Understanding K-means Clustering in Machine Learning

Dr. Michael J. Garbade Follow

Sep 13, 2018 · 5 min read

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

AndreyBu, who has more than 5 years of machine learning experience and currently teaches people his skills, says that "the objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (*k*) of clusters in a dataset."

A cluster refers to a collection of data points aggregated together because of certain similarities.

You'll define a target number *k*, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

In other words, the K-means algorithm identifies *k* number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The *'means'* in the K-means refers to averaging of the data; that is, finding the centroid.

How the K-means algorithm works

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

- The centroids have stabilized—there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

K-means algorithm example problem

Let's see the steps on how the K-means machine learning algorithm works using the Python programming language.

We'll use the Scikit-learn library and some random data to illustrate a K-means clustering simple explanation.

Step 1: Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline
```

As you can see from the above code, we'll import the following libraries in our project:

- Pandas for reading and writing spreadsheets
- Numpy for carrying out efficient computations
- Matplotlib for visualization of data

Step 2: Generate random data

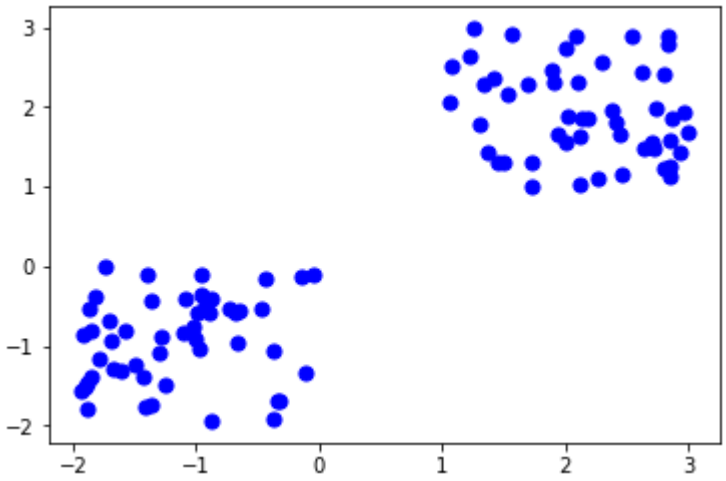
Here is the code for generating some random data in a two-dimensional space:

```
X= -2 * np.random.rand(100,2)
```

```
X1 = 1 + 2 * np.random.rand(50,2)

X[50:100, :] = X1

plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')

plt.show()
```

A total of 100 data points has been generated and divided into two groups, of 50 points each.

Here is how the data is displayed on a two-dimensional space:



Step 3: Use Scikit-Learn

We'll use some of the available functions in the Scikit-learn library to process the randomly generated data.

Here is the code:

```
from sklearn.cluster import KMeans

Kmean = KMeans(n_clusters=2)

Kmean.fit(X)
```

In this case, we arbitrarily gave *k* (n_clusters) an arbitrary value of two.

Here is the output of the K-means parameters we get if we run the code:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300
 n_clusters=2, n_init=10, n_jobs=1,
precompute_distances='auto',
 random_state=None, tol=0.0001, verbose=0)
```

Step 4: Finding the centroid

Here is the code for finding the center of the clusters:

```
Kmean.cluster_centers_
```

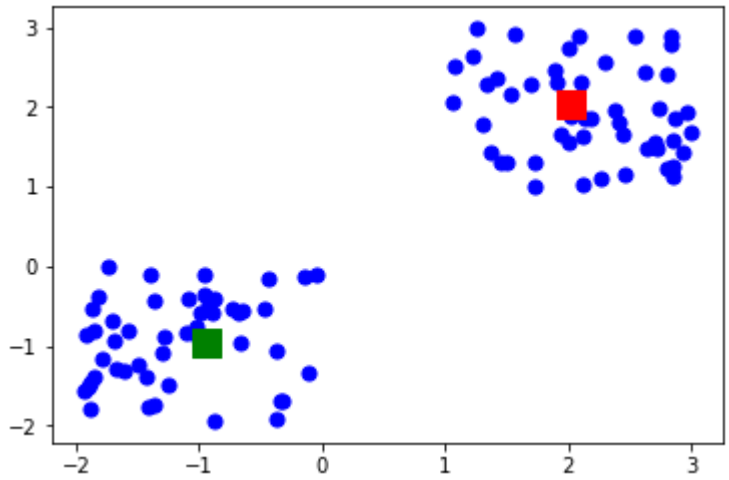
Here is the result of the value of the centroids:

```
array([[-0.94665068, -0.97138368],
 [ 2.01559419, 2.02597093]])
```

Let's display the cluster centroids (using green and red color).

```
plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')

plt.scatter(-0.94665068, -0.97138368, s=200, c='g',
marker='s')

plt.scatter(2.01559419, 2.02597093, s=200, c='r',
marker='s')

plt.show()
```

Here is the output:



Step 5: Testing the algorithm

Here is the code for getting the labels property of the K-means clustering example dataset; that is, how the data points are categorized into the two clusters.

```
Kmean.labels_
```

Here is the result of running the above K-means algorithm code:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

As you can see above, 50 data points belong to the **0** cluster while the rest belong to the **1** cluster.

For example, let's use the code below for predicting the cluster of a data point:

```
sample_test=np.array([-3.0,-3.0])
```

```
second_test=sample_test.reshape(1, -1)
```

```
Kmean.predict(second_test)
```

Here is the result:

```
array([0])
```

It shows that the test data point belongs to the **0** (green centroid) cluster.

Wrapping up

Here is the entire K-means clustering algorithm code in Python:

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
```

```
%matplotlib inline
```

```
X= -2 * np.random.rand(100,2)
```

```
X1 = 1 + 2 * np.random.rand(50,2)
```

```
X[50:100, :] = X1
```

```
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')
```

```
plt.show()
```

```
from sklearn.cluster import KMeans
```

```
Kmean = KMeans(n_clusters=2)
```

```
Kmean.fit(X)
```

```
Kmean.cluster_centers_
```

```
plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')

plt.scatter(-0.94665068, -0.97138368, s=200, c='g',
marker='s')

plt.scatter(2.01559419, 2.02597093, s=200, c='r',
marker='s')

plt.show()

Kmean.labels_

sample_test=np.array([-3.0,-3.0])

second_test=sample_test.reshape(1, -1)

Kmean.predict(second_test)
```

K-means clustering is an extensively used technique for data cluster analysis.

It is easy to understand, especially if you accelerate your learning using a K-means clustering tutorial. Furthermore, it delivers training results quickly.

However, its performance is usually not as competitive as those of the other sophisticated clustering techniques because slight variations in the data could lead to high variance.

Furthermore, clusters are assumed to be spherical and evenly sized, something which may reduce the accuracy of the K-means clustering Python results.

What's your experience with K-means clustering in machine learning?

Please share your comments below.