How can the Elbow Method in Python be used to determine the optimal number of clusters?

Authored by

stats writer

June 25, 2024

The Elbow Method is a popular technique in Python used to determine the optimal number of clusters in a dataset. It works by plotting the number of clusters against the corresponding values of the within-cluster sum of squares, also known as the "elbow curve". The point on the curve where the decrease in within-cluster sum of squares starts to level off is considered the optimal number of clusters. This method helps in finding the most efficient and accurate number of clusters for a given dataset, allowing for better understanding and organization of the data. By using the Elbow Method, users can make more informed decisions when it comes to clustering data and improving the overall performance of their models.

Use the Elbow Method in Python to Find Optimal Clusters

One of the most common clustering algorithms in is known as k-means clustering.

K-means clustering is a technique in which we place each observation in a dataset into one of *K* clusters.

The end goal is to have *K* clusters in which the observations within each cluster are quite similar to each other while the observations in different clusters are quite different from each other.

When performing k-means clustering, the first step is to choose a value for *K* - the number of clusters we'd like to place the observations in.

One of the most common ways to choose a value for *K*

is known as the elbow method, which involves creating a plot with the number of clusters on the x-axis and the total within sum of squares on the y-axis and then identifying where an "elbow" or bend appears in the plot.

The point on the x-axis where the "elbow" occurs tells us the optimal number of clusters to use in the k-means clustering algorithm.

The following example shows how to use the elbow method in Python.

Step 1: Import Necessary Modules

First, we'll import all of the modules that we will need to perform k-means clustering:

```
import pandas as pd
import numpy as np
import matplotlib.pyplotas plt
from sklearn.clusterimport KMeans
from sklearn.preprocessingimport StandardScaler
```

Step 2: Create the DataFrame

Next, we'll create a DataFrame that contains three variables for 20 different basketball players:

```
#create DataFrame
df = pd.DataFrame({'points': ,
'assists': ,
'rebounds': })

#drop rows with NA values in any columns
df = df.dropna()

#create scaled DataFrame where each variable has mean of 0 and standard dev of 1
scaled_df = StandardScaler().fit_transform(df)
```

Step 3: Use Elbow Method to Find the Optimal Number of Clusters

Suppose we would like to use k-means clustering to group together players that are similar based on these three metrics.

To perform k-means clustering in Python, we can use the function from the sklearn module.

The most important argument in this function is n_clusters, which specifies how many clusters to place

the observations in.

We will then look for an "elbow" where the sum of squares begins to "bend" or level off. This point represents the optimal number of clusters.

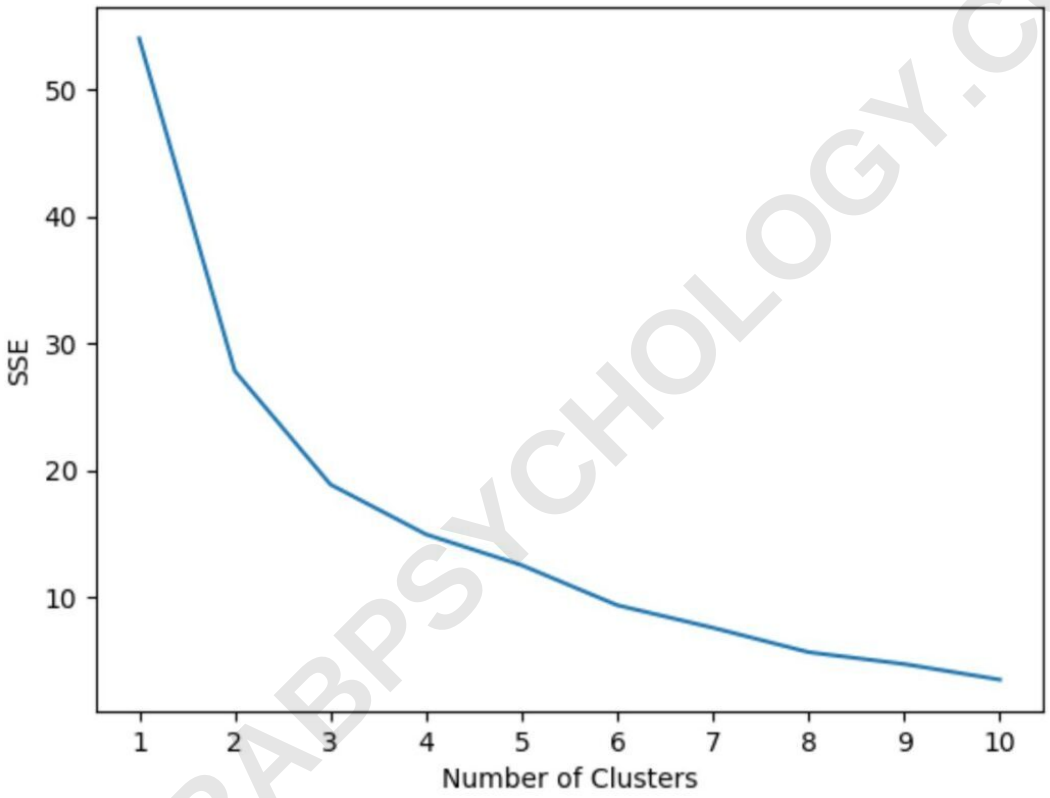
The following code shows how to create this type of plot that displays the number of clusters on the x-axis and the SSE on the y-axis:

```
#initialize kmeans parameters
kmeans_kwargs = {
"init": "random",
"n_init": 10,
"random_state": 1,
}

#create list to hold SSE values for each k
sse =
for k in range(1, 11):
kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
kmeans.fit(scaled_df)
sse.append(kmeans.inertia_)

#visualize results
```

```
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



In this plot it appears that there is an elbow or "bend" at k = 3 clusters.

Thus, we will use 3 clusters when fitting our k-means clustering model in the next step.

Step 4: Perform K-Means Clustering with Optimal *K*

The following code shows how to perform k-means clustering on the dataset using the optimal value for *k* of 3:

```
#instantiate the k-means class, using optimal number of clusters
kmeans = KMeans(init="random", n_clusters=3, n_init=10, random_state=1)

#fit k-means algorithm to data
kmeans.fit(scaled_df)

#view cluster assignments for each observation
kmeans.labels_

array()
```

The resulting array shows the cluster assignments for each observation in the DataFrame.

To make these results easier to interpret, we can add a column to the DataFrame that shows the cluster assignment of each player:

```
#append cluster assingments to original DataFrame
df = kmeans.labels_#view updated DataFrame
print(df)
```

```
points assists rebounds cluster
0 18.0 3.0 15 1
2 19.0 4.0 14 1
3 14.0 5.0 10 1
4 14.0 4.0 8 1
5 11.0 7.0 14 1
6 20.0 8.0 13 1
7 28.0 7.0 9 2
8 30.0 6.0 5 2
9 31.0 9.0 4 0
10 35.0 12.0 11 0
11 33.0 14.0 6 0
13 25.0 9.0 5 0
14 25.0 4.0 3 2
15 27.0 3.0 8 2
16 29.0 4.0 12 2
17 30.0 12.0 7 0
18 19.0 15.0 6 0
19 23.0 11.0 5 0
```

The cluster column contains a cluster number (0, 1, or 2) that each player was assigned to.

Players that belong to the same cluster have roughly similar values for the points, assists, and rebounds columns.

Note: You can find the complete documentation for the KMeans function from sklearn .

The following tutorials explain how to perform other common tasks in Python: