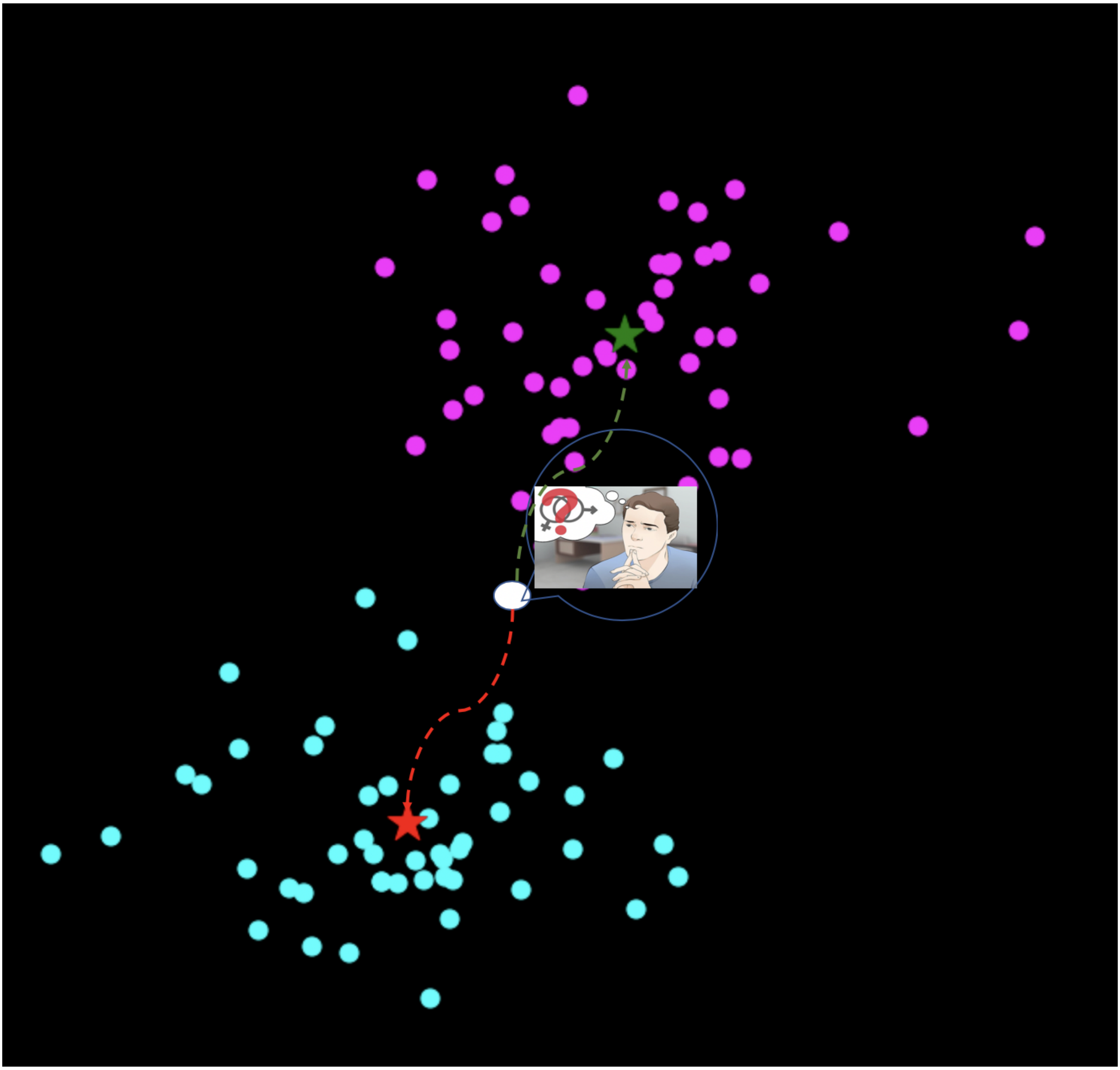
Published in Towards Data Science · Follow

Imad Dabbura · Follow
Sep 17, 2018 · 13 min read

K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks



the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application-specific.

Clustering analysis can be done on the basis of features where we try to find subgroups of samples based on features or on the basis of samples where we try to find subgroups of features based on samples. We'll cover here clustering based on features. Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.

In this post, we will cover only **Kmeans** which is considered as one of the most used clustering algorithms due to its simplicity.

Kmeans Algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to

- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called **Expectation-Maximization**. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster. Below is a break down of how we can solve it mathematically (feel free to skip it).

The objective function is:

$$J = \sum_{i=1}^{m}\sum_{k=1}^{K} w_{ik}\|x^i - \mu_k\|^2 \qquad (1)$$

where wik=1 for data point xi if it belongs to cluster *k*; otherwise, wik=0. Also, μk is the centroid of xi's cluster.

It's a minimization problem of two parts. We first minimize J w.r.t. wik and treat μk fixed. Then we minimize J w.r.t. μk and treat wik fixed. Technically speaking, we differentiate J w.r.t. wik first and update cluster assignments (*E-step*). Then we differentiate J w.r.t. μk and recompute the centroids after the cluster assignments from previous step (*M-step*). Therefore, E-step is:

$$\frac{\partial J}{\partial w_{ik}} = \sum_{i=1}^{m}\sum_{k=1}^{K}\|x^i - \mu_k\|^2$$

$$\Rightarrow w_{ik} = \begin{cases} 1 & \text{if } k = argmin_j \; \|x^i - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

In other words, assign the data point xi to the closest cluster judged by its sum of squared distance from cluster's centroid.

