K-Means Clustering with scikit-learn

Lorraine Li
Follow
May 30 · 7 min read ★

Learn the fundamentals and mathematics behind the popular k-means clustering algorithm and how to implement it in `scikit-learn`!

Clustering (or **cluster analysis**) is a technique that allows us to find groups of similar objects, objects that are more related to each other than to objects in other groups. Examples of business-oriented applications of clustering include the grouping of documents, music, and movies by different topics, or finding customers that share similar interests based on common purchase behaviors as a basis for recommendation engines.

In this tutorial, we will learn about one of the most popular clustering algorithms, **k-means**, which is widely used in academia as well as in industry. We will cover:

- The basic concepts of k-means clustering
- The mathematics behind the k-means algorithm
- The advantages and disadvantages of k-means
- How to implement the algorithm on a sample dataset using `scikit-learn`
- How to visualize clusters
- How to choose the optimal *k* using the elbow method

Let's get started!

> This tutorial is adapted from *Part 3* of Next Tech's **Python Machine Learning** series, which takes you through machine learning and deep learning algorithms with Python from 0 to 100. It includes an in-browser sandboxed environment with all the necessary software and libraries pre-

installed, and projects using public datasets. You can get started for free here!

. . .

Fundamentals of K-Means Clustering

As we will see, the k-means algorithm is extremely easy to implement and is also computationally very efficient compared to other clustering algorithms, which might explain its popularity. The k-means algorithm belongs to the category of **prototype-based clustering**.

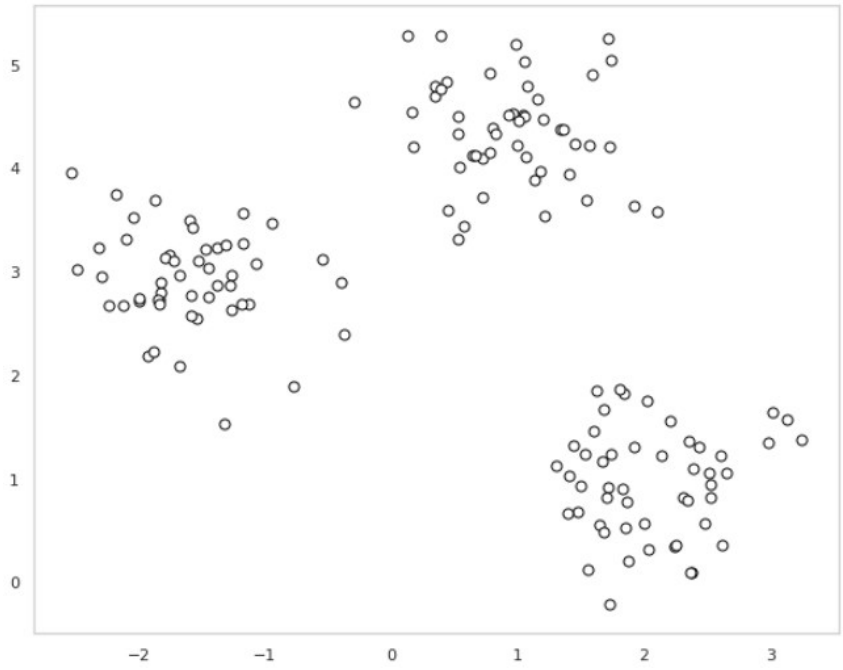
Prototype-based clustering means that each cluster is represented by a prototype, which can either be the **centroid** (*average*) of similar points with continuous features, or the **medoid** (the most *representative* or most frequently occurring point) in the case of categorical features.

While k-means is very good at identifying clusters with a spherical shape, one of the drawbacks of this clustering algorithm is that we have to specify the number of clusters, *k*, a priori. An inappropriate choice for *k* can result in poor clustering performance — we will discuss later in this tutorial how to choose *k*.

Although k-means clustering can be applied to data in higher dimensions, we will walk through the following examples using a simple two-dimensional dataset for the purpose of visualization.

You can follow along with the code in this tutorial by using a Next Tech sandbox, which has all the necessary libraries pre-installed, or if you'd prefer, you can run the snippets in your own local environment.

Once your sandbox loads, let's import the toy dataset from `scikit-learn` and visualize the datapoints:



The dataset that we just created consists of 150 randomly generated points

that are roughly grouped into three regions with higher density, which is visualized via a two-dimensional scatterplot.

