Published in Towards Data Science



Rukshan Pramoditha Follow

Apr 24, 2022 · 8 min read · ✦ · Listen

Save

# How to Select the Best Number of Principal Components for the Dataset

Six methods you should follow

Photo by Randy Fath on Unsplash

Selecting the best number of principal components is the major challenge when applying Principal Component Analysis (PCA) to the dataset.

In technical terms, selecting the best number of principal components is called a type of hyperparameter tuning process in which we select the optimal value for the hyperparameter **n_components** in the Scikit-learn **PCA()** class.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=?)
```

In other words, when we apply PCA to the original dataset with **p** number of variables to get a transformed dataset with **k** number of variables (principal components), **n_components** is equal to **k**, where the value of **k** is much less than the value of **p**.

Since **n_components** is a hyperparameter, it does not learn from the data. We have to manually specify its value (tune the hyperparameter) before we run the **PCA()**

function.

There is no magic rule behind selecting the optimal number for **n_components**. It depends on what we really want from PCA. Some visual inspection and domain knowledge may also be helpful to deduce the right value for **n_components**.

Today's article is devoted to discussing six effective methods that will help you to select the best number of principal components for the dataset.

Let's get started!

#### Method 1: If your sole intention of doing PCA is for data visualization, you should select 2 or 3 principal components.

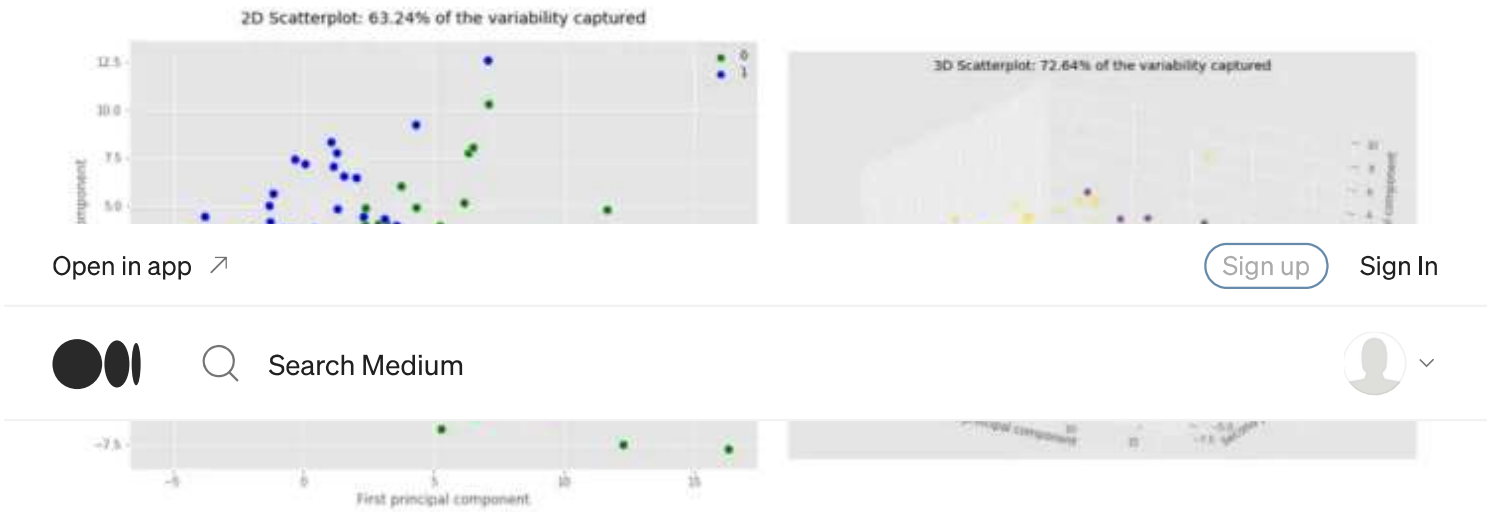
PCA is extremely useful for data visualization. Visualization of high-dimensional data can be achieved through PCA.

Since we are only familiar with 2D and 3D plots, we should convert high-dimensional data into 2 or 3-dimensional data to visualize them on 2D or 3D plots. This can be achieved through PCA.

For a 2D plot, we need to select 2 principal components.

For a 3D plot, we need to select 3 principal components.

In the following image, you can see an example for visualization of 30-dimensional *breast_cancer* data. Since the dataset contains 30 variables, its dimensionality is 30. We cannot plot 30-dimensional data in a 30D plot that we cannot imagine how it exists. For that reason, we apply PCA to the dataset and select 2 or 3 principal components which can be plotted on a 2D or 3D plot.




**Visualization of high-dimensional data using PCA** (Image by author)

These plots approximately represent the original data. This is because when we reduce the dimensionality of data, we lose some variance in the original data.

**Method 2: If you want an exact amount of variance to be kept in data after applying PCA, specify a float between 0 and 1 to the hyperparameter n_components.**

This is the easiest way to select the best number of principal components for the dataset. For example, if you want to keep 85% of the variance in the original data after applying PCA, you can specify the float 0.85 to the hyperparameter **n_components** as follows.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.85)
```