And M-step is:

$$\frac{\partial J}{\partial \mu_k} = 2\sum_{i=1}^{m} w_{ik}(x^i - \mu_k) = 0$$

$$\Rightarrow \mu_k = \frac{\sum_{i=1}^{m} w_{ik}x^i}{} \qquad (3)$$

Few things to note here:

- Since clustering algorithms including kmeans use distance-based measurements to determine the similarity between data points, it's recommended to standardize the data to have a mean of zero and a standard deviation of one since almost always the features in any dataset would have different units of measurements such as age vs income.
- Given kmeans iterative nature and the random initialization of centroids at the start of the algorithm, different initializations may lead to different clusters since kmeans algorithm may *stuck in a local optimum and may not converge to global optimum*. Therefore, it's recommended to run the algorithm using different initializations of centroids and pick the results of the run that that yielded the lower sum of squared distance.
- Assignment of examples isn't changing is the same thing as no change in within-cluster variation:

$$\frac{1}{m_k}\sum_{i=1}^{m_k} ||x^i - \mu_{c^k}||^2 \tag{4}$$

Implementation

We'll use simple implementation of kmeans here to just illustrate some concepts. Then we will use `sklearn` implementation that is more efficient take care of many things for us.

Applications

kmeans algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc. The goal usually when we undergo a cluster analysis is either:

1. Get a meaningful intuition of the structure of the data we're dealing with.
2. Cluster-then-predict where different models will be built for different subgroups if we believe there is a wide variation in the behaviors of different subgroups. An example of that is clustering patients into different subgroups and build a model for each subgroup to predict the probability of the risk of having heart attack.

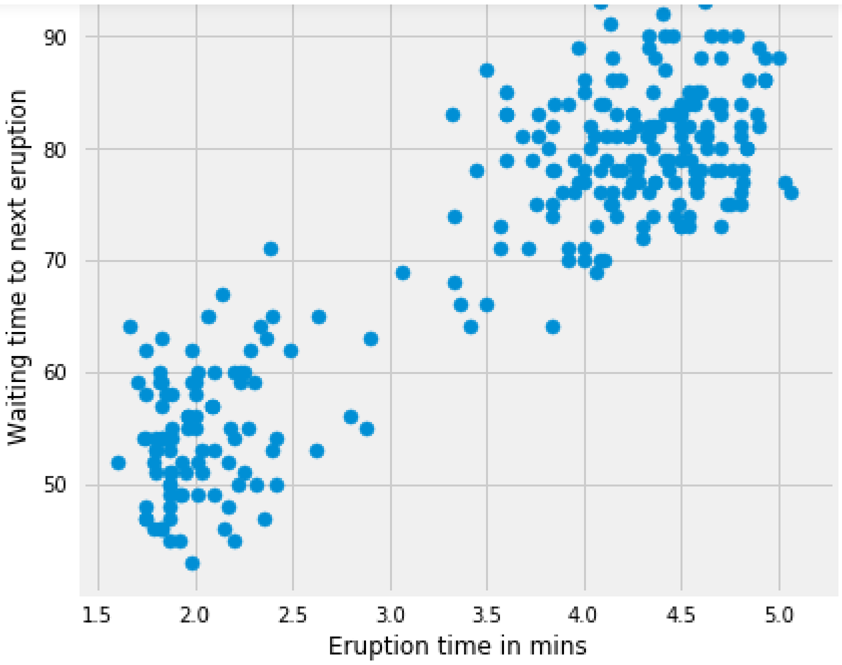
In this post, we'll apply clustering on two cases:

- Geyser eruptions segmentation (2D dataset).
- Image compression.

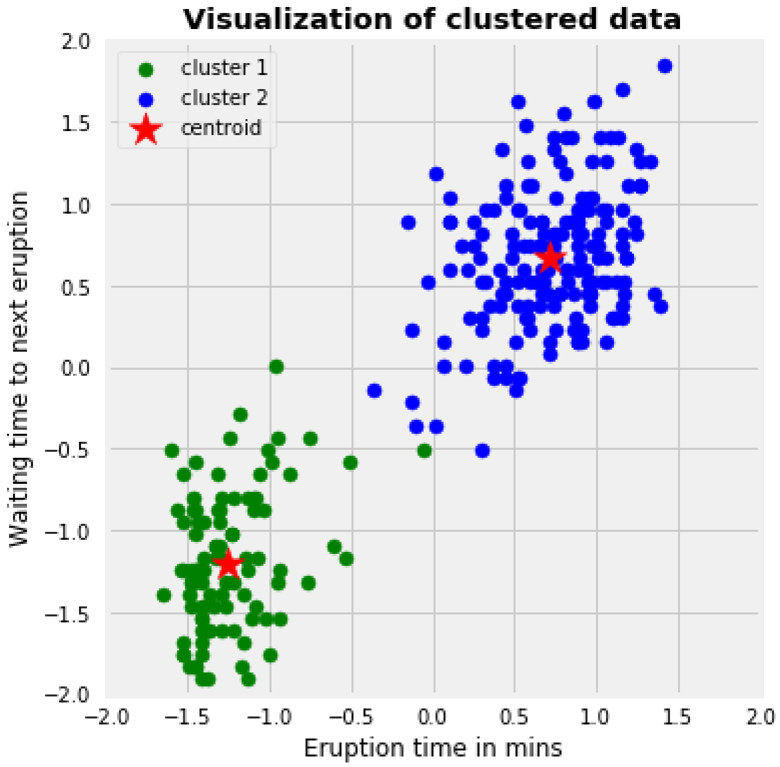
Kmeans on Geyser's Eruptions Segmentation

We'll first implement the kmeans algorithm on 2D dataset and see how it works. The dataset has 272 observations and 2 features. The data covers the waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. We will try to find *K* subgroups within the data points and group them accordingly. Below is the description of the features:

- eruptions (float): Eruption time in minutes.
- waiting (int): Waiting time to next eruption.



