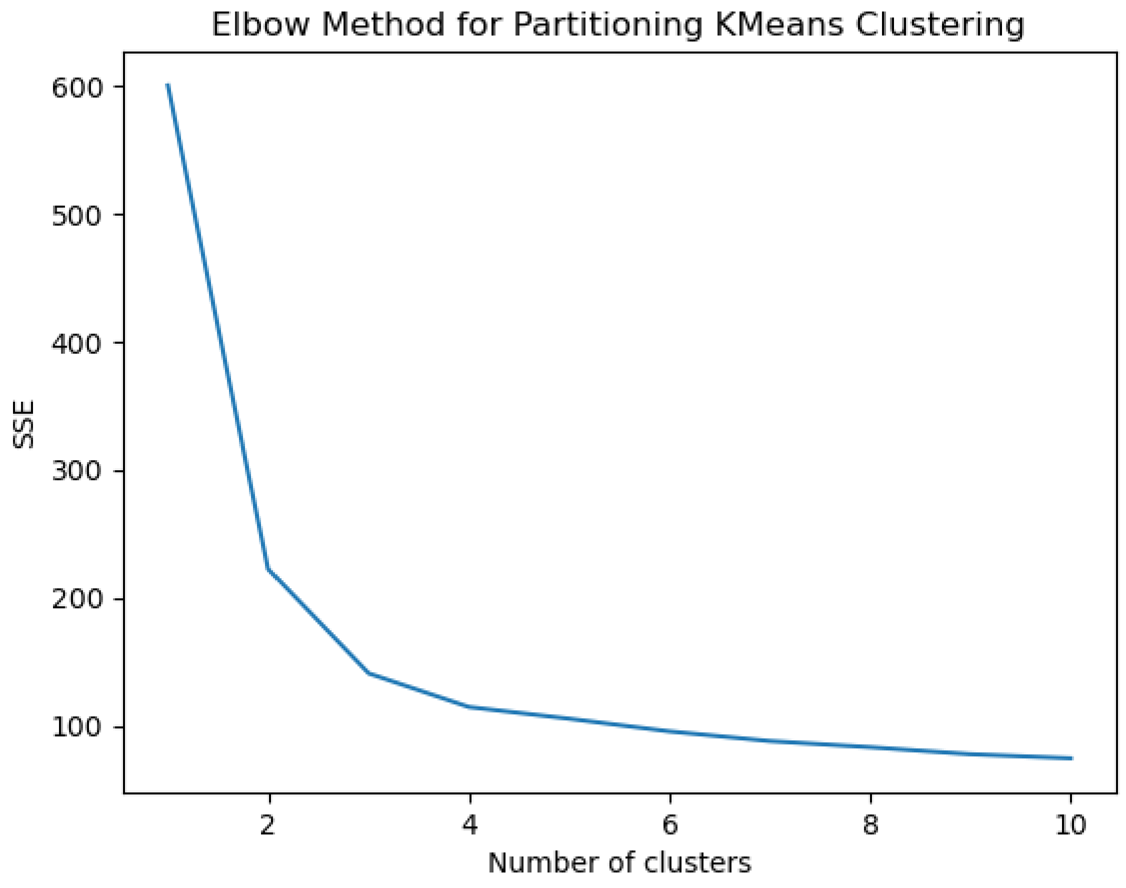
After finding the optimal number of clusters, the script fits all the algorithms on the data and plots the results. It also plots a scatter matrix to visualize the results.

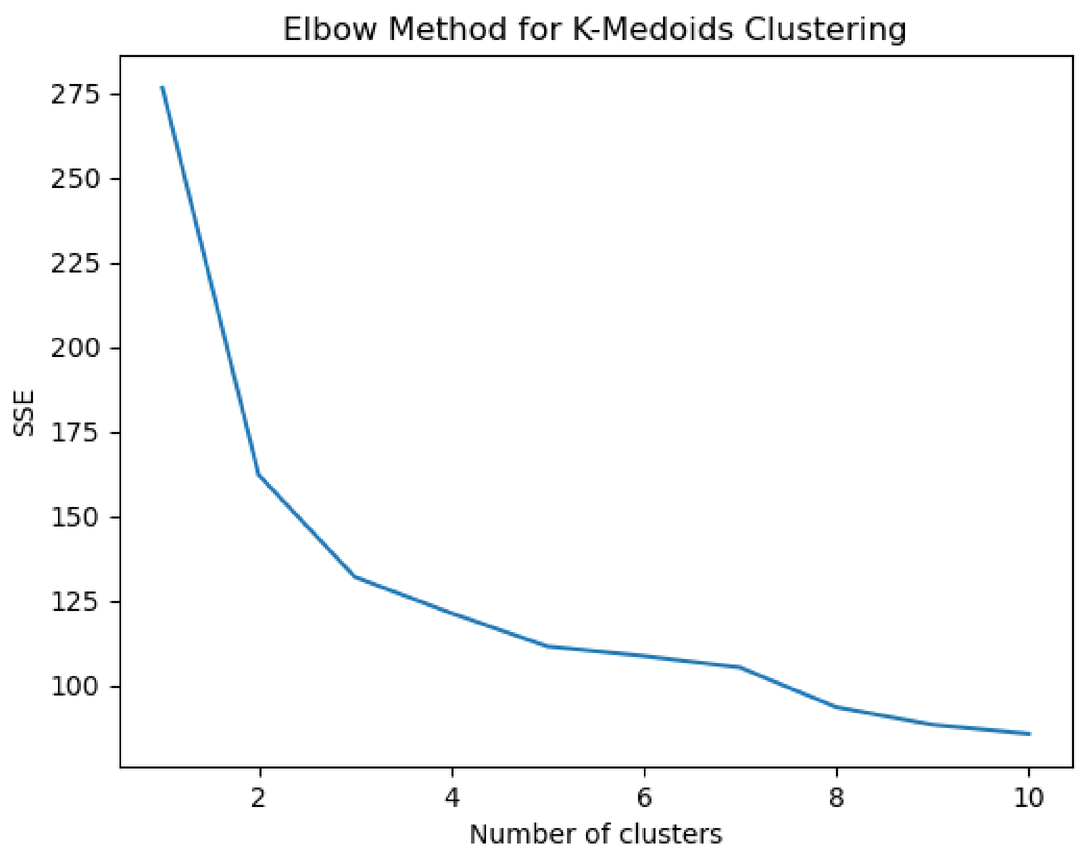
Finally, it calculates various performance metrics like adjusted rand index, adjusted mutual information, silhouette score, etc. for each algorithm and displays the values in a table format and also in bar plots.