In real-world applications of clustering, we do not have any ground truth category information (information provided as empirical evidence as opposed to inference) about those samples; otherwise, it would fall into the category of supervised learning. Thus, our goal is to group the samples based on their feature similarities, which can be achieved using the k-means algorithm that can be summarized by the following four steps:

1. Randomly pick *k* centroids from the sample points as initial cluster centers.
2. Assign each sample to the nearest centroid *μ^(j), j ∈ {1, ..., k}.*
3. Move the centroids to the center of the samples that were assigned to it.
4. Repeat steps 2 and 3 until the cluster assignments do not change or a user-defined tolerance or maximum number of iterations is reached.

Now, the next question is *how do we measure similarity between objects*? We can define similarity as the opposite of distance, and a commonly used distance for clustering samples with continuous features is the **squared Euclidean distance** between two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in *m*-dimensional space:

$$d(\boldsymbol{x}, \boldsymbol{y})^2 - \sum_{j=1}^{m} (x_j - y_j)^2 = \|\boldsymbol{x} - \boldsymbol{y}\|_2^2$$

Note that in the preceding equation, the index *j* refers to the *j*th dimension (feature column) of the sample points $\boldsymbol{x}$ and $\boldsymbol{y}$. We will use the superscripts *i* and *j* to refer to the sample index and cluster index, respectively.

Based on this Euclidean distance metric, we can describe the k-means algorithm as a simple optimization problem, an iterative approach for minimizing the within-cluster **Sum of Squared Errors**(**SSE**), which is sometimes also called **cluster inertia**:

$$\text{SSE} = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\|_2^2$$

Here,

$$\boldsymbol{\mu}^{(j)} \text{ is the centroid for cluster j}$$

and

$$\begin{aligned} w^{(i,j)} &= 1 \text{ if the sample } \boldsymbol{x}^{(i)} \text{ is in cluster } j \\ &= 0 \text{ otherwise} \end{aligned}$$

Note that when we are applying k-means to real-world data using a Euclidean distance metric, we want to make sure that the features are

measured on the same scale and apply z-score standardization or min-max scaling if necessary.

K-means clustering using `scikit-learn`

Now that we have learned how the k-means algorithm works, let's apply it to our sample dataset using the `KMeans` class from `scikit-learn`'s `cluster` module:

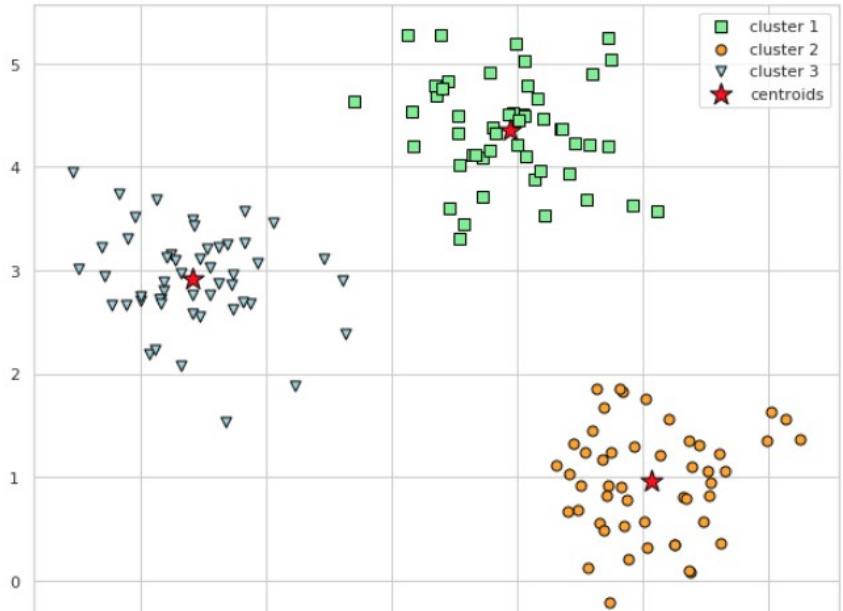
Using the preceding code, we set the number of desired clusters to `3`. We set `n_init=10` to run the k-means clustering algorithms 10 times independently with different random centroids to choose the final model as the one with the lowest SSE. Via the `max_iter` parameter, we specify the maximum number of iterations for each single run (here, `300`).

Note that the k-means implementation in `scikit-learn` stops early if it converges before the maximum number of iterations is reached. However, it is possible that k-means does not reach convergence for a particular run, which can be problematic (computationally expensive) if we choose relatively large values for `max_iter`.

