**K-Means, silhouette method, and clustering for dimensionality reduction**

[[Nikita Volzhin](https://medium.com/@volzhinnv?source=post_page-----cd50e016d36e--------------------------------)](https://medium.com/@volzhinnv?source=post_page-----cd50e016d36e--------------------------------)

[Nikita Volzhin](https://medium.com/@volzhinnv?source=post_page-----cd50e016d36e--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F4e81753aeef4&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40volzhinnv%2Fk-means-silhouette-method-and-clustering-for-dimensionality-reduction-cd50e016d36e&user=Nikita+Volzhin&userId=4e81753aeef4&source=post_page-4e81753aeef4----cd50e016d36e---------------------post_header-----------)

7 min read

·

Jan 21

2

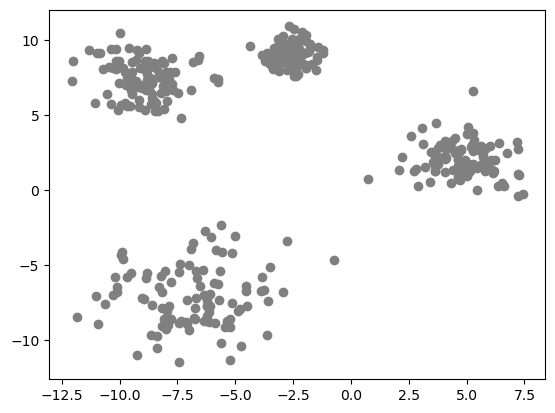
**Intro**

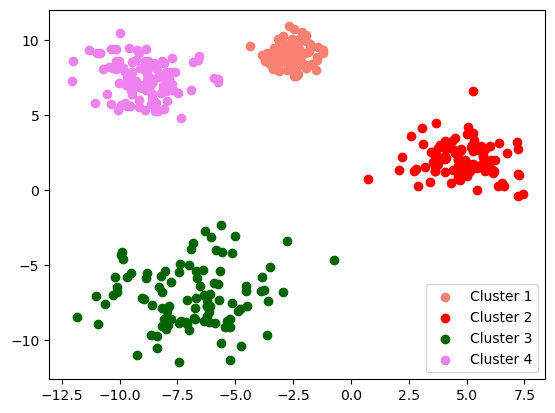
Given an unlabeled dataset (which means you do not have a dependent variable you want to predict) you can select several groups and assign each observation to a particular group. That is called clustering, hence, the groups are called clusters. On its own clustering may seem quite a useless technique: we get let’s say 5, 7, or 37 clusters each of which may even not convey any particular meaning. However, they are highly useful when combined with other machine learning techniques or for auditory targeting and recommendation systems. In this article, I will explain what clustering is, how it works on the example of K-Means, and how to use it for dimensionality reduction.

**Theory**

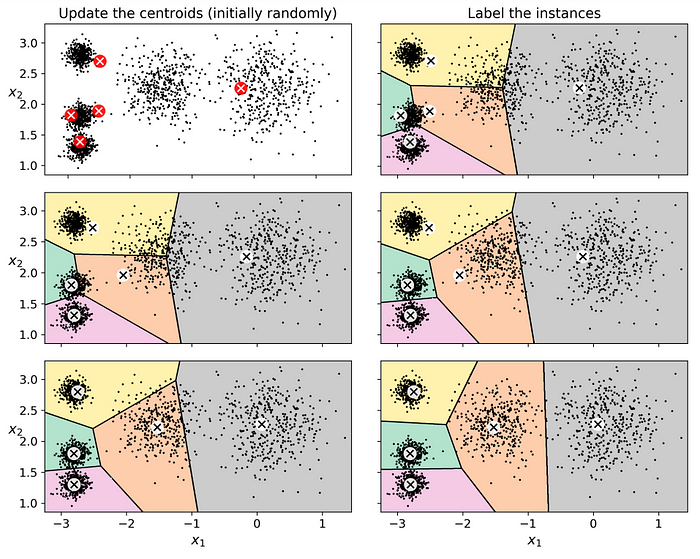
Clustering is a type of unsupervised learning, which means there is no “correct answer”, no y-vector containing the labels we want to predict. The algorithm should on its own detect certain patterns in the data and infer groups of observations which have something similar (usually that is the observations located close to each other).

Take a look at this simple visualization. On the left there is an unlabeled dataset, whereas on the right the most obvious way to cluster it.





