

# Comprehensive Guide on Principal Component Analysis (PCA)

By [Isshin Inada](#)



Colab Notebook for Principal Component Analysis (PCA)

Click here to access all the Python code snippets (including code to generate the graphs) used for this guide!

## What is principal component analysis (PCA)?

Principal component analysis, or PCA, is one of a family of techniques for dimensionality reductions. For instance, we can reduce 20 features (or predictor variables) into just 2 features by applying PCA. The basic idea is to use the dependencies between the features to represent them in a lower dimensional form while trying to minimize information loss.

Dimensionality reduction is useful in the following cases:

- when we use algorithms (e.g. k-means clustering) that use distance-based metrics such as the Euclidean distance. This is because the notion of distances breaks down in high dimensions; no matter how close or distant two points are, their distance always converges to a single constant in high dimensions. This phenomenon is called the **curse of dimensionality**, and this makes distance-based algorithms useless in cases when you have a large number of features.
- when we want to compress data with minimal information loss. By representing our dataset with fewer features, we can save more memory and computation time when training our models.
- when we want to filter out noise. PCA aims to preserve the most important parts of the dataset, which means that subtle noises are more likely to be ignored.
- when we want to visualize high-dimensional data. By mapping our features into  $\mathbb{R}$ ,  $\mathbb{R}^2$  or  $\mathbb{R}^3$ , we can visualize our dataset using a simple scatter plot. This allows us to do some fascinating things like [visualize words](#)!

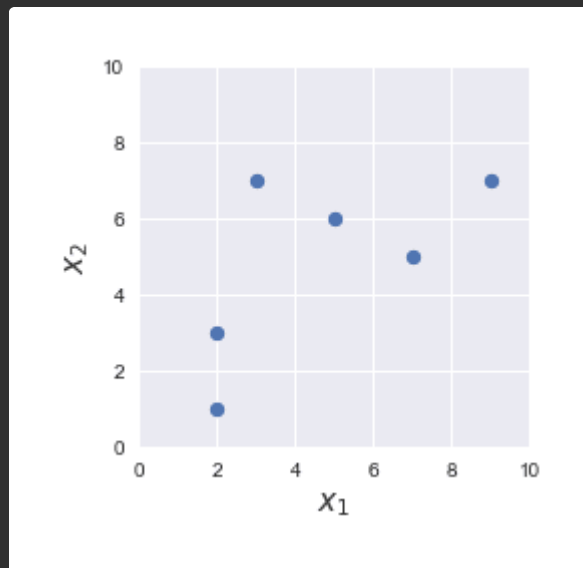
As we always do with our [comprehensive guide series](#), we will go through a simple example to get an intuitive understanding of how PCA works while also introducing some mathematical rigor. We will then apply PCA to real-world datasets to demonstrate the above use cases. Please feel free to contact me on [DISCORD](#) or [isshin@skytowner.com](mailto:isshin@skytowner.com) if you get stuck while following along - always happy to clarify concepts!

## Performing PCA manually with a simple example

Consider the following two-dimensional dataset:

	$x_1$	$x_2$
1	9	7
2	7	5
3	3	7
4	5	6
5	2	3
6	2	1

Let's begin with a quick visualization of our dataset:



Our goal is to use PCA to reduce the dimensions of our dataset from two to one, that is, from  $\mathbb{R}^2$  to  $\mathbb{R}$ .

### Standardization

As we shall see later, PCA uses the variance of each feature to extract information, which means that the units (or magnitudes) of the features will heavily affect the outcome of PCA. For instance, suppose we want to perform PCA on two variables - a person's age and weight. If the unit of weight is in grams, then the magnitude of its spread will be much larger than that of the age variable. The variance of the weight would be in the order of magnitude of 10,000 while that of age would be 10. As a consequence, PCA would focus more on extracting information from variables with higher magnitudes and ignore the other variables of less magnitudes. As you can imagine, this leads to heavily biased results.

The way to overcome this is to initially perform standardization such that all the variables are transformed to the same unitless scale. There are multiple ways to scale features as discussed in our [feature scaling](#) guide, but PCA uses **Z-score normalization**:

$$z = \frac{x - \bar{x}}{\sigma} \quad (1)$$

Where:

- $z$  is the scaled value of the feature
- $x$  is the original value of the feature
- $\bar{x}$  is the mean of all values of the feature
- $\sigma$  is the standard deviation of all values of the feature

After performing this standardization, the transformed features would each have a mean of 0 and a standard deviation of 1.

As an example, let's manually standardize our first feature  $x_1$ . We first need to compute the mean and the variance of this feature. We begin with the mean:

$$\begin{aligned} \bar{x}_1 &= \frac{1}{6}(9 + 7 + 3 + 5 + 2 + 2) \\ &\approx 4.67 \end{aligned}$$

Next, let's compute the standard deviation:

$$\begin{aligned}\sigma_1 &= \sqrt{\frac{1}{6} \sum_{i=1}^6 (x_i - \bar{x})^2} \\ &= \sqrt{\frac{1}{6} \sum_{i=1}^6 (x_i - 4.67)^2} \\ &\approx 2.62\end{aligned}$$

We can compute each scaled value of  $x_1$  like so:

$$\begin{aligned}z_1^{(i)} &= \frac{x_1^{(i)} - \bar{x}_1}{\sigma_1} \\ &= \frac{x_1^{(i)} - 4.67}{2.62}\end{aligned}$$

Where:

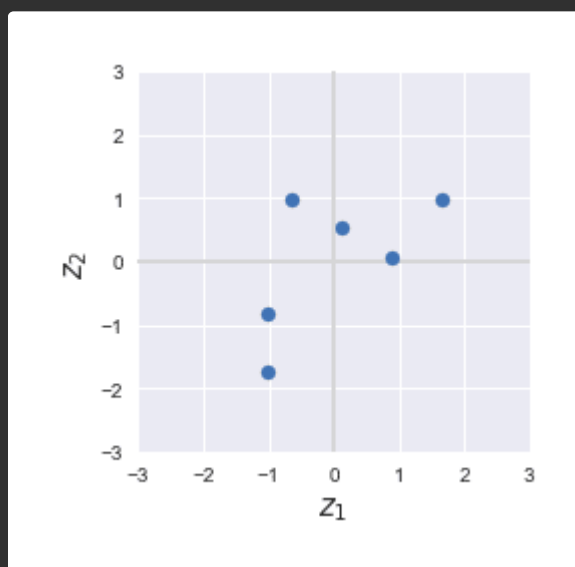
- $z_1^{(i)}$  is the scaled value of the  $i$ -th value in feature  $x_1$ .
- $x_1^{(i)}$  is the original  $i$ -th value in feature  $x_1$ .