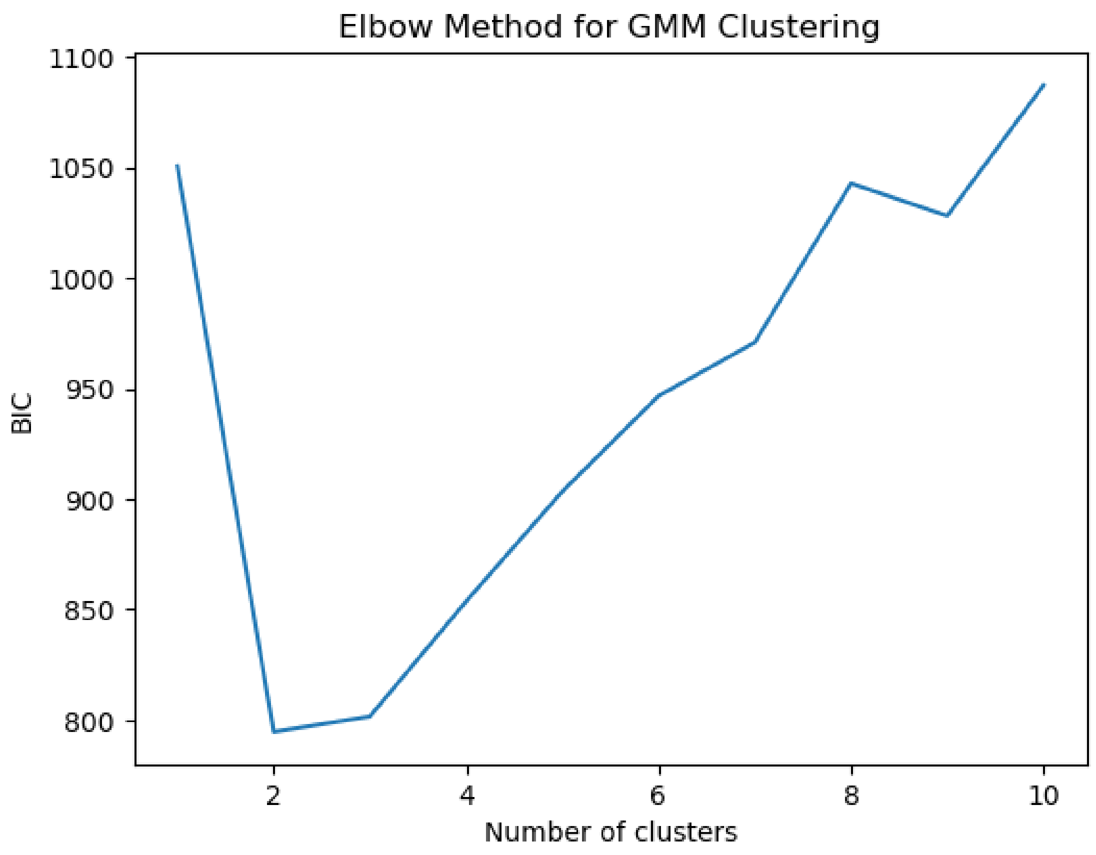
Results

Elbow method results





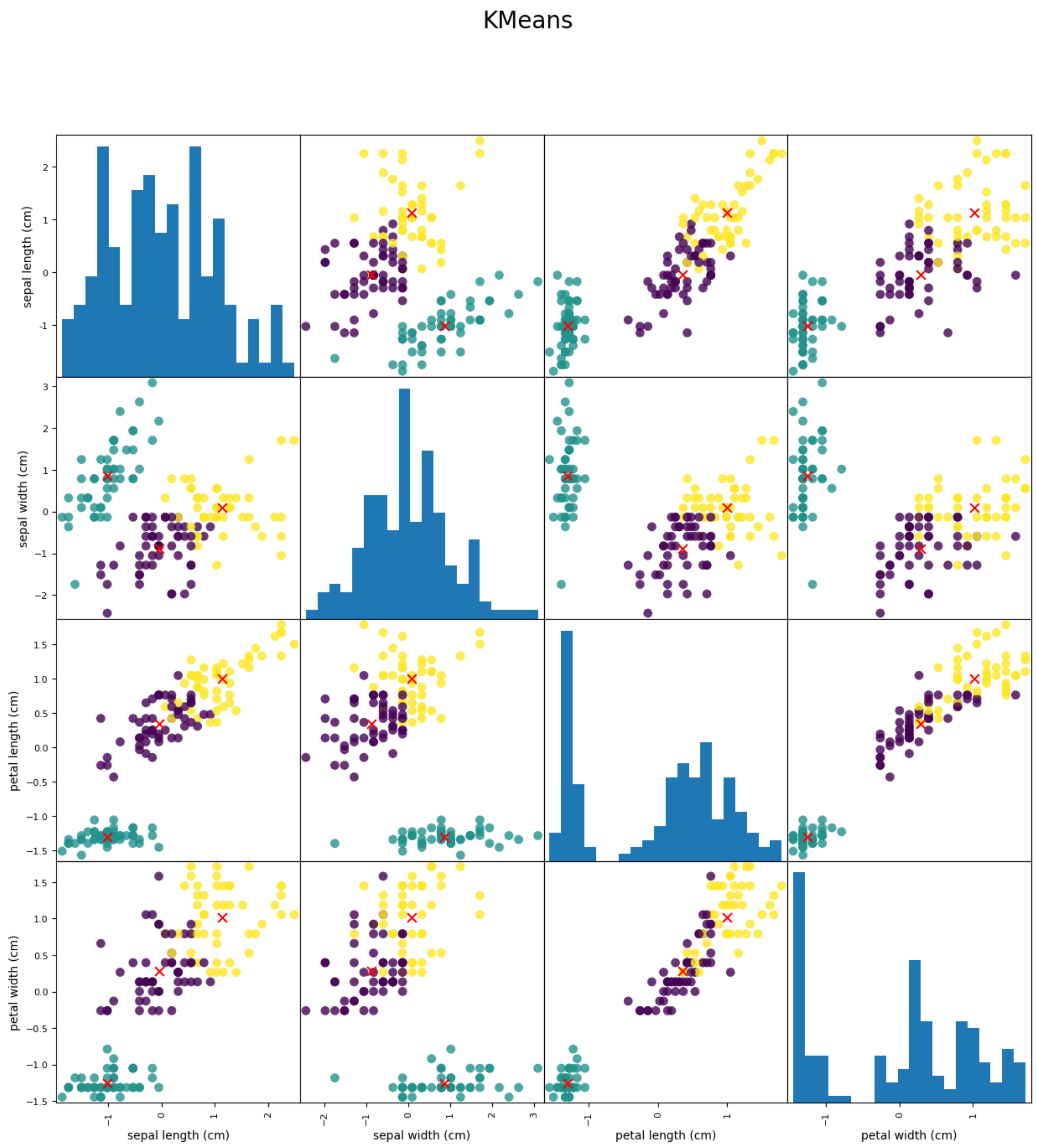




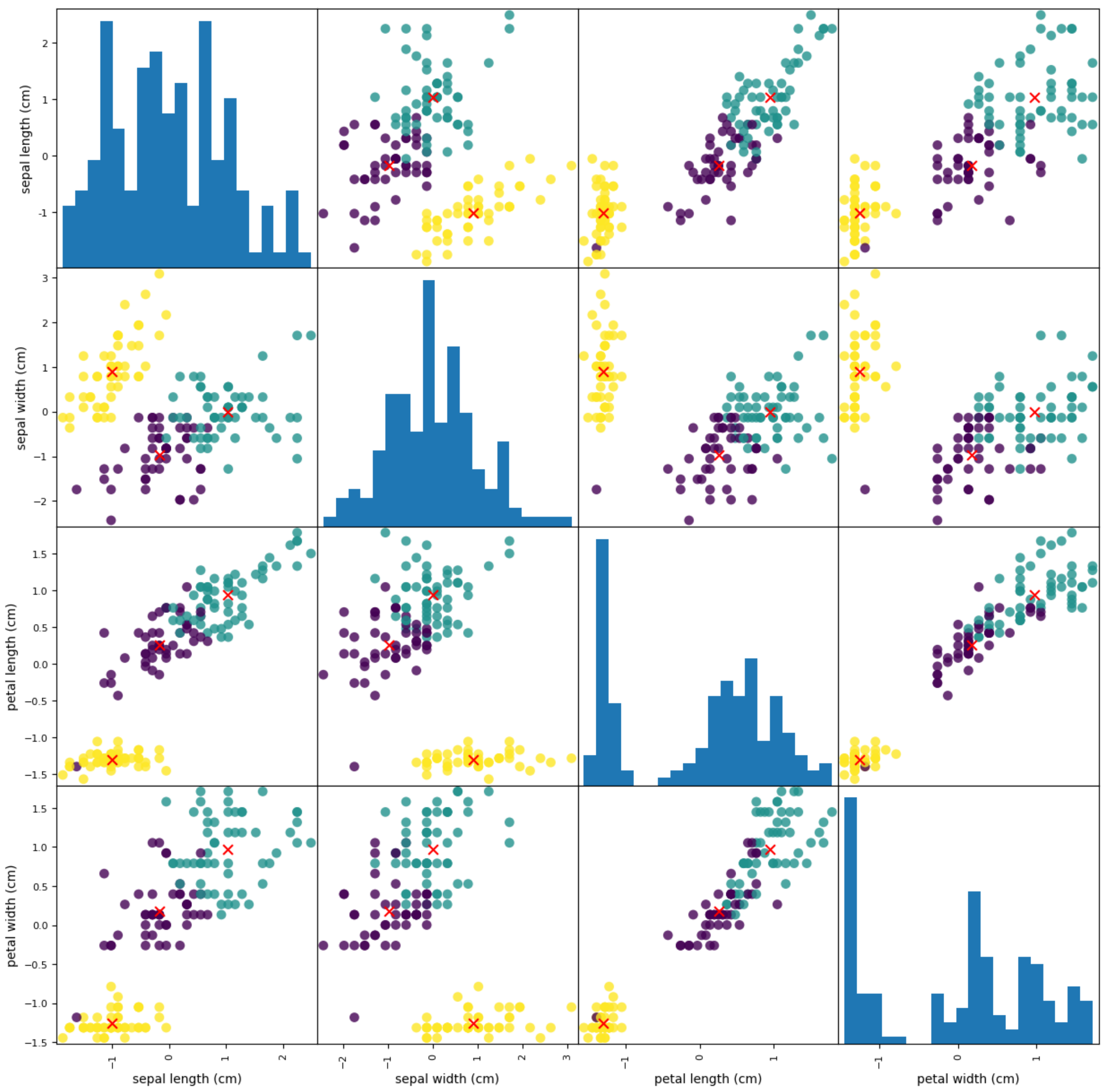
The user will have to choose and optimal number of clusters for each algorithm. The best k may be different for each algorithm. K=3 seems to be an optimal solution for this application.

Clustering results

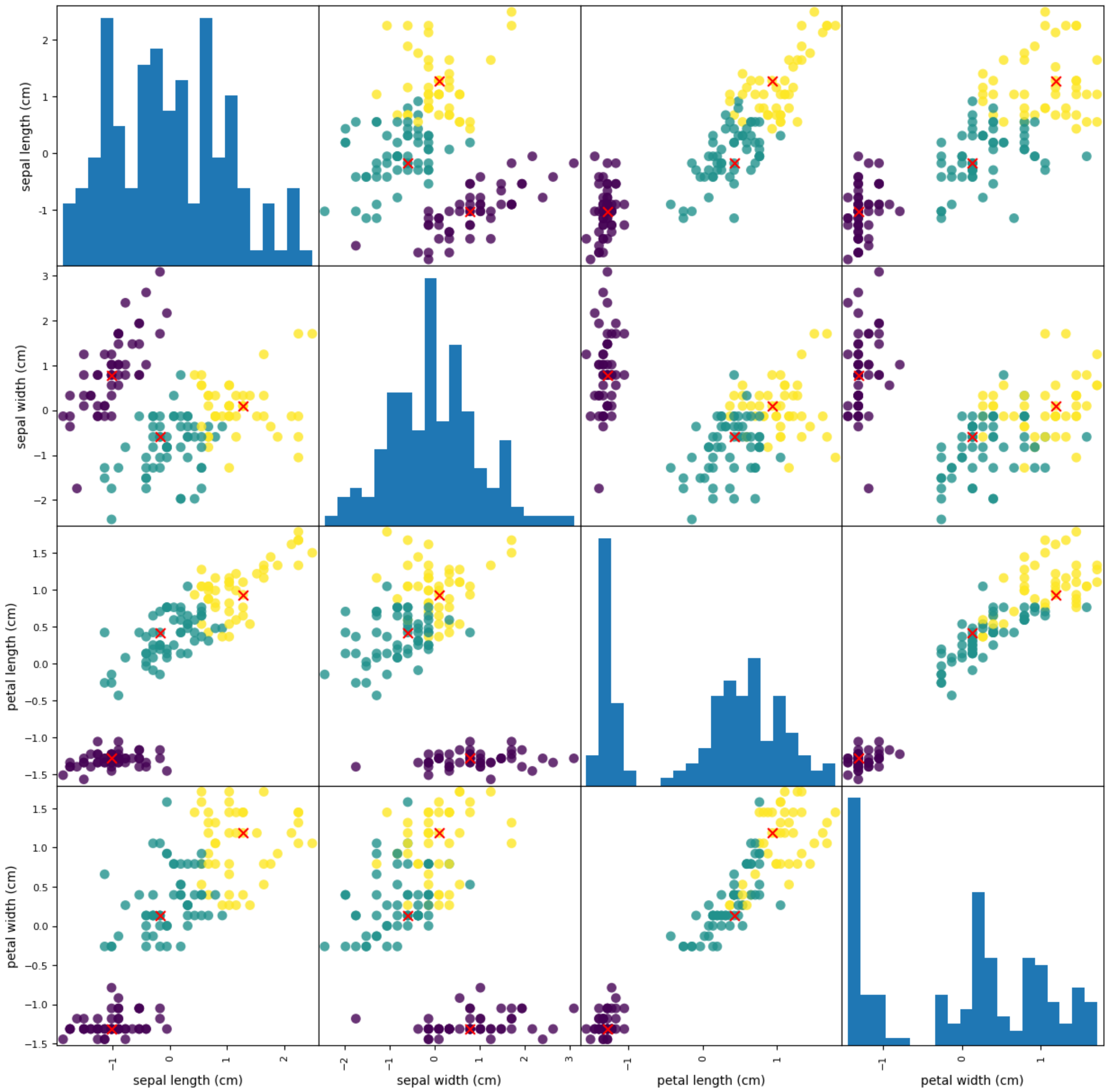
Below, the clustering results for k = 3 (number of clusters) are presented. The color of each data point is set by its assigned cluster label. The centroids are also plotted on the scatter matrix plot. For each subplot, a red 'x' is plotted at the position of each centroid. The diagonal plots are histograms of each attribute.