One of the most popular and simple clustering algorithms in **KMeans Clustering.**You provide it with the number of clusters, for example, 4, it randomly generates 4 points (called **centroid**), then it assigns each point to a particular cluster based on which centroid is closer to the given point (each centroid is in charge of one cluster). Then the algorithm calculates the centers of mass of the clusters and moves the centroids there. Then algorithm repeats itself until the system is in equilibrium. It is illustrated in the picture below.



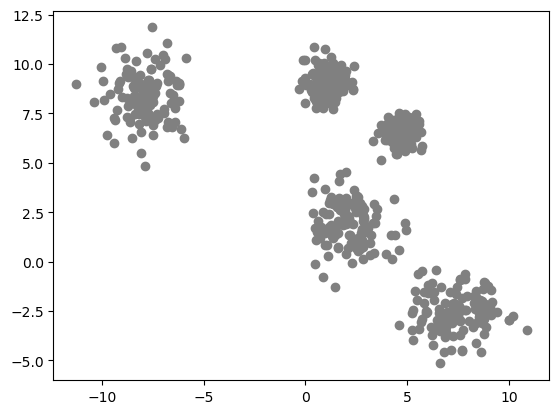
The algorithm is straightforward, but it has some drawbacks. The main one is that since the initialization of the centroids is random, it can give different results on the same dataset. To avoid this inconsistency, by default, it is run 10 times and it selects the best performance among these 10 (you can regulate this number using hyperparameters).

Ok, but how does it know which clustering is the best one? Easily! It counts the mean square distance between each point and the centroid of its cluster, the result is called **inertia** (also called**within-cluster sum of squares (WCSS)**). It is quite intuitive that the less inertia the better the solution.

**Practice**

Let’s generate a 2D dataset so we can visualize the result (in the next example I’m gonna use a higher-dimensional dataset don’t worry;)).

from sklearn.datasets import make\_blobs  
import matplotlib.pyplot as plt  
import numpy as np  
  
dataset = make\_blobs(n\_samples=600, centers=5, cluster\_std=[1.1, 0.6, 0.5, 1.2, 1.1], random\_state=32)  
X = dataset[0]  
plt.scatter(X[:, 0], X[:, 1], color = 'grey')  
plt.show()



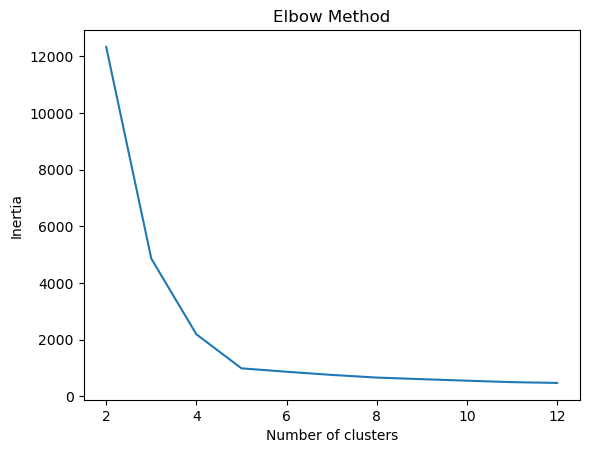
Our brain instantly detects 5 clusters. But what if we have a dataset of 30 dimesions? We won’t be able to actually see it. So, there are some analytical methods to select the right number of clusters.

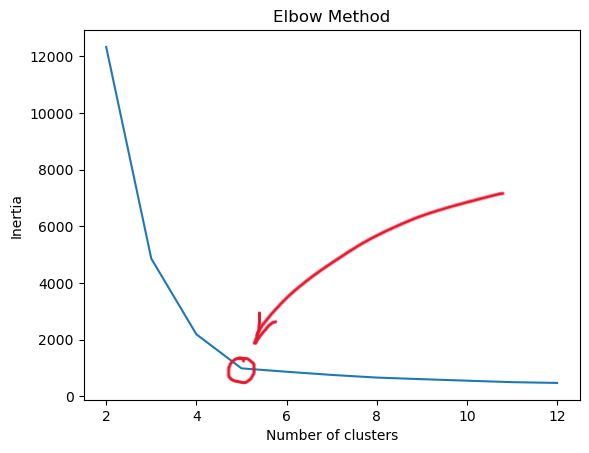
**Elbow method**

The thing is that **inertia**drops quickly when we increase the number of clusters until a certain point, and then it slows down and decreases more and more slowly. The point where it stops fast decreasing is called the “elbow” and can be detected on a graph. The code below applies KMeans with different numbers of clusters and plots the interia graph.