For example, to compute the first value:

$$\begin{aligned}z_1^{(1)} &= \frac{x_1^{(1)} - 4.67}{2.62} \\ &= \frac{9 - 4.67}{2.62} \\ &\approx 1.65\end{aligned}$$

We repeat this process for the rest of the values in the feature to finally obtain the scaled feature  $z_1$ . Remember that we've only standardized  $x_1$  here - we need to repeat the entire process (starting with computing the mean and standard deviation) for each feature ( $x_2$  in this case)!

Our dataset visually looks like the following after standardization:

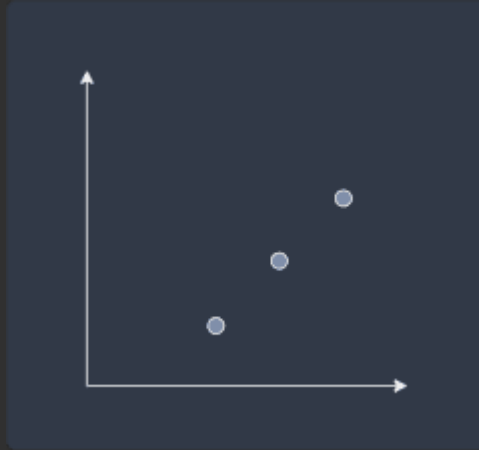


As we can see, the layout of the points still looks similar even after standardization, and they are now centered around the origin!

## Finding the principal components

### What are principal components?

The next step of PCA is to find a line (also known as **principal components**) on which to project the data points that captures the relationship between the two variables well. How well the relationship is captured is based on how much variance is preserved when the points are projected onto the line. Let's intuitively understand what this means by going over a side example. Suppose we have the following data points:



Our goal is to find a line that preserves maximal information (in the form of variance) after projection. Let's first see a line that does this horribly:



Here, we can see that the points land on the same place when they are projected onto the line. Can you see how we are losing all the information about our data points? In particular, the projected points are not spread out at all, which mathematically means that the variance of the projected points is zero. In other words, information is captured in the form of variance - the higher variance of the projected points, the more information we can capture about the variables' relationship.

In fact, this is true in other contexts of machine learning as well. If you imagine a feature having all the same values (zero variance), then that feature is useless and provides no information value when building models.

Going back to our example, let's see what a good line might like:



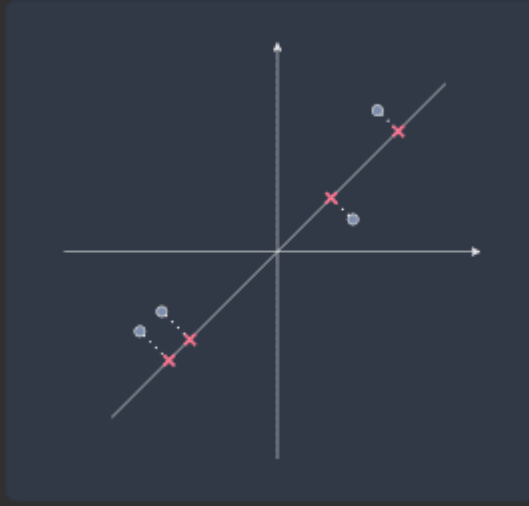
Here, can you see how the projected points are spread out unlike in the worse case? If we swap the original data points with the projected points, we are still preserving the underlying pattern by maintaining the variance of the original points. With this short side example, hopefully you now understand how PCA tries to minimize information loss by preserving the variance of the original data points as much as possible. The optimal line on which the data points are projected is known as **principal component (PC)**.

### How do we measure variance?

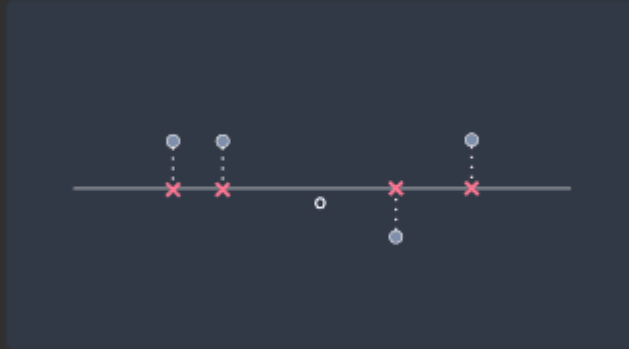
We have said that the goal of PCA is to preserve as much information as possible by maximizing the variance of the projected points. Let's make this statement mathematically precise. Straying away from our example again, suppose we had the following data points after standardization:



In PCA, we enforce a constraint that our principal component must go through the origin - you'll understand why shortly. Suppose our line and the corresponding projected points are as follows:



The objective of PCA is to maximize the variance of the projected points. To make it easier for the eyes, let's rotate the plot so that our line is horizontal:



We haven't done any transformation here - we've simply rotated the plot so that you don't have to keep tilting your head. We denote the distance between the projected data point  $\mathbf{p}^{(i)}$  to the mean of the data points  $\mu_P$  as  $\|\mathbf{p}^{(i)}\|$ . From elementary statistics, we know that the variance is computed as follows:

$$\mathbb{V}(\mathbf{P}) = \frac{1}{n} \sum_{i=1}^n (\|\mathbf{p}^{(i)}\| - \mu_P)^2 \quad (2)$$

Here,  $\mathbf{P}$  denotes the collection of projected data points. Note that  $\|\mathbf{p}^{(i)}\|$  is a scalar because it is a distance measure. Now the question is, what is the mean of the projected points  $\mu_P$  in (2)? As we will prove [later](#) [🔗](#), the projected points are also centered around the mean! Therefore, (2) now reduces down to the following:

$$\mathbb{V}(\mathbf{P}) = \frac{1}{n} \sum_{i=1}^n (\|\mathbf{p}^{(i)}\|)^2 \quad (3)$$