Partitioning KMeans

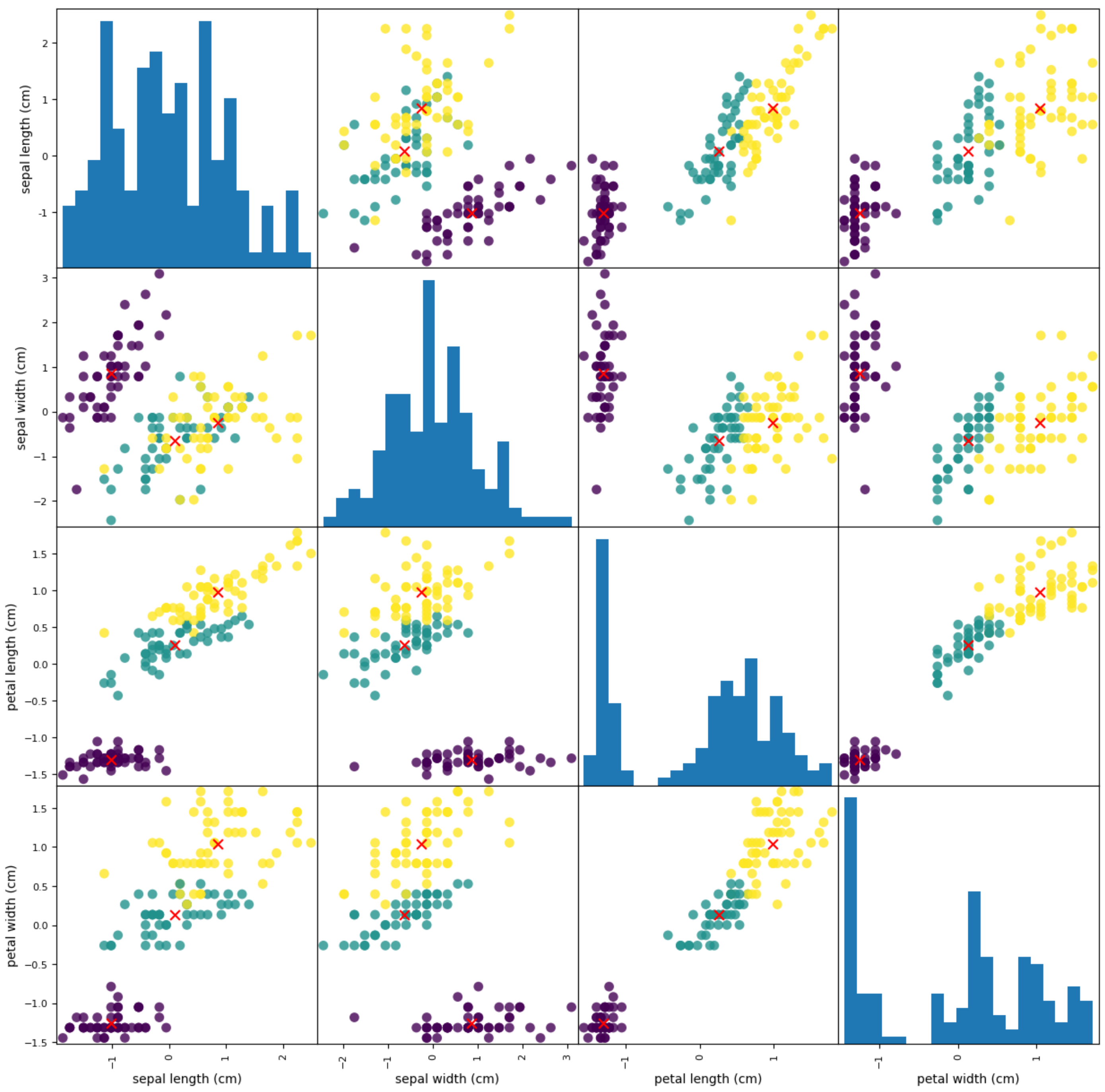


KMedoids



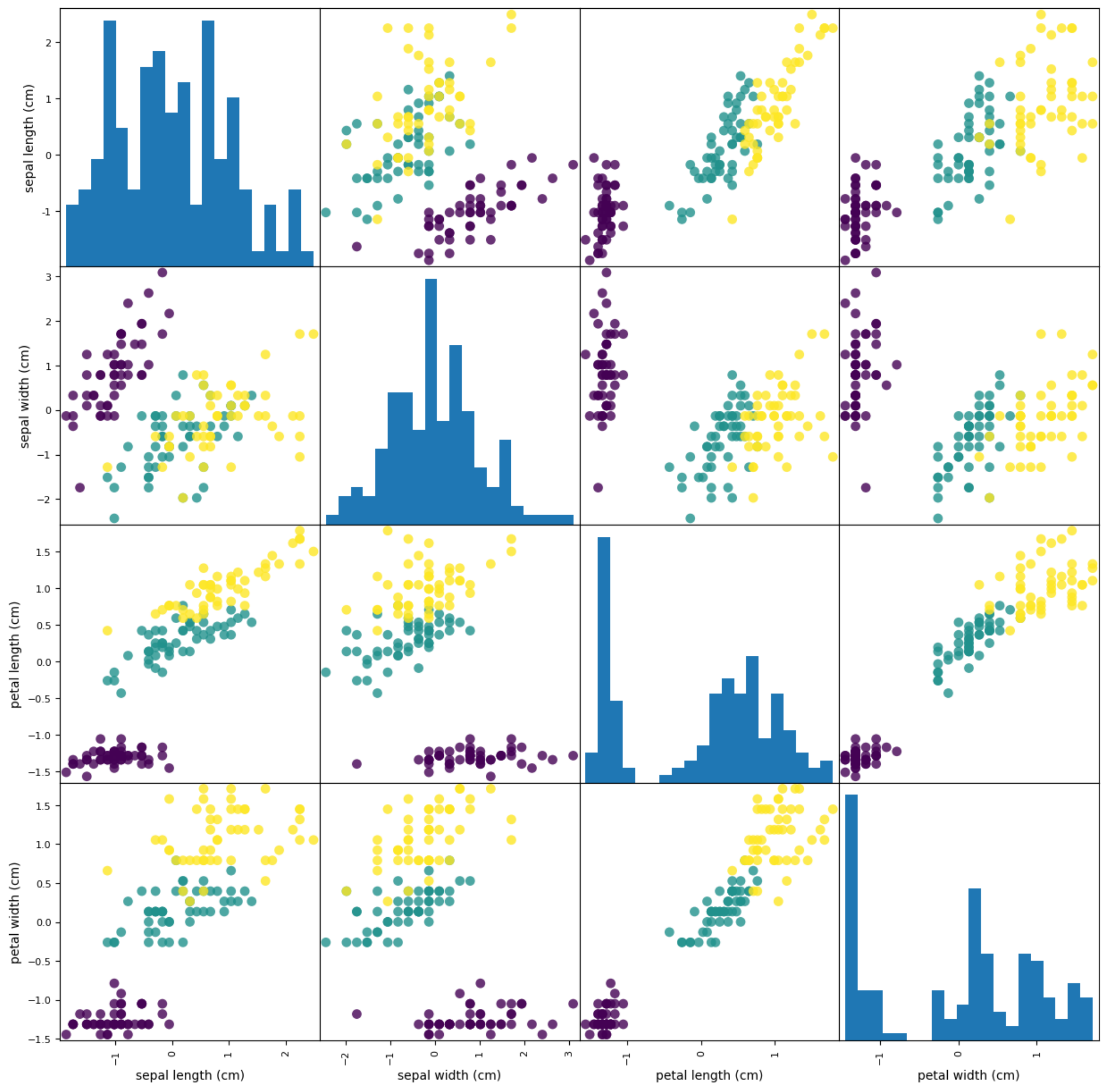
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

GMM



sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

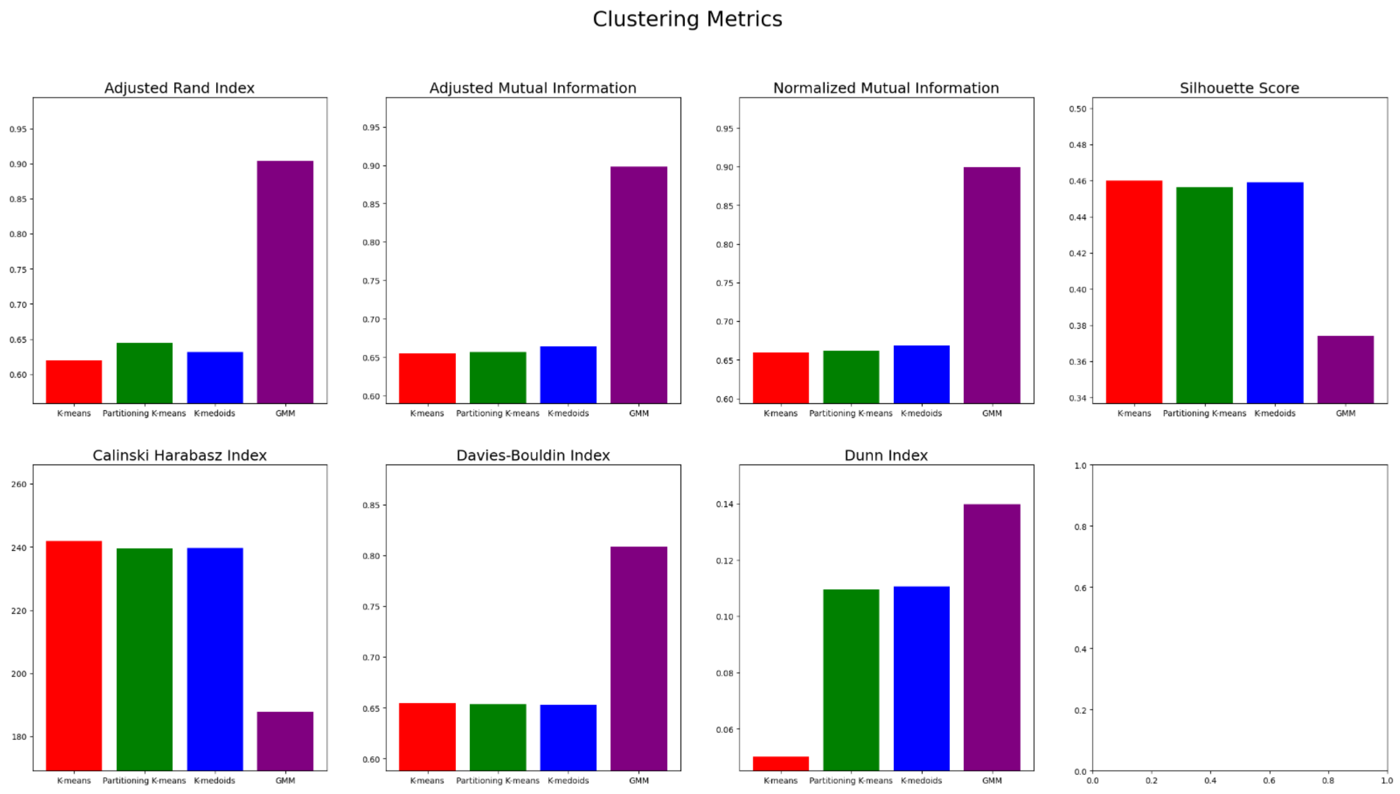
True labels



In most unsupervised clustering tasks, the actual class labels of the data points would not be available. In this case the actual labels are available so we can make use of this information to investigate further the clustering performance of the algorithms compared to the real results.

Conclusion - Clustering metrics results and commentary

The results for each algorithm are presented with a unique color. Red is k-means, Green is Partitioning k-means, Blue is k-medoids and Purple is GMM. The results of each clustering metric are visualized in a different bar plot. There are 7 bar plots in total, each for a different clustering performance metric.