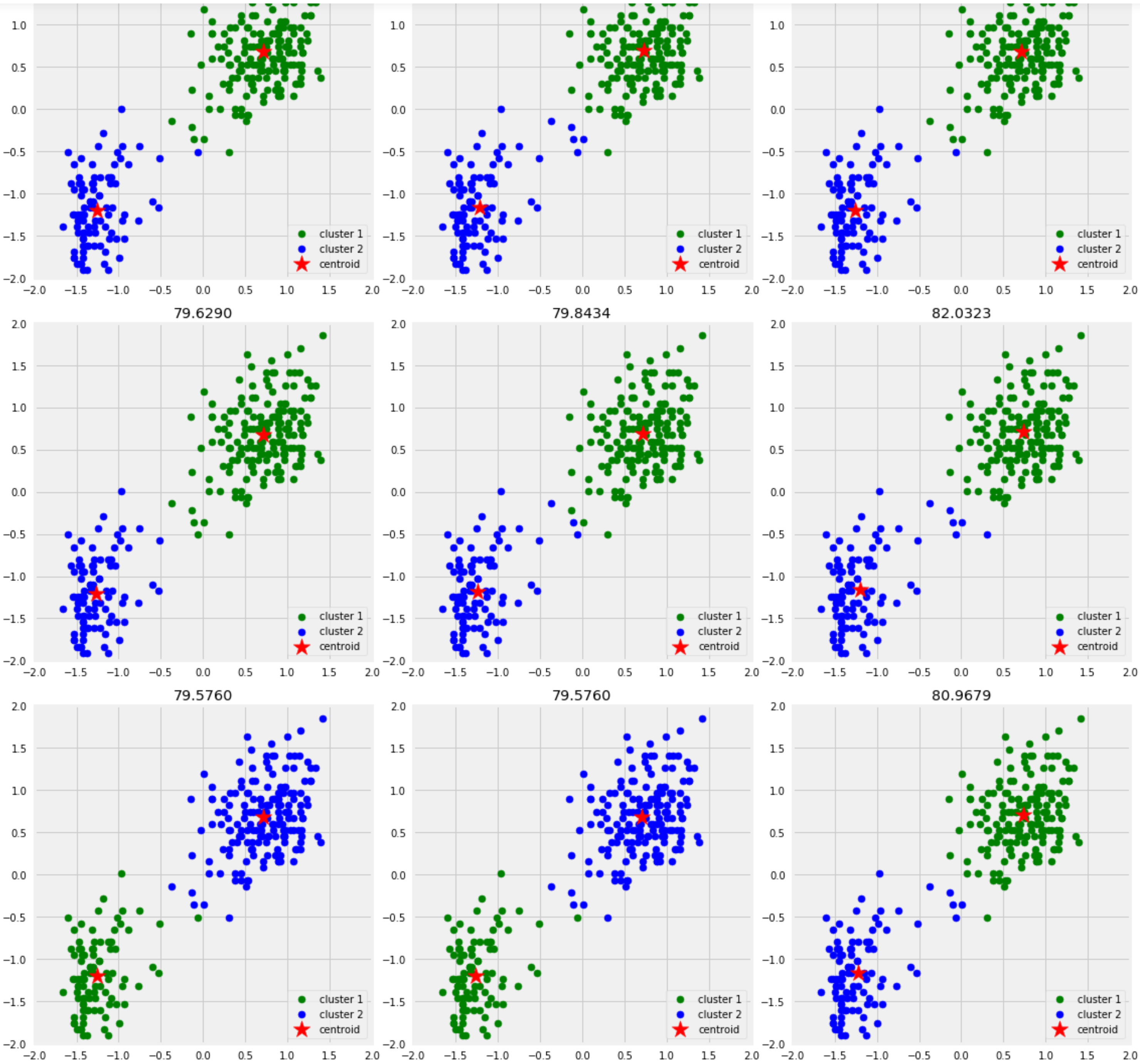
We'll use this data because it's easy to plot and visually spot the clusters since its a 2-dimension dataset. It's obvious that we have 2 clusters. Let's standardize the data first and run the kmeans algorithm on the standardized data with K=2.



The above graph shows the scatter plot of the data colored by the cluster they belong to. In this example, we chose K=2. The symbol '*' is the centroid of each cluster. We can think of those 2 clusters as geyser had different kinds of behaviors under different scenarios.

Next, we'll show that different initializations of centroids may yield to different results. I'll use 9 different `random_state` to change the initialization of the centroids and plot the results. The title of each plot will be the sum of squared distance of each initialization.

As a side note, this dataset is considered very easy and converges in less than 10 iterations. Therefore, to see the effect of random initialization on convergence, I am going to go with 3 iterations to illustrate the concept. However, in real world applications, datasets are not at all that clean and nice!



As the graph above shows that we only ended up with two different ways of clusterings based on different initializations. We would pick the one with the lowest sum of squared distance.