This is the value that PCA aims to maximize! In fact, we can actually go one step further and reduce (3) because of the following fact:

$$\operatorname{argmax}_{\|\mathbf{p}^{(i)}\|} \left( \frac{1}{n} \sum_{i=1}^n (\|\mathbf{p}^{(i)}\|)^2 \right) = \operatorname{argmax}_{\|\mathbf{p}^{(i)}\|} \left( \sum_{i=1}^n (\|\mathbf{p}^{(i)}\|)^2 \right) \quad (4)$$

The  $\operatorname{argmax}$  just means that the maximum occurs at the same values of  $\|\mathbf{p}^{(i)}\|$ . Using this relationship, we conclude that PCA maximizes the following quantity:

$$\sum_{i=1}^n (\|\mathbf{p}^{(i)}\|)^2 \quad (5)$$

Equation (5) is basically the **sum of squared distances from the origin to the projected data points**. PCA gives us all the  $\|p^{(i)}\|$  that maximizes (5), that is, after performing PCA, we will know the distance from the origin to each data point.

In the following section, we will go over how PCA optimizes (5). Up to now, the mathematics has not been rigorous - all we've done so far is performed standardization to make each variable have a mean of zero and a standard deviation of one. The challenging part actually begins here - we need to know what **variance-covariance matrices**, **eigenvalues** and **eigenvectors** are in order to understand how PCA computes the optimal line. These topics are complex enough to warrant their own guides (they're coming soon by the way 🚀), so I'll only **briefly** discuss what they are here.

### Computing the variance-covariance matrix

First, let's go over **variance-covariance matrices**, which fortunately are much easier to explain than eigenvalues and eigenvectors. Suppose we have two random variables  $X_1$  and  $X_2$ . The **variance-covariance matrix**  $\Sigma$  of  $X_1$  and  $X_2$  is defined as follows:

$$\Sigma = \begin{pmatrix} \mathbb{V}(X_1) & \text{cov}(X_1, X_2) \\ \text{cov}(X_2, X_1) & \mathbb{V}(X_2) \end{pmatrix}$$

Where:

- $X_1$  and  $X_2$  are random variables
- $\mathbb{V}(X_1)$  is the variance of  $X_1$
- $\text{cov}(X_1, X_2)$  is the covariance of  $X_1$  and  $X_2$ .

Basically, the **variance-covariance matrix** captures the relationship between random variables since the matrix contains the covariance of each pair of random variables.

Let's compute the **variance-covariance matrix** for our example:

$$\Sigma = \begin{pmatrix} \mathbb{V}(z_1) & \text{cov}(z_1, z_2) \\ \text{cov}(z_2, z_1) & \mathbb{V}(z_2) \end{pmatrix}$$

Fortunately, because we have performed standardization in the previous step, we already know what the variance of our two features is:

$$\begin{aligned} \mathbb{V}(z_1) &= 1 \\ \mathbb{V}(z_2) &= 1 \end{aligned}$$

Next, we need to compute the covariance of  $x_1$  and  $x_2$ :

$$\text{cov}(z_1, z_2) = \frac{1}{6} \sum_{i=1}^6 (z_1^{(i)} - \mu_1)(z_2^{(i)} - \mu_2)$$

Where:

- $z_1^{(i)}$  is the  $i$ -th data of feature  $x_1$
- $\mu_1$  and  $\mu_2$  are the mean of  $x_1$  and  $x_2$  respectively

Once again, because we have performed standardization, we know that  $\mu_1 = 0$  and  $\mu_2 = 0$ . Therefore, our equation reduces down to:

$$\text{cov}(z_1, z_2) = \frac{1}{6} \sum_{i=1}^6 (z_1^{(i)} \cdot z_2^{(i)})$$

$$\approx 0.63$$

From the property of covariances, we know that  $\text{cov}(z_1, z_2) = \text{cov}(z_2, z_1)$ , so we don't need to calculate the latter.

With the variance and covariance computed, we now have everything we need for our variance-covariance matrix:

$$\Sigma = \begin{pmatrix} 1 & 0.63 \\ 0.63 & 1 \end{pmatrix}$$

Great, with the variance-covariance matrix now well defined, let's now move on to eigenvalues and eigenvectors.

#### NOTE

To be technically precise, the variance-covariance matrix that we computed here is actually the correlation matrix because we have standardized our data. Therefore,  $\text{cov}(z_1, z_2)$  measures the correlation instead, and the value is bound between  $-1$  and  $1$ .

### Eigenvalues and eigenvectors

Unfortunately, the concept of **eigenvalues** and **eigenvectors** is what makes PCA so difficult to understand, and the main issue is that they are abstract topics and require in-depth knowledge about linear algebra to fully grasp. I'll write a separate comprehensive guide about them very soon - join our [DISCORD](#) to be notified when I do - but for now, do your best to just ignore the burning question of "why?" and accept that the method works 😊.

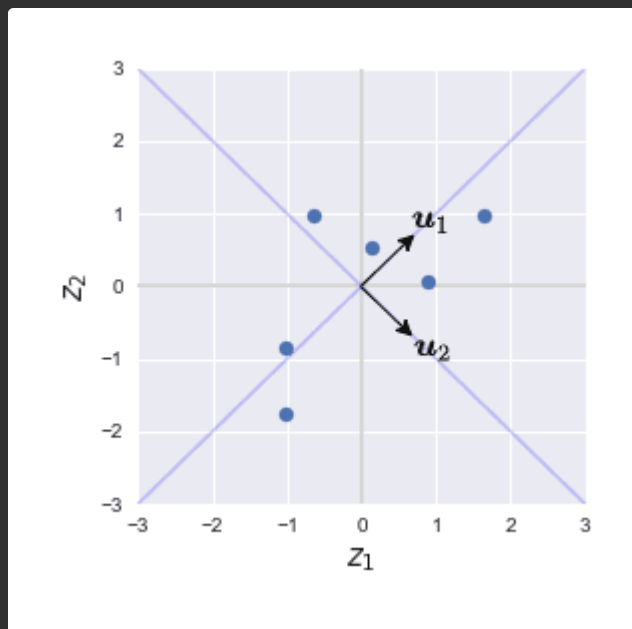
### Computing eigenvalues and eigenvectors of the variance-covariance matrix

Recall that the goal of PCA is to find the line that maximizes the variance of the projected points. **It turns out that the eigenvectors of the variance-covariance matrix give us the optimal lines, or so-called principal components.** I'm not going to go over the methodology behind computing eigenvalues and eigenvectors, but instead, give you the results directly. Note that I'm actually using Python's NumPy to compute these eigenvectors - please check out my Colab Notebook ([COLAB](#)) for the code. The computed eigenvectors are as follows:

$$u_1 = \begin{pmatrix} 0.71 \\ 0.71 \end{pmatrix} \quad u_2 = \begin{pmatrix} 0.71 \\ -0.71 \end{pmatrix}$$

Let's plot these eigenvectors:





Here, note the following:

- $u_1$  and  $u_2$  are the **eigenvectors** of the **variance-covariance matrix**, and they actually maximize the variance of the projected data points.
- $u_1$  and  $u_2$  are called **principal components (PCs)**.
- the eigenvectors are always **orthogonal** (perpendicular) to each other.
- $u_1$  and  $u_2$  are **unit-vectors**, that is, they have length one.
- the blue lines represent the lines onto which the data points will be projected

#### NOTE

If you standardize the data as we have done, then you will always get the same pair of eigenvectors as above regardless of your initial two-dimensional dataset. These eigenvectors go through the origin at 45 degrees.

From the above plot, we can see that we have two principal components. We now need to pick which principal component we want to project our data points to. The decision is never made randomly since the individual principal components retain varying amounts of information of the original data points. In order to pick the best principal components that preserve the most information of the original data points, we check the corresponding **eigenvalues** of the **eigenvectors**. For each **eigenvector**, there exists a corresponding **eigenvalue**, and hence in our case, we have 2 **eigenvalues** to check. These **eigenvalues** are extremely important because they tell us how much information is preserved - the higher the **eigenvalue**, the more information that is preserved. In this case, the **eigenvalues** of  $u_1$  and  $u_2$  are computed as follows:

$$\begin{aligned}\lambda_1 &= 1.63 \\ \lambda_2 &= 0.37\end{aligned}$$

Where:

- $\lambda_1$  is the eigenvalue of the eigenvector  $u_1$
- $\lambda_2$  is the eigenvalue of the eigenvector  $u_2$

We can see that the **eigenvalue** of  $u_1$  is greater than that of  $u_2$ . This means that the principal component  $u_1$  preserves more information than  $u_2$ , and hence we should pick  $u_1$  to project our data points onto. The **eigenvector** with the highest **eigenvalue** is called the **1st principal component (PC1)**, and the **eigenvector** with the second-highest **eigenvalue** is called the **2nd principal component (PC2)**, and so on. In this case,  $u_1$  is PC1 and  $u_2$  is PC2.

## Computing the explained variance ratio

We have just discussed how **eigenvalues** give us an idea of how much information their corresponding **eigenvector** contains if we were to project our data points onto them. Instead of comparing **eigenvalues** directly, we often compute a simple measure called **explained variance ratio** like so:

$$r_i = \frac{\lambda_i}{\sum_{j=1}^m \lambda_j}$$

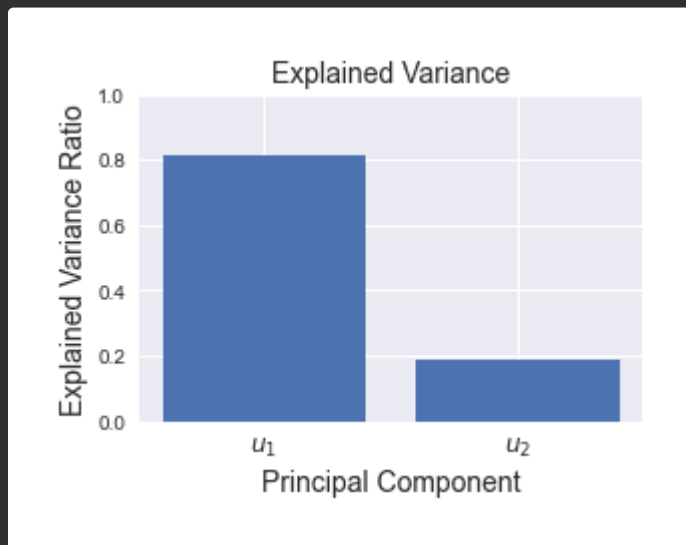
Where:

- $r_i$  is the explained variance of the  $i$ -th eigenvector
- $\lambda_i$  is the eigenvalue of the  $i$ -th eigenvector
- $m$  is the number of eigenvectors

In our example, we only have two eigenvalues ( $m = 2$ ), and so we just need to compute two explained variance ratios like so:

$$r_1 = \frac{\lambda_1}{\lambda_1 + \lambda_2} = \frac{1.63}{1.63 + 0.37} \approx 0.81$$
$$r_2 = \frac{\lambda_2}{\lambda_1 + \lambda_2} = \frac{0.37}{1.63 + 0.37} \approx 0.19$$

We usually plot a simple bar graph of the explained variance ratios:



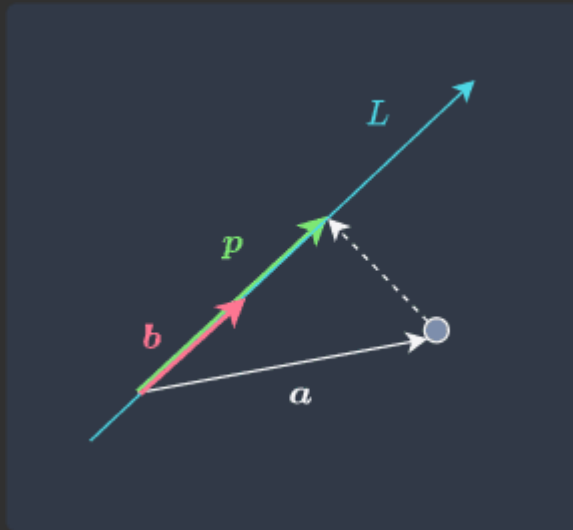
As you can see, the **explained variance ratio** for  $u_1$  is much greater than  $u_2$ , which is what we should expect since the **eigenvalue** of  $u_1$  is much larger than  $u_2$ . The reason why **explained variance ratio** is typically computed instead of directly comparing **eigenvalues** is that the value of the **explained variance ratio** of a principal component can be interpreted as how much of the information is preserved if we were to project our points onto that principal component. For instance, if we project our dataset onto the PC1 ( $u_1$ ), then we would be able to preserve roughly 81% of the original information - which is quite good!