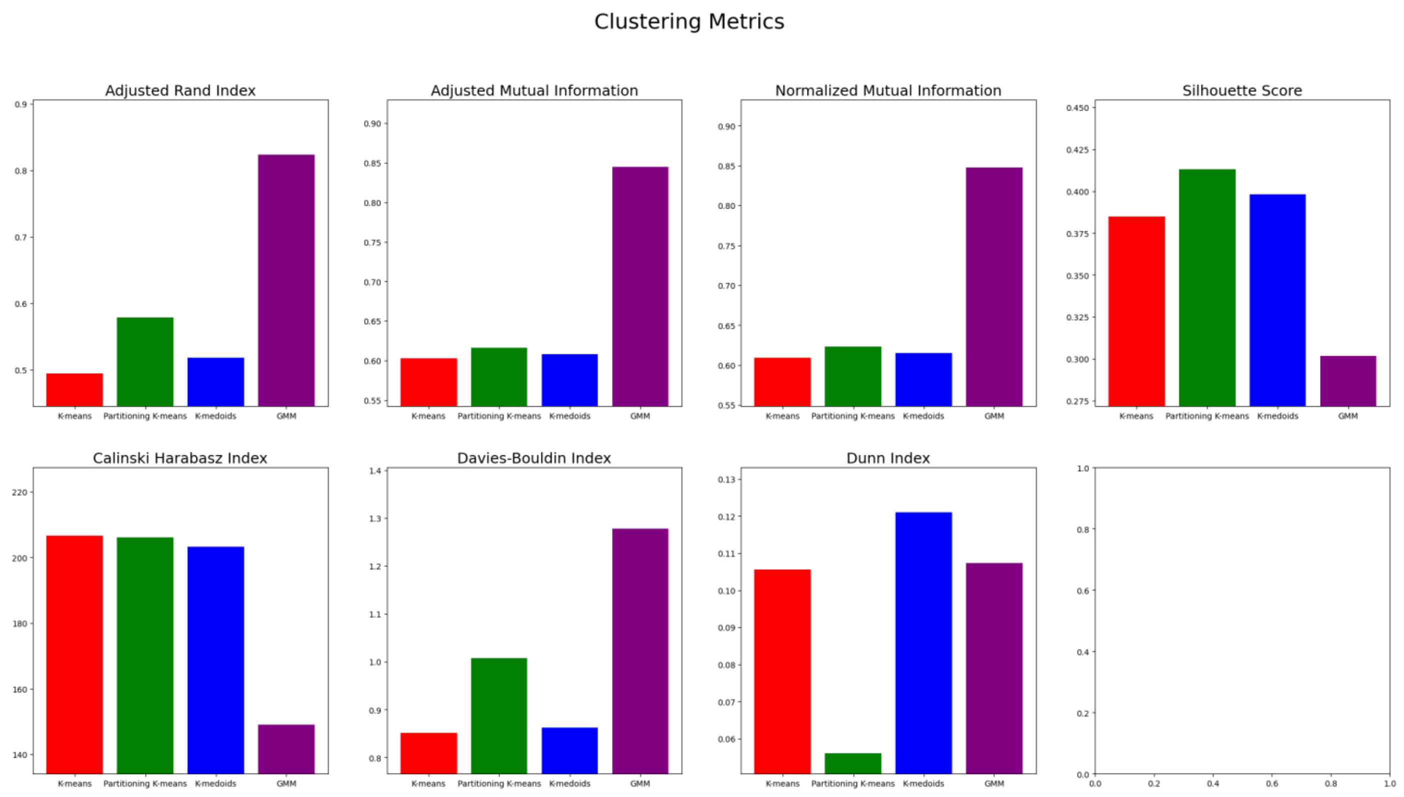


Bar plots of clustering performance metrics results for k=3

	K-means	Partitioning K-means	K-medoids	GMM
Adjusted Rand Index	0.620	0.645	0.631	0.904
Adjusted Mutual Information	0.655	0.657	0.665	0.898
Normalized Mutual Information	0.659	0.661	0.669	0.900
Silhouette Score	0.460	0.457	0.459	0.374
Calinski-Harabasz Index	241.904	239.485	239.748	187.774
Davies-Bouldin Index	0.655	0.654	0.653	0.809
Dunn Index	0.050	0.110	0.111	0.140

Table of clustering performance metrics results for k=3

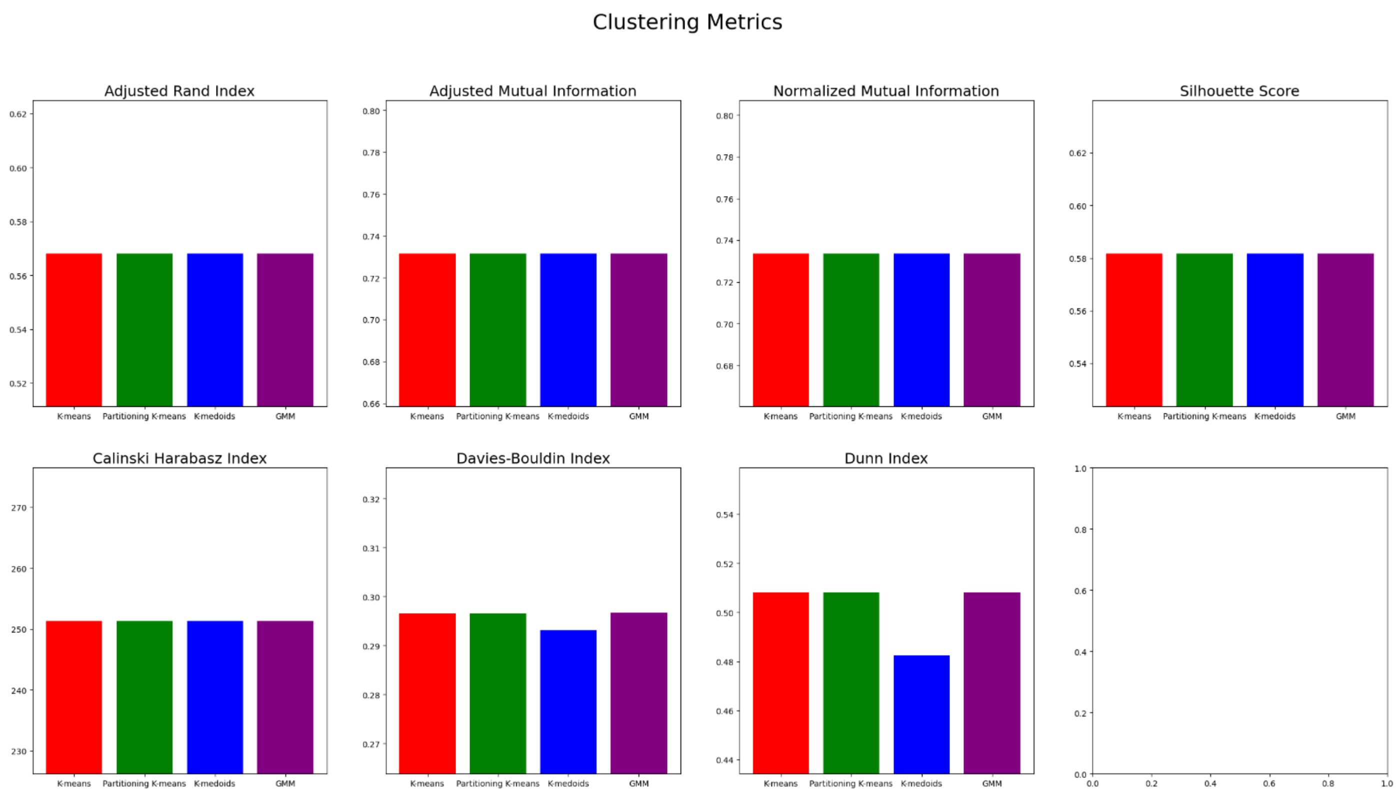
The clustering metrics results will be calculated for k=2 and k=4 as well, in order to identify how their values change for varying k numbers.



Bar plots of clustering performance metrics results for k=4

	K-means	Partitioning K-means	K-medoids	GMM
Adjusted Rand Index	0.495	0.579	0.518	0.824
Adjusted Mutual Information	0.603	0.616	0.608	0.845
Normalized Mutual Information	0.609	0.623	0.615	0.847
Silhouette Score	0.385	0.413	0.398	0.302
Calinski-Harabasz Index	206.681	206.073	203.291	149.032
Davies-Bouldin Index	0.851	1.008	0.863	1.278
Dunn Index	0.106	0.056	0.121	0.107