from sklearn.cluster import KMeans  
  
inertia = list()  
for i in range(2, 13):  
 kmeans = KMeans(n\_clusters = i)  
 kmeans.fit(X)  
 inertia.append(kmeans.inertia\_)  
  
plt.plot(range(2, 13), inertia)  
plt.title('Elbow Method')  
plt.xlabel('Number of clusters')  
plt.ylabel('Inertia')  
plt.show()

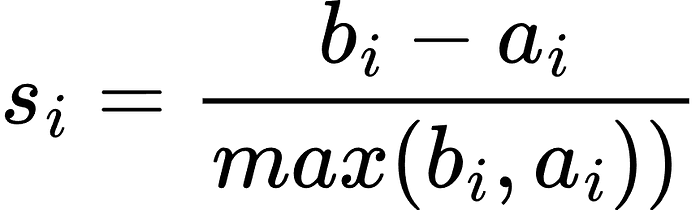
Here we can clearly see the “elbow” (I marked it in the right picture). So 5 is the optimal number of clusters (which is right).





**Silhouette Method**

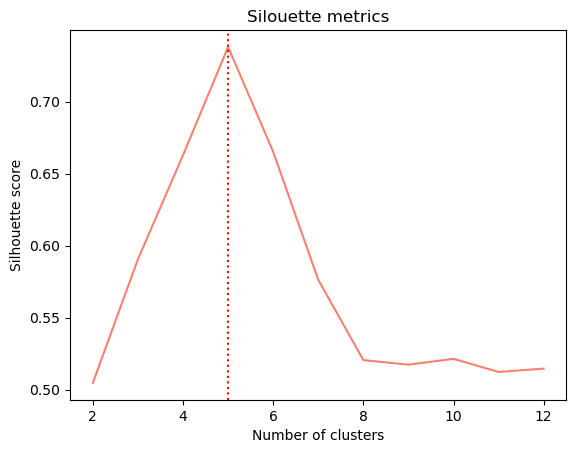
But graphs are not always as clear as this one. Sometimes the curve decreases very smoothly. In this case, you should try the **Silhouette score**. That is another metric to assess the number of clusters. Here is the formula to calculate the **silhouette coefficient** for a single instance. “a\_i” is the mean distance to the other instances in the same cluster, “b\_i” is the mean distance to the instances of the next closest cluster.



To calculate the **silhouette score** for the whole dataset, you take the mean of **silhouette coefficients** over all the instances. It might sound complicated, but the most important here is that the higher the score, the better, and it does not constantly decrease as inertia. Hence, to get the right number of clusters, you just take the one with the highest silhouette score. The code below shows how to find the score (notice that for it you need to provide the labels). The graph shows how the score changes with different numbers of clusters.

from sklearn.metrics import silhouette\_score  
  
sil\_scores = list()  
for i in range(2, 13):  
 kmeans = KMeans(n\_clusters = i, random\_state = 42)  
 kmeans.fit(X)  
 sil\_scores.append(silhouette\_score(X, kmeans.labels\_))  
  
plt.plot(range(2, 13), sil\_scores, color = 'salmon')  
plt.xlabel('Number of clusters')  
plt.ylabel('Silhouette score')  
plt.title('Silouette metrics')  
plt.axvline(x = sil\_scores.index(max(sil\_scores))+2, linestyle = 'dotted', color = 'red')   
plt.show()

On the graph I marked the right number of clusters according to Silhouette metrics with a dotted line. It shows 5 which is also correct!



**Dimensionality reduction**

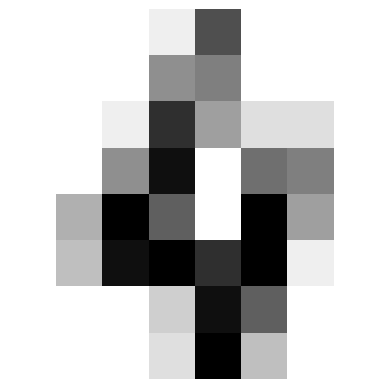
In the previous article I talked about dimensionality reduction. You can check it out by the link below if you have not read it yet.