## Projecting data points onto principal components

So far, we have computed the principal components and their **explained variance ratios**. Since the **explained variance ratio** of  $u_1$  is greater than that of  $u_2$ , we will project our points to  $u_1$  instead of  $u_2$ . We now need to project our data points onto the principal components to perform dimensionality reduction. Note that we are not specifically after the coordinates of the projected points, but instead we are after the distance from the origin to each projected point, that is,  $\|p^{(i)}\|$ .

## Computing projection lengths

In order to obtain the distance from the origin to each projected point, we can turn to linear algebra. Let's forget our example for a quick moment, and review the concept of projections in linear algebra. Consider the following diagram:

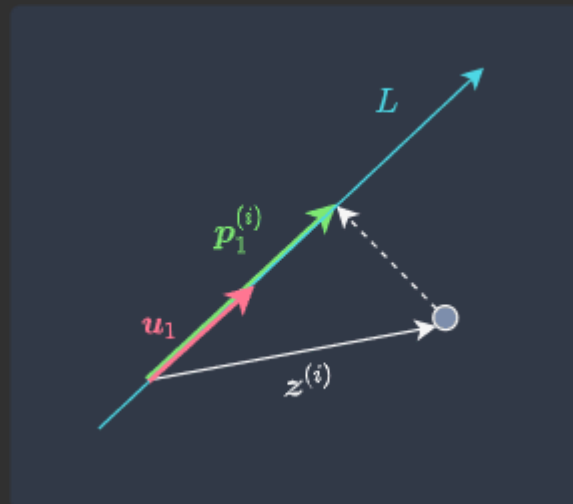


Here,  $p$  is the vector that results from an orthogonal projection from point  $a$  to the line  $L$ . The vector  $b$  is any vector that points in the same direction as the line  $L$ . The dot product of two vectors  $a$  and  $b$  returns the following:

$$a \cdot b = \|p\| \|b\| \quad (6)$$

Where  $\|p\|$  and  $\|b\|$  are the lengths of vectors  $p$  and  $b$  respectively.

Let's now change the notations to align with our example:



Here,

- $u_1$  is our PC1 - a unit-eigenvector of length one
- $z^{(i)}$  is the  $i$ -th scaled observation
- $p_1^{(i)}$  is the vector resulting from the orthogonal projection on PC1

Using the logic of (6), the dot product of  $u_1$  and  $z^{(i)}$  is:

$$u_1 \cdot z^{(i)} = \|p_1^{(i)}\| \|u_1\|$$

Since  $u_1$  is a unit vector of length one, this formula reduces down to the following:

$$\mathbf{u}_1 \cdot \mathbf{z}^{(i)} = \|\mathbf{p}_1^{(i)}\| \quad (7)$$

This means that in order to obtain the length from the origin to the projected point, we just need to take the dot product between our eigenvector (PC1) and each data point - simple!

Going back to our example, let's try computing  $\|\mathbf{p}_1^{(1)}\|$  for the right-most data point  $\mathbf{z}^{(1)}$ , which is represented by the following coordinate vector:

$$\mathbf{z}^{(1)} = \begin{pmatrix} 1.65 \\ 0.99 \end{pmatrix}$$

Using (7), let's compute  $\|\mathbf{p}_1^{(1)}\|$ :

$$\begin{aligned} \|\mathbf{p}_1^{(1)}\| &= \mathbf{u}_1 \cdot \mathbf{z}^{(1)} \\ &= \begin{pmatrix} 0.71 \\ 0.71 \end{pmatrix} \cdot \begin{pmatrix} 1.65 \\ 0.99 \end{pmatrix} \\ &\approx 1.87 \end{aligned}$$

This means that the length from the origin to the first projected point is 1.87.

Now, we need to perform the same step for each of the other data points. Mathematically, we can neatly represent this operation with the help of matrices. Let's pack our eigenvectors into a single matrix, which we call  $\mathbf{W}$ :

$$\mathbf{W} = \begin{pmatrix} | & | \\ \mathbf{u}_1 & \mathbf{u}_2 \\ | & | \end{pmatrix}$$

Here, the eigenvectors are stacked horizontally. In our case, the content of  $\mathbf{W}$  would be:

$$\mathbf{W} = \begin{pmatrix} 0.71 & 0.71 \\ 0.71 & -0.71 \end{pmatrix}$$

Similarly, let's also pack our data points into a single matrix  $\mathbf{Z}$ :

$$\mathbf{Z} = \begin{pmatrix} - & \mathbf{z}^{(1)} & - \\ - & \mathbf{z}^{(2)} & - \\ - & \vdots & - \\ - & \mathbf{z}^{(6)} & - \end{pmatrix}$$

In our example, each row of  $\mathbf{Z}$  represents a data point that lives in  $\mathbb{R}^2$ . This means that  $\mathbf{Z} \in \mathbb{R}^{6 \times 2}$ , that is,  $\mathbf{Z}$  has 6 rows and 2 columns. Let's now compute the product of  $\mathbf{Z}$  and  $\mathbf{W}$ , which we will call  $\mathbf{X}'$ :

$$\begin{aligned}
\mathbf{X}' &= \mathbf{Z}\mathbf{W} \\
&= \begin{pmatrix} - & \mathbf{z}^{(1)} & - \\ - & \mathbf{z}^{(2)} & - \\ - & \vdots & - \\ - & \mathbf{z}^{(6)} & - \end{pmatrix} \begin{pmatrix} | & | \\ \mathbf{u}_1 & \mathbf{u}_2 \\ | & | \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{z}^{(1)} \cdot \mathbf{u}_1 & \mathbf{z}^{(1)} \cdot \mathbf{u}_2 \\ \mathbf{z}^{(2)} \cdot \mathbf{u}_1 & \mathbf{z}^{(2)} \cdot \mathbf{u}_2 \\ \vdots & \vdots \\ \mathbf{z}^{(6)} \cdot \mathbf{u}_1 & \mathbf{z}^{(6)} \cdot \mathbf{u}_2 \end{pmatrix} \\
&= \begin{pmatrix} \|\mathbf{p}_1^{(1)}\| & \|\mathbf{p}_2^{(1)}\| \\ \|\mathbf{p}_1^{(2)}\| & \|\mathbf{p}_2^{(2)}\| \\ \vdots & \vdots \\ \|\mathbf{p}_1^{(6)}\| & \|\mathbf{p}_2^{(6)}\| \end{pmatrix}
\end{aligned}$$

Since  $\mathbf{Z} \in \mathbb{R}^{6 \times 2}$  and  $\mathbf{W} \in \mathbb{R}^{2 \times 2}$ , the resulting dimensions of  $\mathbf{X}'$  would be  $\mathbb{R}^{6 \times 2}$ . Here,  $\mathbf{X}'$  holds  $\|\mathbf{p}_1^{(i)}\|$  and  $\|\mathbf{p}_2^{(i)}\|$ , which represent the lengths from the origin to the projected data points on both PC1 (first column) and PC2 (second column) respectively. Since we know that PC1 preserves much more information than PC2, we can choose to ignore PC2 (second column). Computationally, this is useful because we can actually set  $\mathbf{W}$  to hold just the first principal component, thereby saving computation time and memory.

Great, we have now managed to obtain a general formula to compute the length from the origin to each projected data point for both principal components!

Let's now compute  $\mathbf{X}'$  in our example, which simply requires computing the dot product of every pair of  $\mathbf{z}^{(i)}$  and the two eigenvectors  $\mathbf{u}_1$  and  $\mathbf{u}_2$ . Here are the results:

$$\mathbf{X}' = \begin{pmatrix} 1.87 & -0.47 \\ 0.68 & -0.57 \\ 0.25 & 1.15 \\ 0.47 & 0.29 \\ -1.31 & 0.13 \\ -1.95 & -0.52 \end{pmatrix}$$

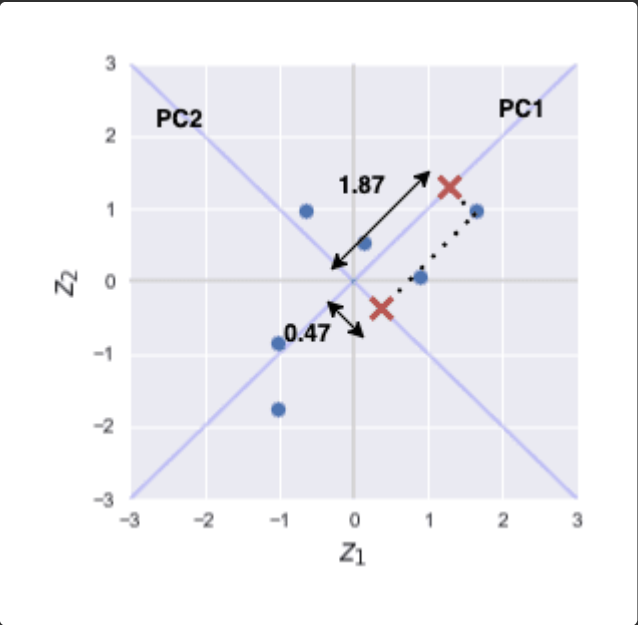
Let's summarize our final results in a table:

	$x'_1$ (PC1)	$x'_2$ (PC2)
1	1.87	-0.47
2	0.68	-0.57
3	0.25	1.15
4	0.47	0.29
5	-1.31	0.13
6	-1.95	-0.52

Here:

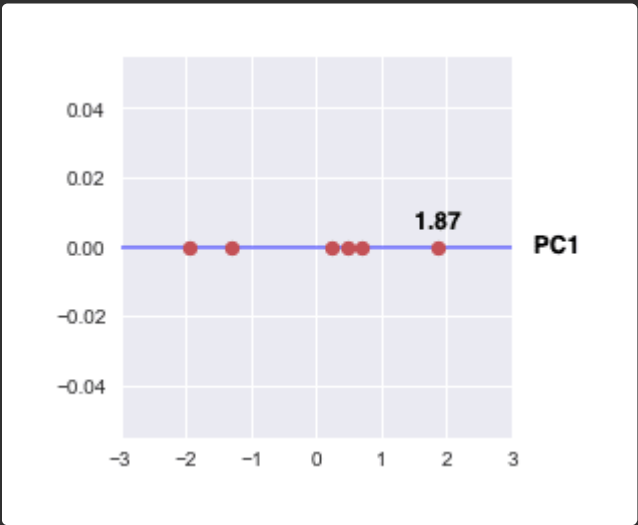
- $x'_1$  represents the length from the origin to the projected point on PC1 ( $\|p_1^{(i)}\|$ )
- $x'_2$  represents the length from the origin to the projected point on PC2 ( $\|p_2^{(i)}\|$ )

For example, the following diagram illustrates what the values represent for the first data point:



**Projecting onto PC1**

We now select PC1 to project our data points onto. We plot  $x'_1$  on a number line:



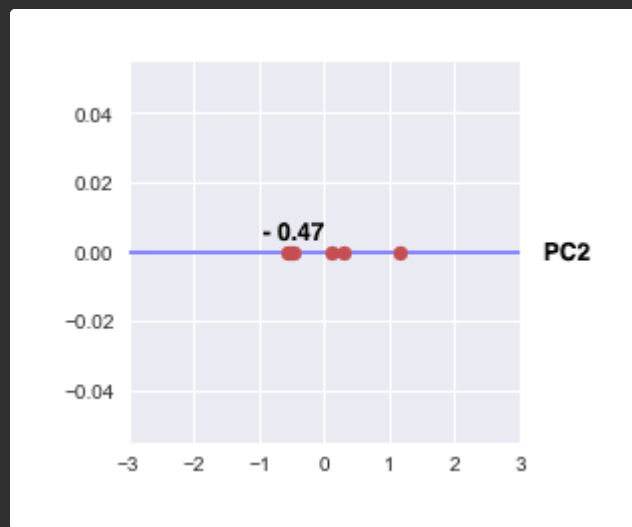
These one-dimensional values retain the most amount of information (captured through variance) of the original two-dimensional data points! The final transformed data points in tabular form are as follows:

	$x'_1$ (PC1)
1	1.87
2	0.68
3	0.25
4	0.47
5	-1.31
6	-1.95

We're actually now done with our task of performing dimensionality reduction! The rest of the guide will cover other aspects of PCA in more detail.

### Projecting onto PC2

We could also perform dimensionality reduction by selecting PC2 instead of PC1:

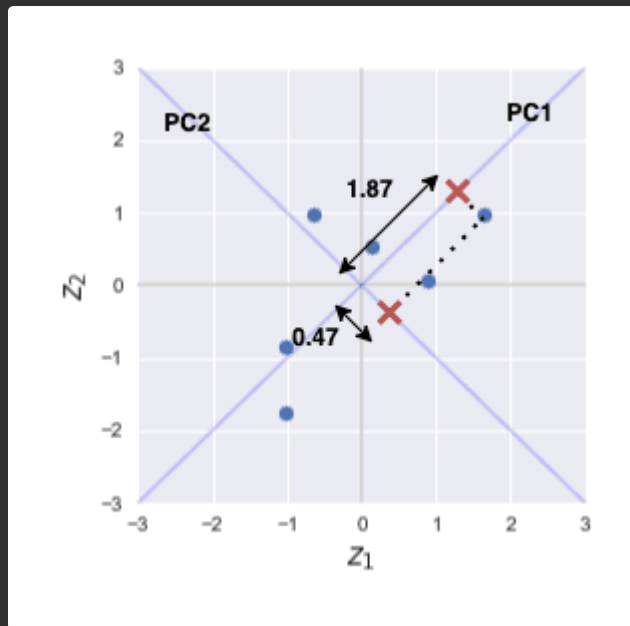


Notice how, compared to PC1, the variance of the projected points on PC2 is low. This means that projecting onto PC1 is much more desirable than onto PC2 as PC1 preserves more information with a smaller variance loss. This is to be expected because the **explained variance ratio** of PC1 is much greater than PC2 as discussed in the [previous section](#) [↗](#).

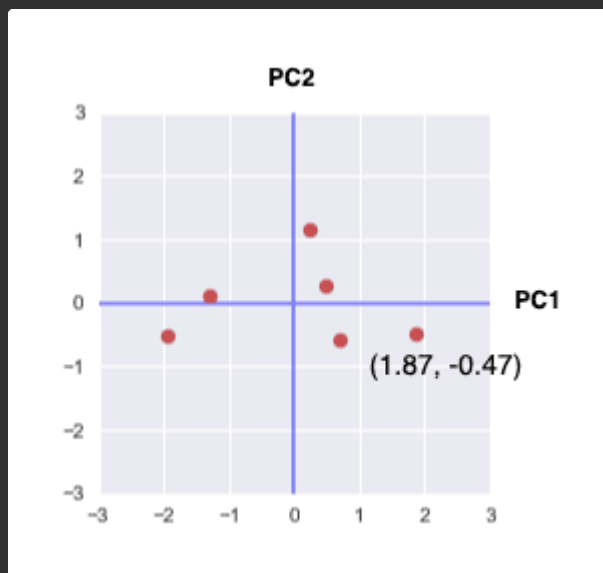
### Projecting onto both PC1 and PC2

We have just looked at how we can perform dimensionality reduction by projecting our data points onto a single principal component. We could, however, project our data points onto both the principal components, in which case, the number of dimensions of the resulting projected data points would still remain as 2.

Recall that the projected point on both PC1 and PC2 for the first data point was as follows:



If we select both PC1 and PC2 to perform projection, the transformed points would be as follows:




Here, we have rotated our previous plot 45 degrees to make PC1 and PC2 the axes line of a new coordinate system! The transformed coordinate of the first data point in this new coordinate system is  $(1.87, -0.47)$ .

### Interpreting eigenvectors as weights of features

Recall that in order to compute the new coordinates, we took the dot product of an eigenvector  $\mathbf{u}$  and the standardized data point  $\mathbf{z}^{(i)}$ . For instance, to compute the new coordinate of the  $i$ -th data point with PC1 as the axis line:

$$\begin{aligned} \|\mathbf{p}_1^{(i)}\| &= \mathbf{u}_1 \cdot \mathbf{z}^{(i)} \\ &= u_1^{(1)} z_1^{(i)} + u_1^{(2)} z_2^{(i)} \end{aligned}$$



Here,  $u_1^{(1)} \in \mathbb{R}$  and  $u_1^{(2)} \in \mathbb{R}$  are the first and second elements of the eigenvector  $\mathbf{u}_1$ . In linear algebra, we call the above a **linear combination** of variables and you can interpret  $u_1^{(1)}$  and  $u_1^{(2)}$  as the weights (or importance) assigned to each dimension (or feature) of the data point  $\mathbf{z}^{(i)}$ . For instance, if the value of  $u_1^{(2)}$  is high, then this means that the second feature of  $\mathbf{z}^{(i)}$  has a large impact on the resulting projection. In our example,  $u_1^{(1)} = u_1^{(2)} = 0.71$ , so the weighting of each feature of  $\mathbf{z}^{(i)}$  is the same. However, as we shall see in the latter section on [performing PCA on the Iris dataset](#) , these weights help us to make sense of the meaning behind principal components.

## Reconstructing the original data points

### Mathematical theory

Recall that we have performed the following transformation to obtain the length from the origin to the projected point for the principal components:

$$\mathbf{X}' = \mathbf{Z}\mathbf{W}$$

In order to obtain the original standardized dataset  $\mathbf{Z}$  back, we just need to rewrite the above as follows:

$$\mathbf{Z} = \mathbf{X}'\mathbf{W}^{-1} \quad (8)$$

Where  $\mathbf{W}^{-1}$  is the inverse matrix of  $\mathbf{W}$ . Interestingly,  $\mathbf{W}$  here is an **orthogonal matrix**, which is basically a matrix where:

- each vector (represented by columns of the matrix) is perpendicular to each other
- every vector is a unit-vector of length one.