Table of clustering performance metrics results for k=4

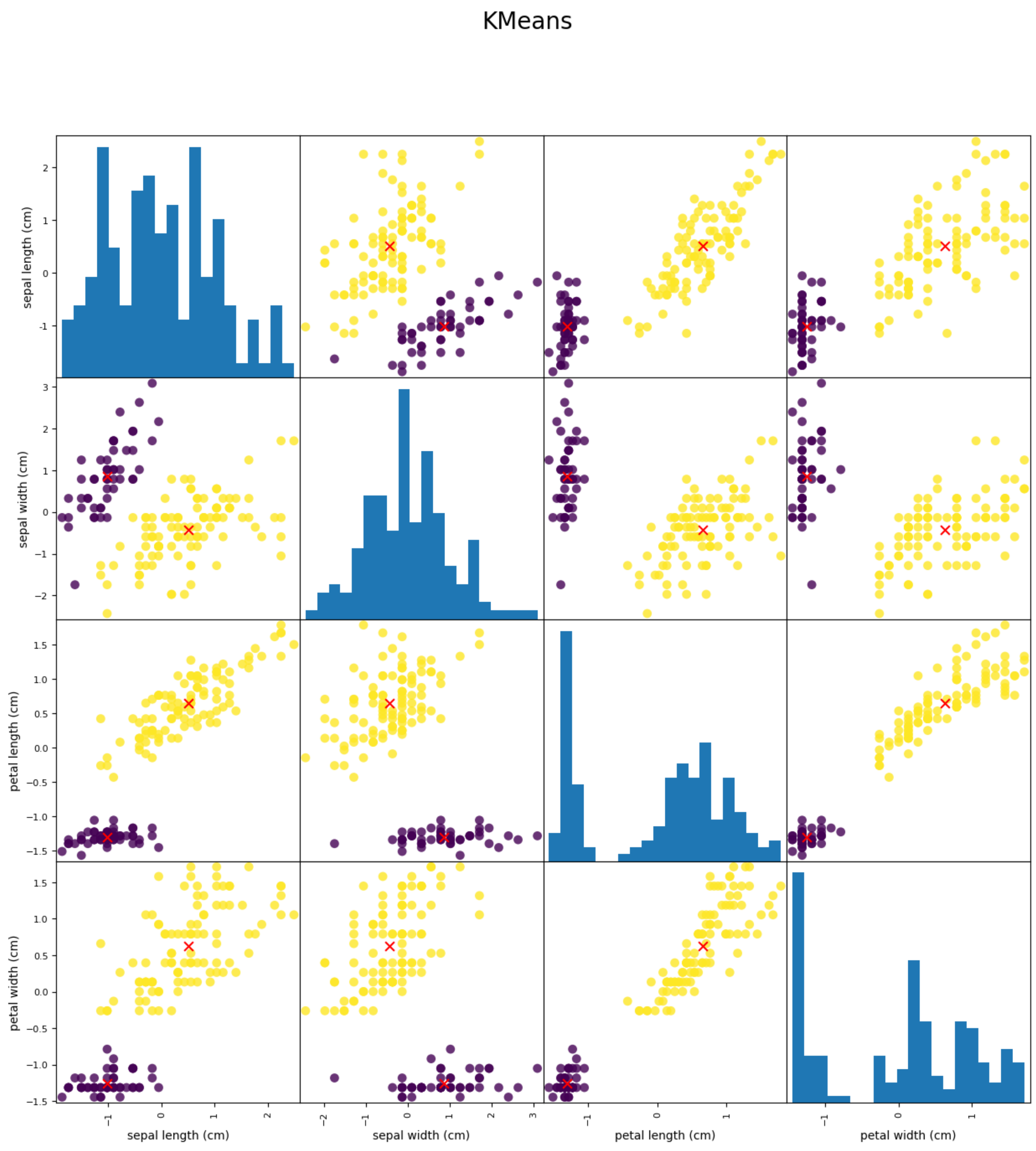


Bar plots of clustering performance metrics results for k=2

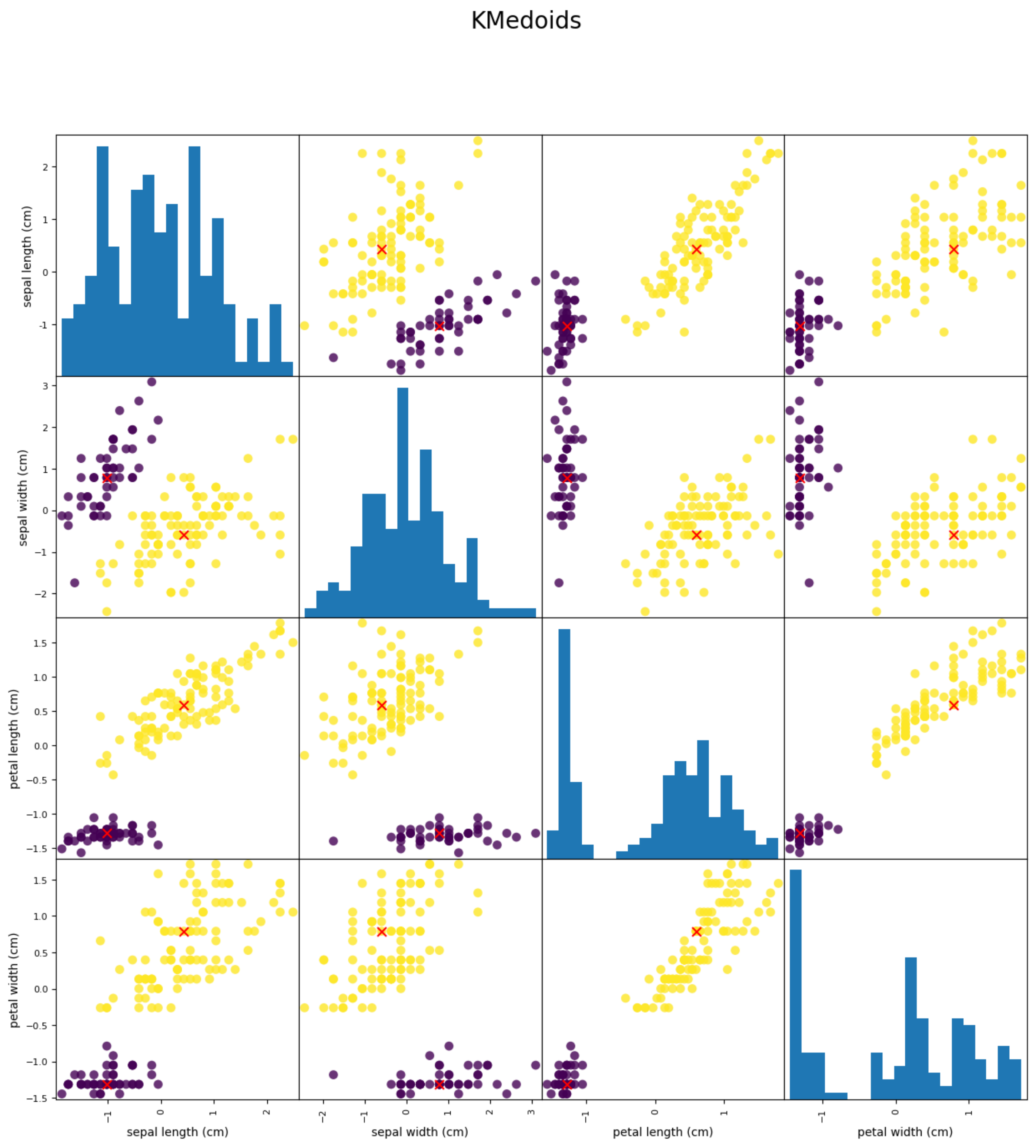
	K-means	Partitioning K-means	K-medoids	GMM
Adjusted Rand Index	0.568	0.568	0.568	0.568
Adjusted Mutual Information	0.732	0.732	0.732	0.732
Normalized Mutual Information	0.734	0.734	0.734	0.734
Silhouette Score	0.582	0.582	0.582	0.582
Calinski-Harabasz Index	251.349	251.349	251.349	251.349
Davies-Bouldin Index	0.297	0.297	0.293	0.297
Dunn Index	0.508	0.508	0.482	0.508

Table of clustering performance metrics results for k=2

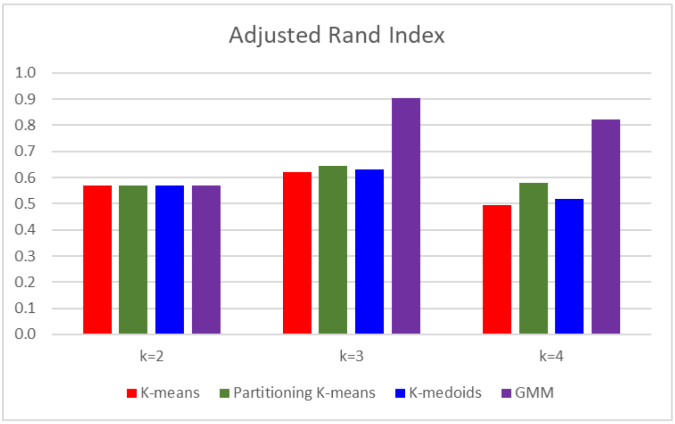
All algorithms give nearly the same clustering solution (class labels assigned to each point) for k=2. This explains why the values of almost all the metrics are identical.

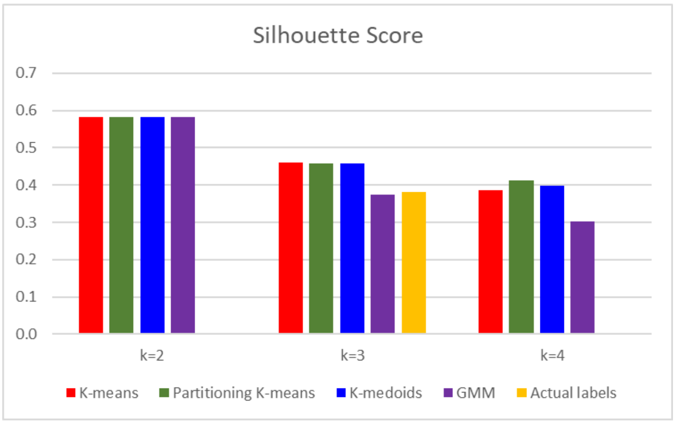


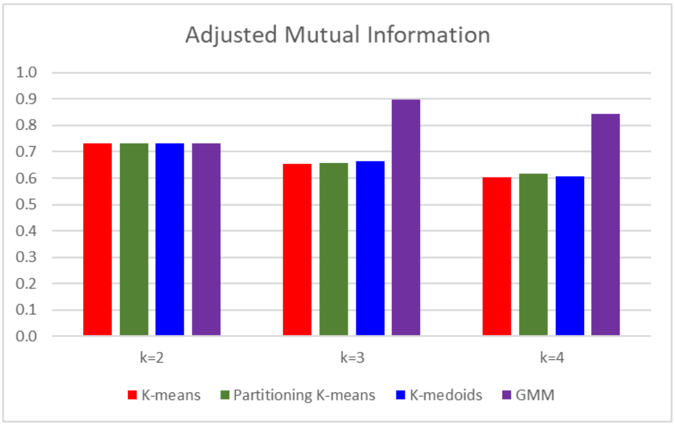
We can visually see that the clustering result for k=2 is very different from the actual cluster labels. That is why the ARI is lower that the k=3 case of clustering.

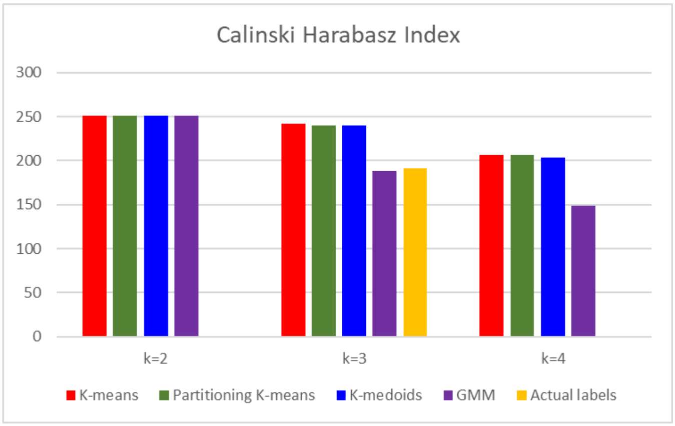


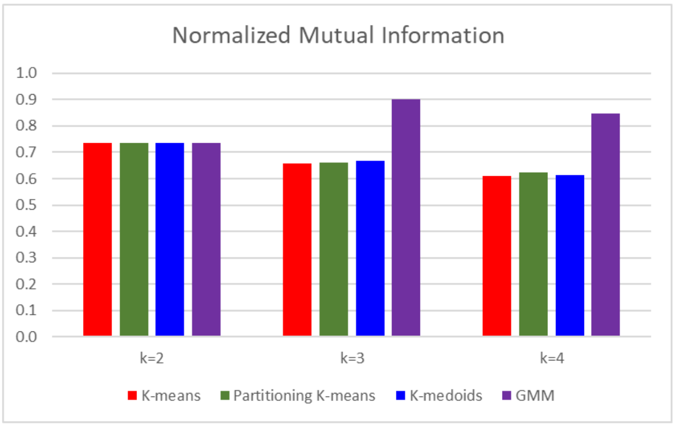
Difference in DBI and DI is due to different centroid positions. K-medoids sets real datapoints as centroids whereas the other algorithms do not. Nevertheless, the output labels for the datapoints are the same for all the algorithms.

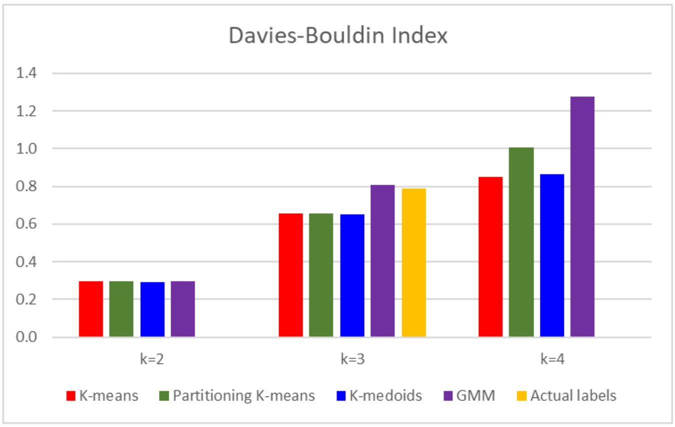


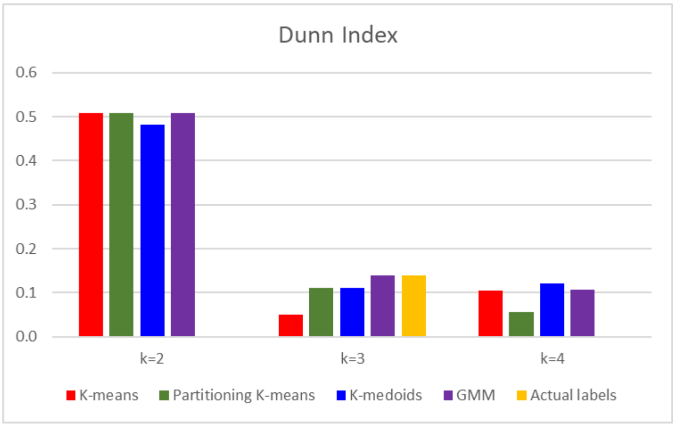












Bar plots of clustering performance metrics results for k=2,3,4. The yellow bar represents the internal clustering metrics results for the actual labels of the iris dataset.

	K-means			Partitioning K-means			K-medoids			GMM		
	k=2	**k=3**	**k=4**	**k=2**	**k=3**	**k=4**	**k=2**	**k=3**	**k=4**	**k=2**	**k=3**	**k=4**
Adjusted Rand Index	0.568	0.62	0.495	0.568	0.645	0.579	0.568	0.631	0.518	0.568	0.904	0.824
Adjusted Mutual Information	0.732	0.655	0.603	0.732	0.657	0.616	0.732	0.665	0.608	0.732	0.898	0.845
Normalized Mutual Information	0.734	0.659	0.609	0.734	0.661	0.623	0.734	0.669	0.615	0.734	0.9	0.847
Silhouette Score	0.582	0.46	0.385	0.582	0.457	0.413	0.582	0.459	0.398	0.582	0.374	0.302
Calinski Harabasz Index	251.349	241.904	206.681	251.349	239.485	206.073	251.349	239.748	203.291	251.349	187.77	149.032
Davies-Bouldin Index	0.297	0.655	0.851	0.297	0.654	1.008	0.293	0.653	0.863	0.297	0.809	1.278
Dunn Index	0.508	0.05	0.106	0.508	0.11	0.056	0.482	0.111	0.121	0.508	0.14	0.107

Table of clustering performance metrics results for k=2,3,4

	Range	
Adjusted Rand Index	-1,1	Higher the better
Adjusted Mutual Information	0,1	Higher the better
Normalized Mutual Information	0,1	Higher the better
Silhouette Score	-1,1	Higher the better
Calinski-Harabasz Index	0, ++	Higher the better
Davies-Bouldin Index	0, ++	Lower the better
Dunn Index	0,1	Higher the better

Table showing the possible range of values for each algorithm. Higher values indicate a better clustering result for all metrics except for the Davies-Bouldin Index

The intended output/result of the Thesis is the development of a software script capable to take different datasets as an input, implement different clustering methods and export the clustering performance metrics. The correct operation of the developed script is validated in this Chapter as the script outputs are complaint with the theoretical background that was presented in the first Chapters of the Thesis. The main results and conclusions of this example use case of the software script are the following:

The k-means family algorithms perform very similarly as they have almost the same values for the different metrics. No meaningful differences are observed between the 3 algorithms for the clustering of this specific dataset. However, this could change if we implemented the clustering for datasets with different characteristics (number of samples/observations, attributes, variance and ‘shape’ of the data).