Then, the algorithm automatically selects the best number of principal components that keep 85% of the variance in the original data. If you want to know how many components that the algorithm has selected, run the following line of code.

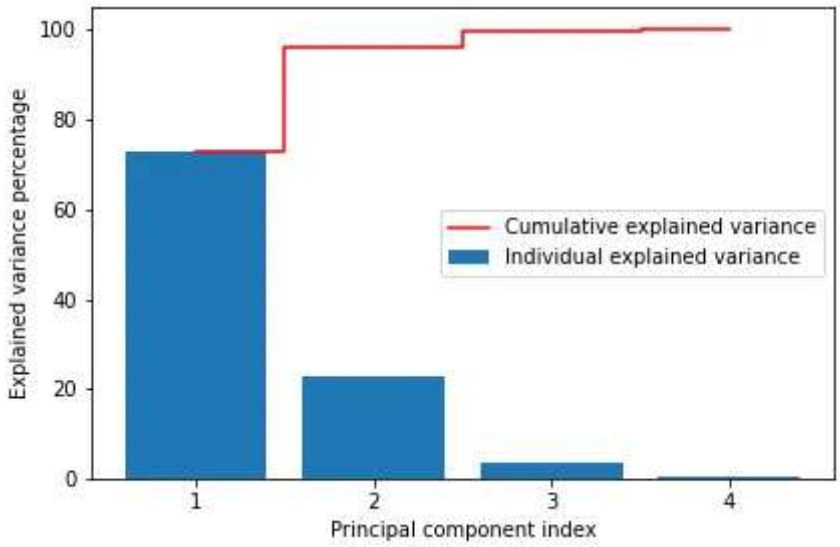
```
pca.n_components_
```

This returns an integer value which is equal to the selected number of components.

**Method 3: Plot the explained variance percentage of individual components and the percentage of total variance captured by all principal components.**

This is the most advanced and effective method that can be used to select the best number of principal components for the dataset.

In this method, we create the foll 




**The percentage of total variance captured by the principal components** (Image by author)

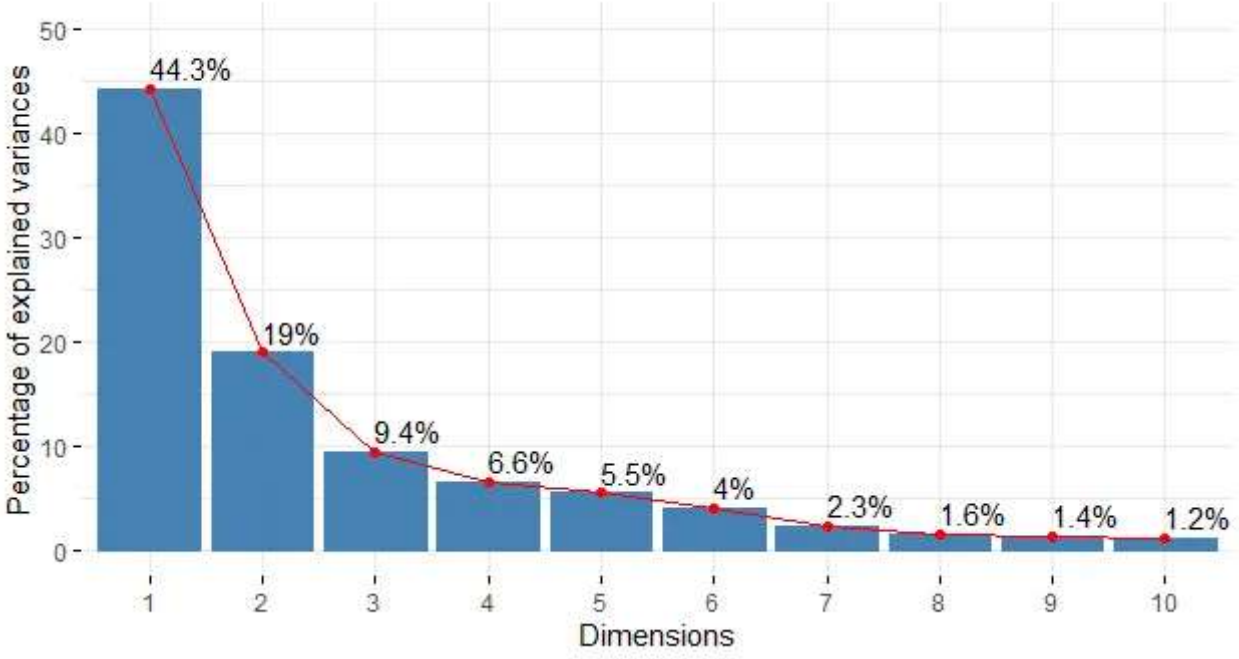
The number of bars is equal to the number of variables in the original dataset. In this plot, each bar shows the explained variance percentage of individual components and the step plot shows the cumulative explained variance percentages.