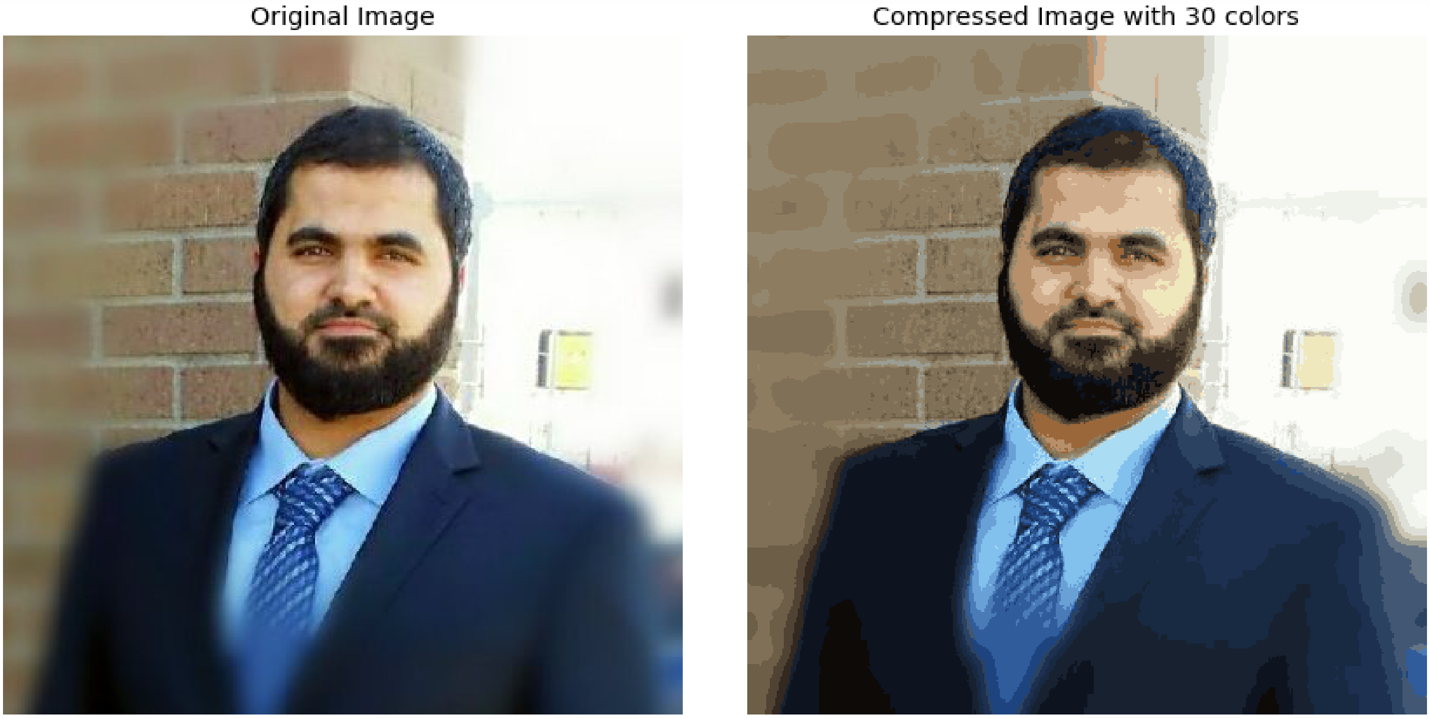
Kmeans on Image Compression

In this part, we'll implement kmeans to compress an image. The image that we'll be working on is 396 x 396 x 3. Therefore, for each pixel location we would have 3 8-bit integers that specify the red, green, and blue intensity values. Our goal is to reduce the number of colors to 30 and represent (compress) the photo using those 30 colors only. To pick which colors to use, we'll use kmeans algorithm on the image and treat every pixel as a data point. That means reshape the image from height x width x channels to (height * width) x channel, i,e we would have 396 x 396

627,984 bits. The huge difference comes from the fact that we'll be using centroids as a lookup for pixels' colors and that would reduce the size of each pixel location to 4-bit instead of 8-bit.

From now on we will be using `sklearn` implementation of kmeans. Few thing to note here:

- `n_init` is the number of times of running the kmeans with different centroid's initialization. The result of the best one will be reported.
- `tol` is the within-cluster variation metric used to declare convergence.
- The default of `init` is **k-means++** which is supposed to yield a better results than just random initialization of centroids.



We can see the comparison between the original image and the compressed one. The compressed image looks close to the original one which means we're able to retain the majority of the characteristics of the original image. With smaller number of clusters we would have higher compression rate at the expense of image quality. As a side note, this image compression method is called *lossy data compression* because we can't reconstruct the original image from the compressed image.

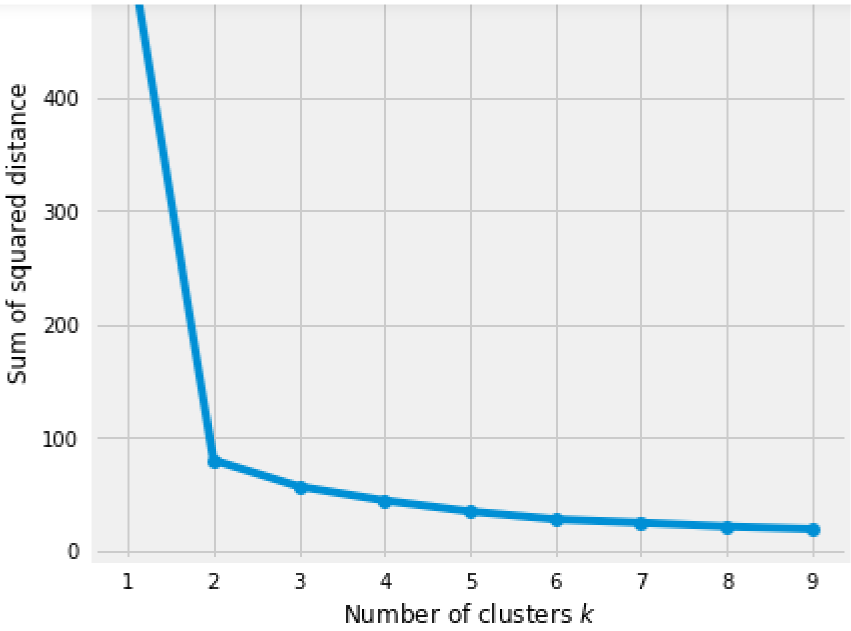
Evaluation Methods

Contrary to supervised learning where we have the ground truth to evaluate the model's performance, clustering analysis doesn't have a solid evaluation metric that we can use to evaluate the outcome of different clustering algorithms. Moreover, since kmeans requires *k* as an input and doesn't learn it from data, there is no right answer in terms of the number of clusters that we should have in any problem. Sometimes domain knowledge and intuition may help but usually that is not the case. In the cluster-predict methodology, we can evaluate how well the models are performing based on different *K* clusters since clusters are used in the downstream modeling.

In this post we'll cover two metrics that may give us some intuition about *k*:

- Elbow method
- Silhouette analysis

at the spot where SSE starts to flatten out and forming an elbow. We’ll use the geyser dataset and evaluate SSE for different values of *k* and see where the curve might form an elbow and flatten out.



The graph above shows that k=2 is not a bad choice. Sometimes it's still hard to figure out a good number of clusters to use because the curve is monotonically decreasing and may not show any elbow or has an obvious point where the curve starts flattening out.