It is important to note that the ideal number of clusters according to the elbow method is different for the GMM algorithm compared to the k-means family algorithms. This means that if the clustering for all the algorithms is applied for the same number k which is ideal for k-means family algorithm, but not for GMM. Nevertheless, the score for k=2 and k=3 is very close for GMM so the impact is not significant when we compare the fourth algorithms for k=3.

It is important to note that because clustering is an unsupervised learning method the real clustering labels will not be available in most cases and the point of these algorithms is to find the best possible solution to find clusters of the datapoints by making use of different criteria. In this specific case we are working with a well-known clustering dataset where we can use the true labels to further make even more comments on how these algorithms perform.

The Adjusted Rand Index (ARI) measures the similarity between the clustering produced by the algorithm and a reference/true clustering which is available for the iris dataset. For k=2 we have better values for some clustering metrics but the ARI is lower which means that the clustering is very different than the true labels. ARI is lower (which indicates a worse clustering result) when we move further away from the ideal number of clusters which according to the elbow method is k=3. The higher values for GMM indicate that the clustering is better in terms of differentiating the different points into the correct cluster. This is expected as GMM may perform better on data with a more complex, non-linear structure or different variances within clusters compared to K-means which may perform better on data with a spherical structure and similar variances. The simplicity of the dataset, gives the ability to be able to visually assess the validity of the script results. The better clustering result can be visually validated by looking at the clustering plots presented above for GMM and the real clustering using the actual/true labels that are available through the dataset. Values for AMI and NMI are also higher for GMM for k=3 and k=4 compared to the k-means family algorithms. The value for k=4 decreases compared to k=3 as we move further away from the true clustering result.

We notice that for k=2, we have the highest silhouette score which decreases for k=3 and further decreases k=4. The reason for this is that the clusters are much better defined/separated in case of k=2 as we can visually see from the clustering plots. The silhouette value/score measures how well matched a data point is to the cluster that it belongs and not well matched to neighboring clusters. It is important to note that high values for internal clustering metrics do not necessarily mean that the clustering is close to the real one. In this case the true labels are available so we can understand from the ARI score that the real clusters are different from the ones for k=2.

The CHI shows that the centroids of different clusters are more well-separated and the data points within each cluster are more compact for k=2. The value of the index decreases as the number of clusters increase.

For k=3 and k=4 the index is lower for GMM compared to the k-means family clustering result as the index is designed to work with convex clusters, and may not work as well with non-convex or irregularly shaped clusters. The metric decreases for k=4 compared to k=3 as we move further away from the optimal cluster result.

The value of the DBI is lower for k=2 which indicates a better clustering solution. For k=3 and k=4 the metric is higher for GMM which indicates that it has a worse performance compared to the k-means family algorithms. This is because the DBI accepts that the clusters are spherical which is true for the k-means family algorithms but not for GMM.

The Dunn Index gives a measure of how separated the clusters are but it can be sensitive to outliers. As there are no outliers for k=2, we have a much higher score compared to the value for k=3 and k=4 where the clusters are not so well separated.

By comparing the metrics values for GMM with the values of the metrics for the actual labels, we can assume that GMM was very close to the real clustering result, although if we didn’t have access to the true labels, it would seem that GMM would be the worst performing algorithm of all four.

Overall, it is very complex to decide on the single most reliable clustering metric. Any evaluation metric should be used in combination with other measures to assess the quality of clustering results. It is important to note that good scores on an internal criterion do not necessarily translate into good effectiveness in an application and that the clusters are close to reality. Good scores on an internal metrics means better results in specific criteria like separation, cohesion etc. and do not ensure that we are close to the real clustering. In most situations in unsupervised learning problems the actual labels would not be available so we would have to rely on the internal clustering metrics in order to choose a clustering method that has good performance in predicting the class labels.

Ideas for future work and expansion of the project include the adjustment of the script to take multiple datasets as an input, the expansion of the number of clustering algorithms and assessment metrics and the creation of a user interface can so that the user will be able choose which clustering algorithms will be run and which metrics will be exported

References

1. https://towardsdatascience.com/advantages-and-disadvantages-of-artificial-intelligence-182a5ef6588c
2. https://rancholabs.medium.com/6-major-sub-fields-of-artificial-intelligence-77f6a5b28109
3. https://www.softwaretestinghelp.com/what-is-artificial-intelligence/
4. https://www.dataversity.net/what-is-cognitive-computing/
5. https://techlech.com/what-are-the-three-types-of-machine-learning-supervised-learning-unsupervised-learning-and-reinforcement-learning/
6. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained
7. https://www.cs.cmu.edu/~tom/pubs/Science-ML-2015.pdf
8. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects.
9. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=deb44af9d65763af055ec66b29e9b12f9b80f564
10. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
11. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. Springer Science & Business Media.
12. https://www.javatpoint.com/difference-between-supervised-and-unsupervised-learning
13. https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d
14. Ng, A. (2012). Unsupervised feature learning and deep learning. In Proceedings of the 2012 IEEE conference on computer vision and pattern recognition
15. https://www.researchgate.net/figure/Broad-classification-of-clustering-algorithms_fig2_341111980
16. Ng, A. (2014). CS229 Lecture notes: Unsupervised learning. Stanford University.
17. Scikit-learn documentation on clustering: https://scikit-learn.org/stable/modules/clustering.html
18. Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques
19. Jain, A. K., & Dubes, R. C. (1988). Algorithms for clustering data. Prentice-Hall.
20. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a).
21. https://devopedia.org/k-means-clustering#Singh-2018
22. https://scikit-learn.org/stable/modules/clustering.html#k-means
23. Singh, Seema. 2018. "K-Means Clustering." Data Driven Investor
24. Page, Justin T, Zachary S Liechty, Mark D Huynh, and Joshua A Udall. 2014. "BamBam: genome sequence analysis tools for biologists." BMC Research Notes, November
25. https://www.academia.edu/8446443/K_Medoid_Clustering_Algorithm_A_Review

26. https://stats.stackexchange.com/questions/156210/an-example-where-the-output-of-the-k-medoid-algorithm-is-different-than-the-outp
27. https://www.geeksforgeeks.org/partitioning-method-k-mean-in-data-mining/
28. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html
29. https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans
30. https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556
31. https://towardsdatascience.com/gaussian-mixture-models-gmm-6e95cbc38e6e
32. https://scikit-learn.org/stable/modules/mixture.html
33. https://www.sciencedirect.com/topics/engineering/gaussian-mixture-model
34. https://s3.amazonaws.com/assets.datacamp.com/production/course_10628/slides/chapter4.pdf
35. https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/
36. https://www.researchgate.net/publication/339670247_Determining_The_Appropiate_Cluster_Number_Using_Elbow_Method_for_K-Means_Algorithm
37. https://www.researchgate.net/figure/Elbow-plots-of-the-Bayesian-information-criterion-BIC-and-consistent-Akaike-information_fig2_347946003
38. https://machinelearningmastery.com/probabilistic-model-selection-measures/
39. https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods/
40. http://www.sthda.com/english/wiki/wiki.php?id_contents=7952
41. https://blog.paperspace.com/ml-evaluation-metrics-part-2/
42. https://www.sciencedirect.com/topics/computer-science/adjusted-rand-index
43. https://jmlr.csail.mit.edu/papers/volume11/vinh10a/vinh10a.pdf
44. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html#sklearn.metrics.adjusted_mutual_info_score
45. https://luisdrita.com/normalized-mutual-information-a10785ba4898
46. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html
47. https://www.sciencedirect.com/science/article/pii/0377042787901257?via%3Dihub
48. https://www.researchgate.net/figure/Derivation-of-the-Overall-Silhouette-Coefficient-OverallSil_fig1_221570710
49. https://pyshark.com/calinski-harabasz-index-for-k-means-clustering-evaluation-using-python/
50. Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. Communications in Statistics-theory and methods
51. Halkidi, M., Batistakis, Y., & Vazirgiannis, M. (2001). On clustering validation techniques. Journal of intelligent information systems
52. https://pyshark.com/davies-bouldin-index-for-k-means-clustering-evaluation-in-python/

53. Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence,
54. https://pyshark.com/dunn-index-for-k-means-clustering-evaluation-using-python/
55. Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. Journal of Cybernetics
56. https://scikit-learn.org/stable/getting_started.html
57. https://pandas.pydata.org/
58. https://numpy.org/
59. https://matplotlib.org/
60. https://www.mghassany.com/courses/MLcourse/pw-6.html