By looking at this plot, we can easily decide how many components should be kept. In this example, only the first two components capture almost all the variance in the dataset. So, we decide to select only the first two components.

To get the Python code for creating the above type of plot, please refer to the 15th Question of my "**Principal Component Analysis — 18 Questions Answered**" article.

In R, a similar type of plot can be created with the **fviz_eig()** function in the **factoextra** library by running just one line of code!

```
fviz_eig(pca_model, addlabels = TRUE,
         linecolor = "Red", ylim = c(0, 50))
```

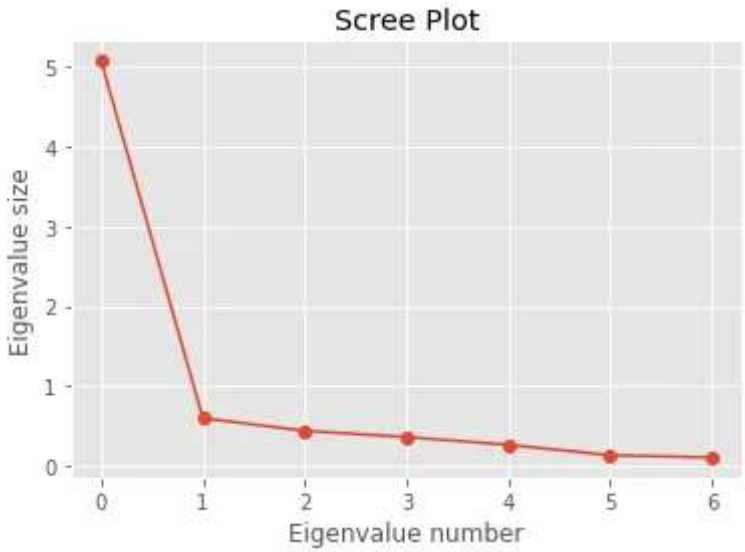



(Image by author)

Here also, each bar shows the explained variance percentage of individual components. The red line is a type of scree plot.

**Method 4: Create the scree plot.**

Another type of plot that we can create to select the best number of principal components is the **Scree Plot** which is the visual representation of eigenvalues that define the magnitude of eigenvectors (principal components).




**A scree plot** (Image by author)

In Scikit-learn PCA() class, the eigenvalues can be obtained using the following line of code.

```
pca.explained_variance_
```

To create the above plot, we can use the following code block.

```
import matplotlib.pyplot as plt
plt.style.use("ggplot")

plt.plot(pca.explained_variance_, marker='o')
plt.xlabel("Eigenvalue number")
plt.ylabel("Eigenvalue size")
plt.title("Scree Plot")
```

We select all the components up to the point where the bend occurs in the Scree Plot. In the above plot, the bend occurs at index 1. So, we decide to select the components at index 0 and index 1 (a total of two components).

#### Method 5: Follow the Kaiser's rule.

> According to the ***Kaiser's*** rule, it is recommended to keep all the components with eigenvalues greater than 1.