One way to deal with convergence problems is to choose larger values for `tol`, which is a parameter that controls the tolerance with regard to the changes in the within-cluster sum-squared-error to declare convergence. In the preceding code, we chose a tolerance of `1e-04` (= 0.0001).

A problem with k-means is that one or more clusters can be empty. However, this problem is accounted for in the current k-means implementation in `scikit-learn`. If a cluster is empty, the algorithm will search for the sample that is farthest away from the centroid of the empty cluster. Then it will reassign the centroid to be this farthest point.

Now that we have predicted the cluster labels `y_km`, let's visualize the clusters that k-means identified in the dataset together with the cluster centroids. These are stored under the `cluster_centers_` attribute of the fitted `KMeans` object:



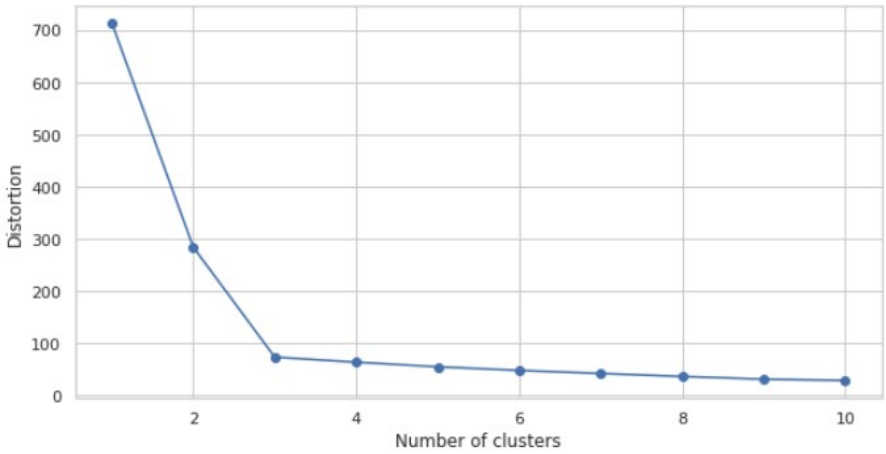
In the resulting scatterplot, we can see that k-means placed the three centroids at the center of each sphere, which looks like a reasonable grouping given this dataset.

The Elbow Method

Although k-means worked well on this toy dataset, it is important to reiterate that a drawback of k-means is that we have to specify the number of clusters, k, before we know what the optimal k is. The number of clusters to choose may not always be so obvious in real-world applications, especially if we are working with a higher dimensional dataset that cannot be visualized.

The **elbow method** is a useful graphical tool to estimate the optimal number of clusters k for a given task. Intuitively, we can say that, if k increases, the within-cluster SSE ("**distortion**") will decrease. This is because the samples will be closer to the centroids they are assigned to.

The idea behind the elbow method is to identify the value of k where the distortion begins to decrease most rapidly, which will become clearer if we plot the distortion for different values of k:



As we can see in the resulting plot, the elbow is located at $k = 3$, which is evidence that $k = 3$ is indeed a good choice for this dataset.

. . .

I hope you enjoyed this tutorial on the k-means algorithm! We explored the basic concepts and mathematics behind the k-means algorithm, how to implement k-means, and how to select an optimal number of clusters, k.

If you'd like to learn more, Next Tech's **Python Machine Learning (Part 3)** course further explores clustering algorithms and techniques such as:

- **Silhouette plots**, another method used to select the optimal k
- **k-means++**, a variant of k-means, that improves clustering results

through more clever seeding of the initial cluster centers.

- Other categories of clustering algorithms, such as **hierarchical** and **density-based clustering**, that do not require us to specify the number of clusters upfront or assume spherical structures in our dataset.

The course also explores regression analysis, sentiment analysis, and how to deploy a dynamic machine learning model to a web application. You can get started here!

Machine Learning | Tutorial | Data Science | Programming | Python

144 claps

WRITTEN BY

Lorraine Li

Follow

Data Scientist @ next.tech

Towards Data Science

Follow

Sharing concepts, ideas, and codes.

See responses (2)

More From Medium

More from Towards Data Science

6 Techniques Which Help Me Study Machine Learning Five Days Per Week

Daniel Bourke in Towards Data Science

Aug 3 · 8 min read ★

4.5K

More from Towards Data Science

I wasn't getting hired as a Data Scientist. So I sought data on who is.

Hanif Samad in Towards Data Science

Aug 1 · 12 min read ★

2.6K

More from Towards Data Science

What is Decision Intelligence?

Cassie Kozyrkov in Towards Data Science

Aug 2 · 13 min read


2K