Silhouette Analysis

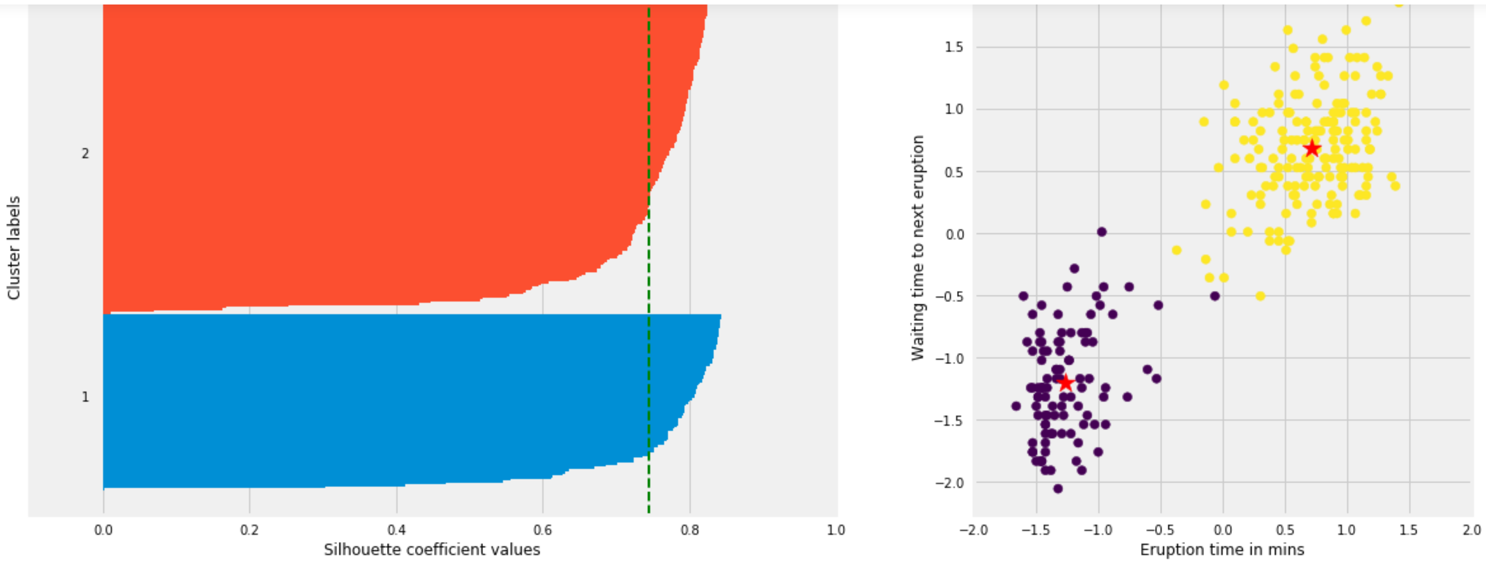
Silhouette analysis can be used to determine the degree of separation between clusters. For each sample:

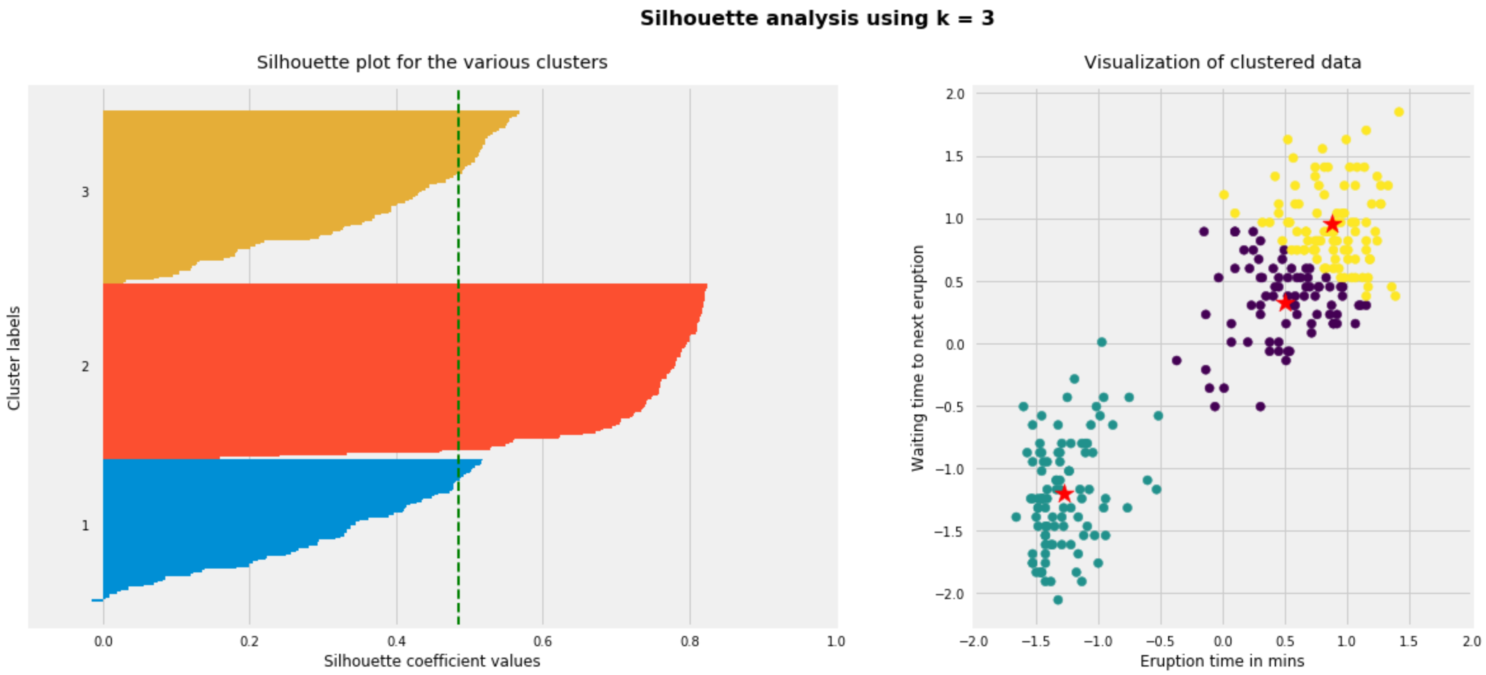
- Compute the average distance from all data points in the same cluster (ai).
- Compute the average distance from all data points in the closest cluster (bi).
- Compute the coefficient:

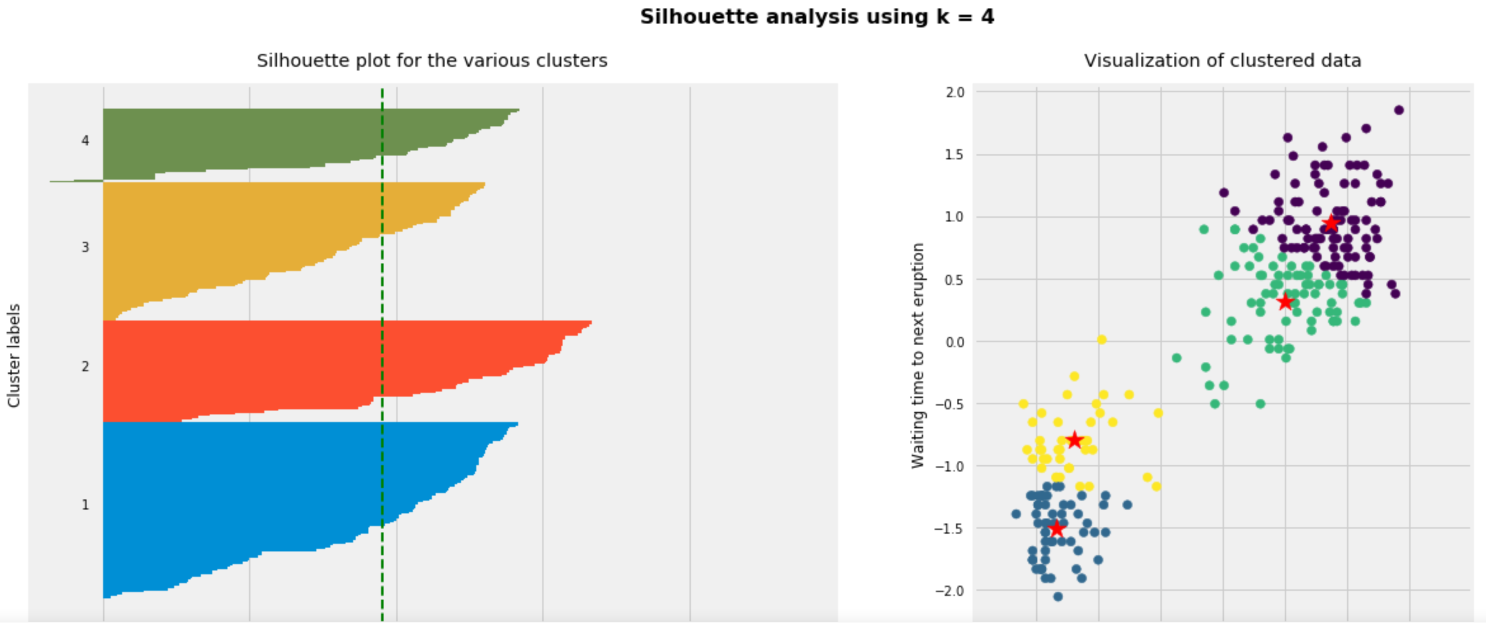
$$\frac{b^i - a^i}{max(a^i, b^i)}$$

The coefficient can take values in the interval [-1, 1].

- If it is 0 –> the sample is very close to the neighboring clusters.
- It it is 1 –> the sample is far away from the neighboring clusters.
- It it is -1 –> the sample is assigned to the wrong clusters.


2
1
Cluster labels
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette coefficient values
Waiting time to next eruption
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Eruption time in mins


Silhouette analysis using k = 3
Silhouette plot for the various clusters
3
2
1
Cluster labels
0.0
0.2
0.4
0.6
0.8
1.0
Silhouette coefficient values
Visualization of clustered data
Waiting time to next eruption
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Eruption time in mins

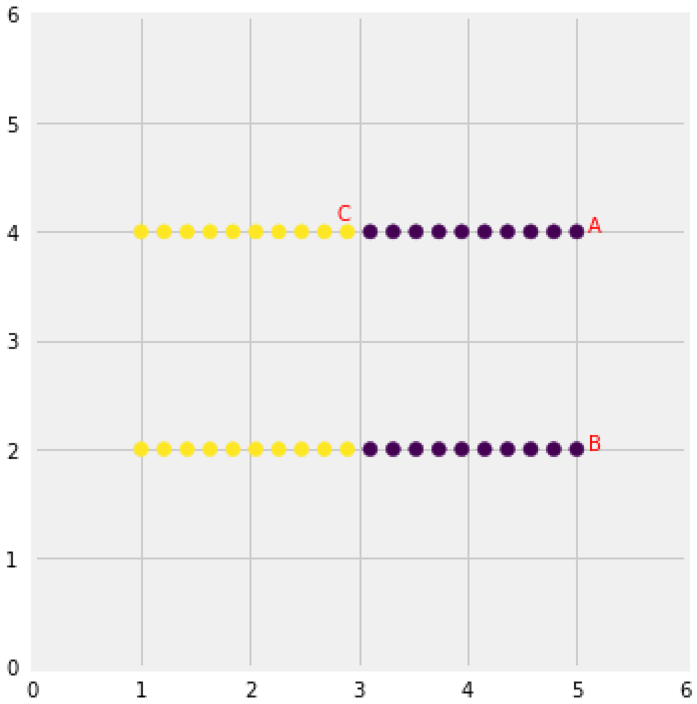

Silhouette analysis using k = 4
Silhouette plot for the various clusters
4
3
2
1
Cluster labels
Visualization of clustered data
Waiting time to next eruption
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0

the silhouette plot gives an indication of how big each cluster is. The plot shows that cluster 1 has almost double the samples than cluster 2. However, as we increased `n_clusters` to 3 and 4, the average silhouette score decreased dramatically to around 0.48 and 0.39 respectively. Moreover, the thickness of silhouette plot started showing wide fluctuations. The bottom line is: Good `n_clusters` will have a well above 0.5 silhouette average score as well as all of the clusters have higher than the average score.