You can get eigenvalues using `explained_variance_` attribute of the PCA() class.

Then, you can select the components with eigenvalues greater than 1. When following this rule, it is better to combine this with the explained variance percentage plot discussed in Method 3 or scree plot discussed in Method 4. Sometimes, an eigenvalue that is slightly less than 1 (e.g. 0.95) may capture some significant amount of variance in the data. So, it is better to keep that one also if it shows relatively a higher bar in the explained variance percentage plot.

#### Method 6: Use a performance evaluation metric such as RMSE (for Regression) or Accuracy Score (for Classification).

This method can only be used if you're planning to do regression or classification tasks with the reduced (transformed) dataset after applying PCA.

By using the plot discussed in Method 3, you can select the initial number of principal components and get the reduced (transformed) dataset. Then, you build a regression or classification model and measure its performance through RMSE or Accuracy Score. Then, you slightly change the number of principal components, build the model again and measure its performance score. After repeating these steps several times, you can select the best number of principal components that actually give the best performance score.

Kindly note that the model's performance score depends on other factors too. For example, it depends on the random state of splitting train-test sets, amount of data, class imbalance, number of trees in the mode (if the model is a random forest or a similar variant), number of iterations we set during optimization, outliers and missing values in the data, etc. So, be careful when using this method!

## Summary

It is not necessary to try out all these methods at the same time. The most effective method is to use the plot created in Method 3. You can also combine the Kaiser's rule with that plot.

It is obvious that the best number of principal components depends on what we really want from PCA. If our sole intention of doing PCA is for data visualization, the best number of components is 2 or 3. If we really want to reduce the size of the dataset, the best number of principal components is much less than the number of variables in the original dataset.

When applying PCA to a dataset, one golden rule always exists.

> *Select the best number of principal components while keeping as much of the variance in the original data as possible.*

If you ignore a key component that captures a significant amount of variance in the data, the model's performance score will be reduced if you do regression, classification or other visualization tasks with the transformed dataset.

Applying PCA is not a one-time process. It is an iterative process. You need to run the PCA() function several times.

First, you have to run the PCA() function with `n_components=None` which means we keep all the components for now.

```
from sklearn.decomposition import PCA
pca_1 = PCA(n_components=None)
```

Then, we create the plot discussed in Method 3 and select the best number of principal components (called *k*). Finally, we run the PCA() function again with `n_components=k`

```
pca_2 = PCA(n_components=k)
```

This is the end of today's article. If you have any questions regarding this article, please let me know in the comment section.

Thanks for reading!

See you in the next article! As always, happy learning to everyone!

**Read next (recommended) — Written by me!**

Learn how Principal Component Analysis (PCA) works behind the scenes.

**Eigendecomposition of a Covariance Matrix with NumPy**

For Principal Component Analysis (PCA)

rukshanpramoditha.medium.com

One-stop place for your most of the questions regarding PCA.

**Principal Component Analysis — 18 Questions Answered**

One-stop place for your most of the questions regarding PCA

rukshanpramoditha.medium.com

PCA in Action for Dimensionality Reduction.

**RGB Color Image Compression Using Principal Component Analysis**

PCA in Action for Dimensionality Reduction

towardsdatascience.com

The whole ML is full of dimensionality reduction and its applications. Let's see them in action!

**11 Different Uses of Dimensionality Reduction**

The whole ML is full of dimensionality reduction and its applications. Let's see them in action!

towardsdatascience.com

Learn the difference between a parameter and a hyperparameter.

**Parameters Vs Hyperparameters: What is the difference?**

Discuss with 4 different examples

rukshanpramoditha.medium.com

**Become a member**

If you'd like, you can sign up for a membership to get full access to every story I write and I will receive a portion of your membership fee.

**Join Medium with my referral link - Rukshan Pramoditha**

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

rukshanpramoditha.medium.com

**Subscribe to my email list**

Never miss a great story again by subscribing to my email list. You'll receive every story in your inbox as soon as I hit the publish button.

**Get an email whenever Rukshan Pramoditha publishes.**

Get an email whenever Rukshan Pramoditha publishes. By signing up, you will create a Medium account if you don't...

rukshanpramoditha.medium.com

Written by Rukshan Pramoditha

2022–04–24

Machine Learning Unsupervised Learning Data Science Principal Component Dimensionality Reduction