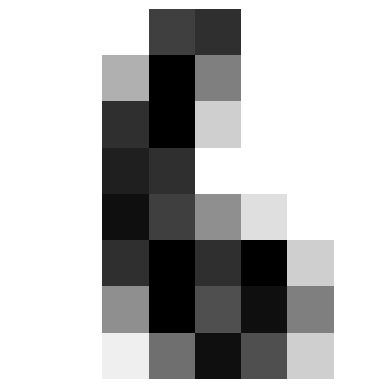
**[Dimensionaly reduction and PCA: how does it work?](https://medium.com/@volzhinnv/dimensionaly-reduction-and-pca-how-does-it-work-f2f0ad6b766b?source=post_page-----cd50e016d36e--------------------------------" \t "_blank)**

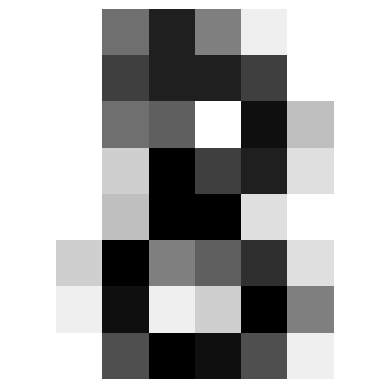
[Intro](https://medium.com/@volzhinnv/dimensionaly-reduction-and-pca-how-does-it-work-f2f0ad6b766b?source=post_page-----cd50e016d36e--------------------------------" \t "_blank)

[medium.com](https://medium.com/@volzhinnv/dimensionaly-reduction-and-pca-how-does-it-work-f2f0ad6b766b?source=post_page-----cd50e016d36e--------------------------------" \t "_blank)

It might sound counterintuitive but you can reduce dimensions using clustering as well and, moreover, the model performance may even increase. Let’s consider the example of hand-written 8x8 numbers (I printed some of them below)







First I import the dataset and split it.

from sklearn.datasets import load\_digits  
from sklearn.model\_selection import train\_test\_split  
  
X, y = load\_digits(return\_X\_y=True)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

Train logistic regression on the raw dataset and get the accuracy score. I got 0.956. Let’s see if it will worsen if I apply KMeans clustering for dimensionality reduction.

from sklearn.linear\_model import LogisticRegression  
  
log\_reg = LogisticRegression(random\_state=42)  
log\_reg.fit(X\_train, y\_train)  
log\_reg.score(X\_test, y\_test)

KMeans will assign each observation to a particular cluster and based on each instance’s distances to the centroids of the clusters our model will predict the digit. **NB:** it would be a bad idea to set the number of clusters to 10 because each digit could be written in different ways and each of these ways is likely to be assigned to a different cluster, so I set it to 50.

kmeans = KMeans(n\_clusters = 50, random\_state = 42)  
  
X\_cluster\_train = kmeans.fit\_transform(X\_train)  
X\_cluster\_test = kmeans.transform(X\_test)  
  
log\_reg.fit(X\_cluster\_train, y\_train)  
log\_reg.score(X\_cluster\_test, y\_test)

I got an accuracy score of 0.962 which is even slightly better than before clustering.

Here are two important things to mention. First, in general, the accuracy of prediction is not guaranteed to decrease after reducing dimensionality even with clustering. Moreover, usually, it will decrease. Second, *fit\_transform*does not return clusters, but distances to centroids of the clusters. So when we apply clustering for dim.reduction we go from original space to cluster-distance space (50-dimensional in our case). If you want to get clusters, just use the *predict*method instead.

**Conclusion**

Clustering is a great machine-learning technique which could be used not only for auditory targeting but also for dimensionality reduction. In this article, I explained how the KMeans algorithm works, showed 2 ways to select the right number of clusters, and how clustering can be used for dimensionality reduction. There are still a ton of ways clustering can be used for (e.g. image segmentation or semi-supervised learning). KMeans is a popular algorithm, but it is not ideal and may give poor results, so I encourage you also to take a look at some other algorithms (e.g. agglomerative clustering). You can continue learning about them on your own! Thank you for reading the article and I am happy to see any suggestions and ideas in the comments!

***P.S. Love data, science, and data science!***

[Clustering](https://medium.com/tag/clustering?source=post_page-----cd50e016d36e---------------clustering-----------------)

[K Means](https://medium.com/tag/k-means?source=post_page-----cd50e016d36e---------------k_means-----------------)

[Data Science](https://medium.com/tag/data-science?source=post_page-----cd50e016d36e---------------data_science-----------------)

[Machine Learning](https://medium.com/tag/machine-learning?source=post_page-----cd50e016d36e---------------machine_learning-----------------)