Drawbacks

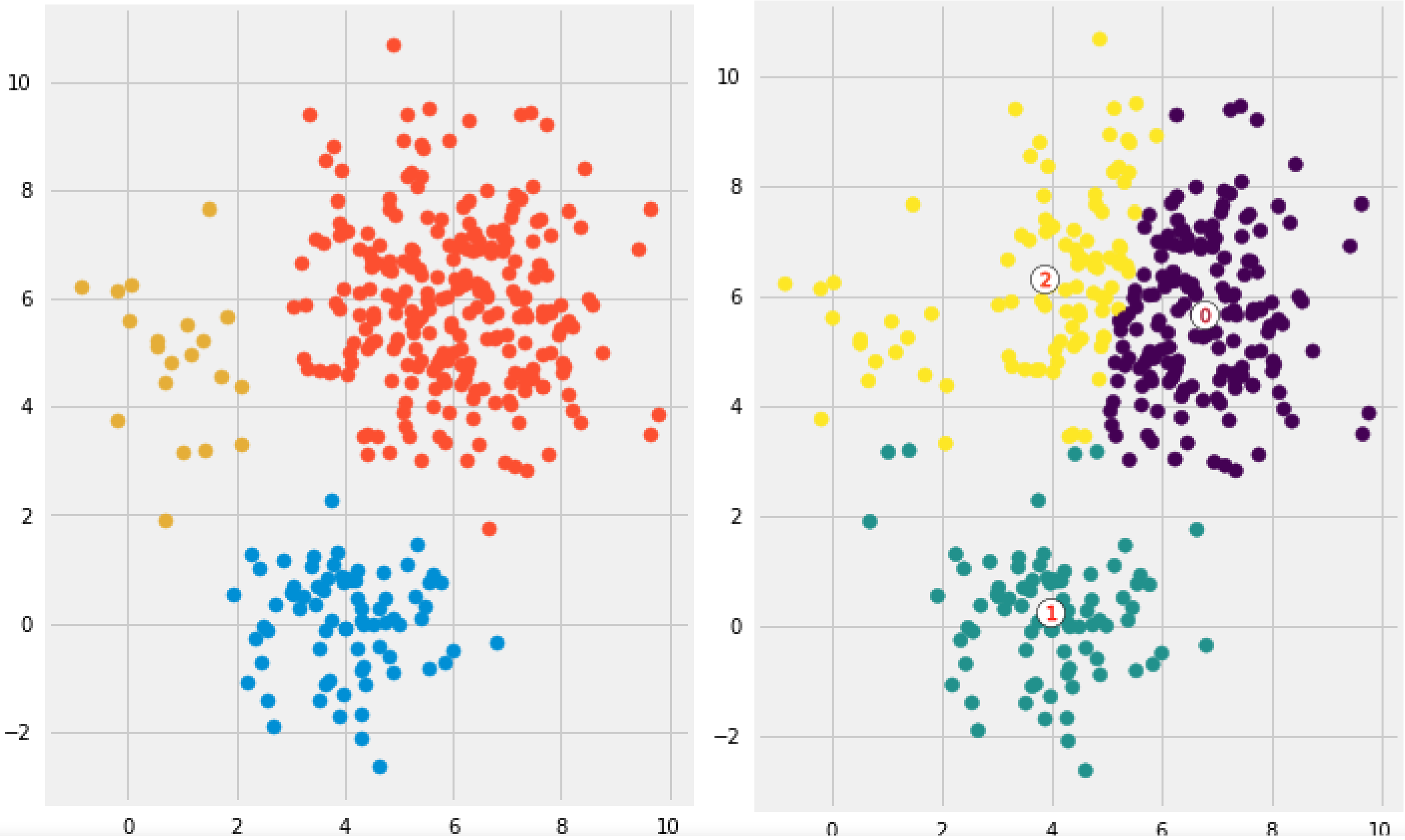
Kmeans algorithm is good in capturing structure of the data if clusters have a spherical-like shape. It always try to construct a nice spherical shape around the centroid. That means, the minute the clusters have a complicated geometric shapes, kmeans does a poor job in clustering the data. We'll illustrate three cases where kmeans will not perform well.

First, kmeans algorithm doesn't let data points that are far-away from each other share the same cluster even though they obviously belong to the same cluster. Below is an example of data points on two different horizontal lines that illustrates how kmeans tries to group half of the data points of each horizontal lines together.



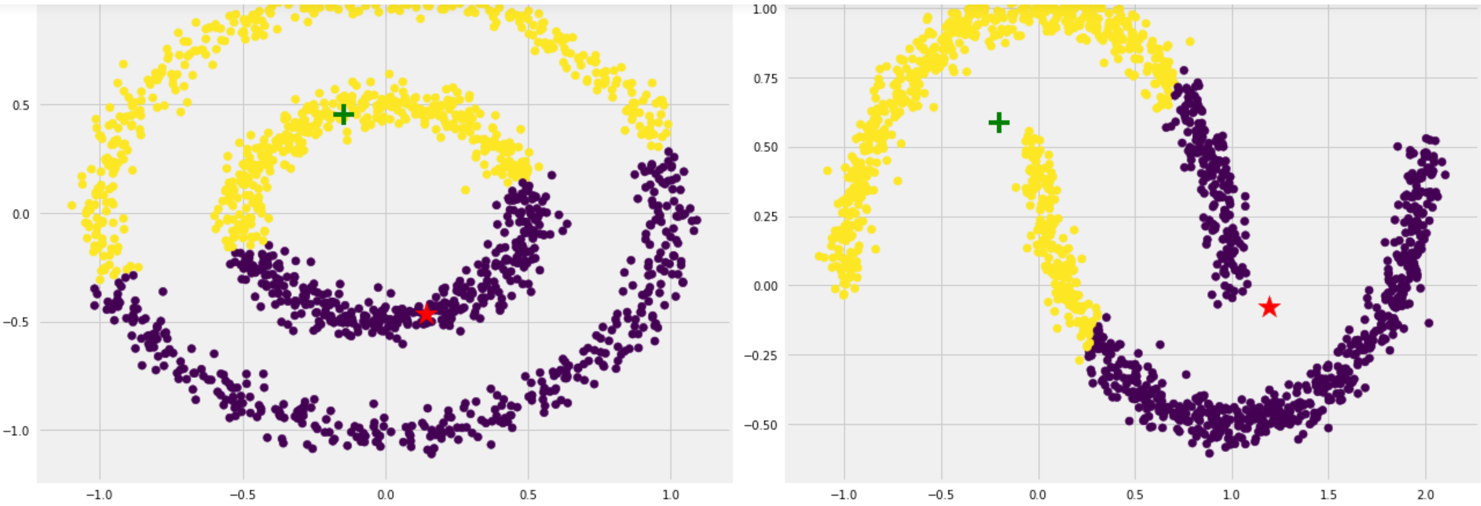
Kmeans considers the point ‘B’ closer to point ‘A’ than point ‘C’ since they have non-spherical shape. Therefore, points ‘A’ and ‘B’ will be in the same cluster but point ‘C’ will be in a different cluster. Note the **Single Linkage** hierarchical clustering method gets this right because it doesn’t separate similar points).

Second, we’ll generate data from multivariate normal distributions with different means and standard deviations. So we would have 3 groups of data where each group was generated from different multivariate normal distribution (different mean/standard deviation). One group will have a lot more data points than the other two combined. Next, we’ll run kmeans on the data with K=3 and see if it will be able to cluster the data correctly. To make the comparison easier, I am going to plot first the data colored based on the distribution it came from. Then I will plot the same data but now colored based on the clusters they have been assigned to.

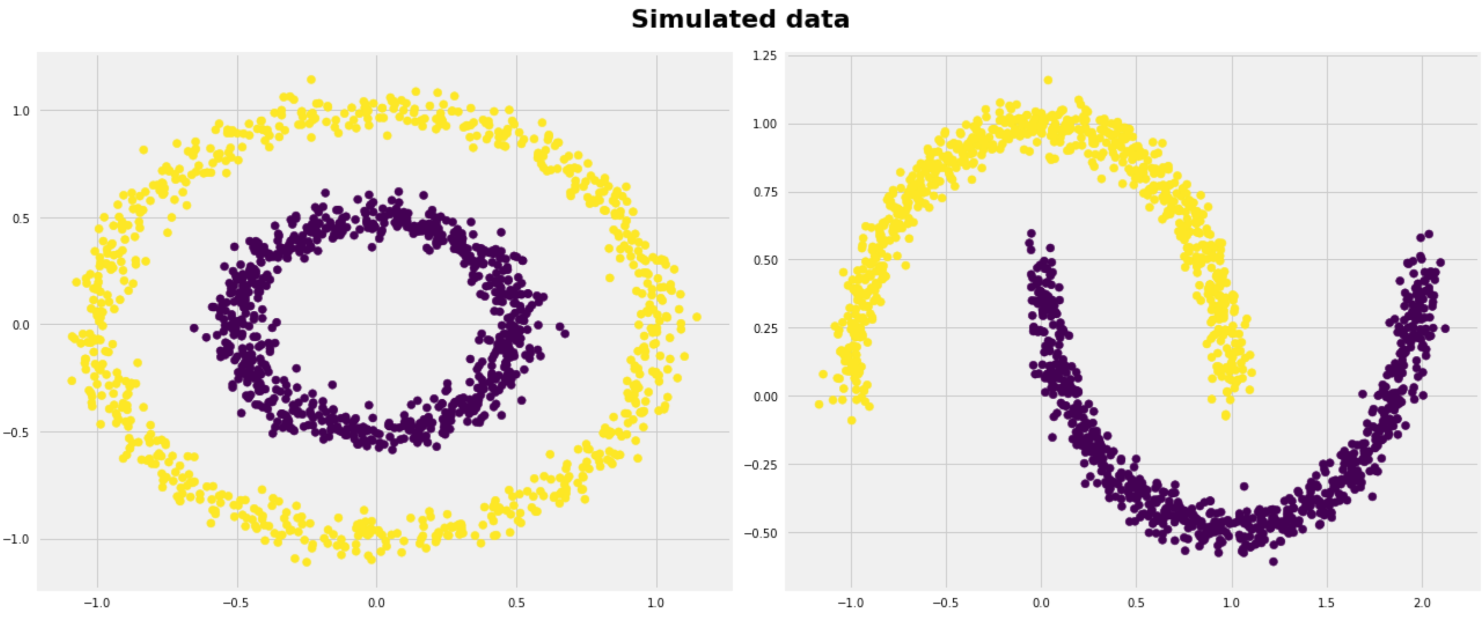

10
8
6
4
2
0
−2
0
2
4
6
8
10
10
8
6
4
2
0
−2
0
2
4
6
8
10
2
0
1

points in smaller clusters may be left away from the centroid in order to focus more on the larger cluster.

Last, we'll generate data that have complicated geometric shapes such as moons and circles within each other and test kmeans on both of the datasets.



As expected, kmeans couldn’t figure out the correct clusters for both datasets. However, we can help kmeans perfectly cluster these kind of datasets if we use kernel methods. The idea is we transform to higher dimensional representation that make the data linearly separable (the same idea that we use in SVMs). Different kinds of algorithms work very well in such scenarios such as `SpectralClustering` , see below:



Conclusion

Kmeans clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of kmeans is to group data points into distinct non-overlapping subgroups. It does a very good job when the clusters have a kind of spherical shapes. However, it suffers as the geometric shapes of clusters deviates from spherical shapes. Moreover, it also doesn’t learn the number of clusters from the data and requires it to be pre-defined. To be a good practitioner, it’s good to know the assumptions behind algorithms/methods so that you would have a pretty good idea about the strength and weakness of each method. This will help you decide when to use each method and under what circumstances. In this post, we covered both strength, weaknesses, and some evaluation methods related to kmeans.

Below are the main takeaways:

- Scale/standardize the data when applying kmeans algorithm.
- Elbow method in selecting number of clusters doesn’t usually work because the error function is monotonically decreasing for all *k*s.
- Kmeans gives more weight to the bigger clusters.

- If there is overlapping between clusters, kmeans doesn't have an intrinsic measure for uncertainty for the examples belong to the overlapping region in order to determine for which cluster to assign each data point.
- Kmeans may still cluster the data even if it can't be clustered such as data that comes from *uniform distributions*.

The notebook that created this post can be found here.

Originally published at imaddabbura.github.io on September 17, 2018.