Recall that the columns of  $\mathbf{W}$  are made up of **eigenvectors**:

$$\mathbf{W} = \begin{pmatrix} | & | \\ \mathbf{u}_1 & \mathbf{u}_2 \\ | & | \end{pmatrix}$$

From the visual plot of the two **eigenvectors**, we know that they are perpendicular to each other. Moreover, we have said that the **eigenvectors** are already converted into a unit-vector such that their lengths are one. Since both conditions are satisfied,  $\mathbf{W}$  is considered as an **orthogonal matrix**. The reason this is important to us is that, **orthogonal matrices** exhibit the following property:

$$\mathbf{W}^{-1} = \mathbf{W}^T$$

This means that the inverse of  $\mathbf{W}$  is equal to the transpose of  $\mathbf{W}$ . With this, we can rewrite (8) as follows:

$$\mathbf{Z} = \mathbf{X}'\mathbf{W}^T \quad (9)$$

This reformulation is useful because computing the transpose, which involves just swapping the rows and columns, is much easier than computing the inverse of a matrix.

### Reconstructing the original data points for our example

In this section, in order to demonstrate information loss that happens after PCA, we will go over what the reconstructed original points would look like after the following two scenarios:

- selecting all principal components (using PC1 and PC2 in our case)
- reducing the dimension by only using a subset of principal components (using PC1 and ignoring PC2 in our case)

### Using both PC1 and PC2

We have said that there is no information loss in the first scenario in which we select all the principal components. This would mean that we will be able to exactly reconstruct the original data points using the transformed points.

Let's demonstrate this using our example. Recall that our transformed data points after PCA was as follows:

$$\mathbf{X}' = \begin{pmatrix} 1.87 & -0.47 \\ 0.68 & -0.57 \\ 0.25 & 1.15 \\ 0.47 & 0.29 \\ -1.31 & 0.13 \\ -1.95 & -0.52 \end{pmatrix}$$

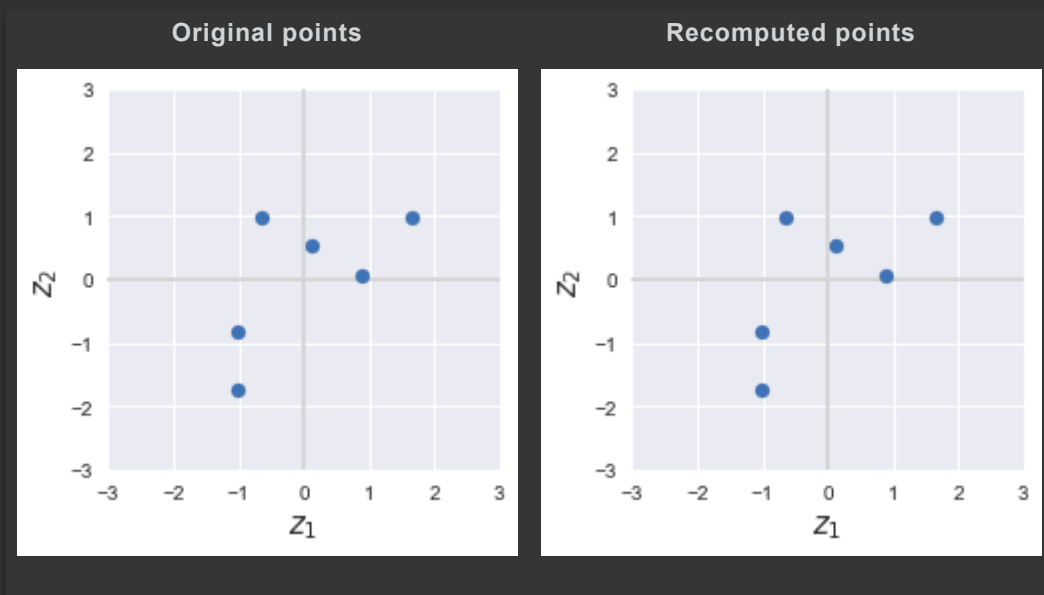
Remember that  $\mathbf{W}$  is a matrix where the columns are the eigenvectors. The transpose of  $\mathbf{W}$  therefore has rows as the eigenvectors:

$$\begin{aligned} \mathbf{W}^T &= \begin{pmatrix} - & \mathbf{u}_1 & - \\ - & \mathbf{u}_2 & - \end{pmatrix} \\ &= \begin{pmatrix} 0.71 & 0.71 \\ 0.71 & -0.71 \end{pmatrix} \end{aligned}$$

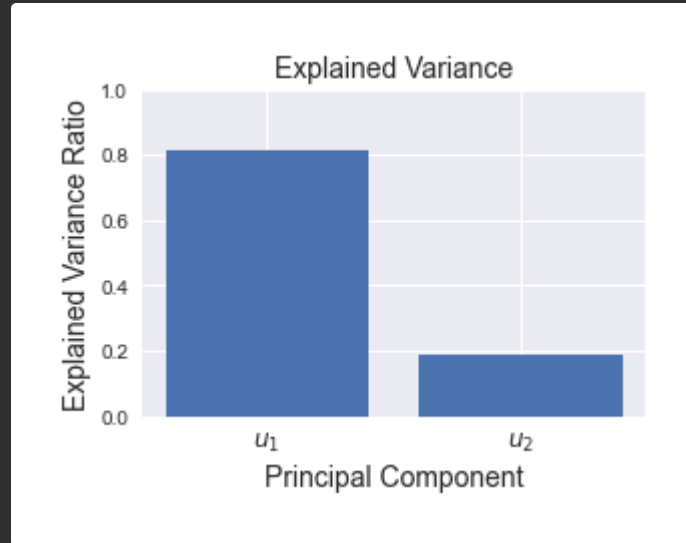
Using equation (9), we can obtain the original standardized data points:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X}'\mathbf{W}^T \\ &\approx \begin{pmatrix} 1.87 & -0.47 \\ 0.68 & -0.57 \\ 0.25 & 1.15 \\ 0.47 & 0.29 \\ -1.31 & 0.13 \\ -1.95 & -0.52 \end{pmatrix} \end{aligned}$$

Each row of  $\mathbf{Z}$  represents the coordinate of the original standardized data points. Let's plot them to confirm we indeed return to the data points before the projection:



Great, the original and recomputed points are exactly identical! We can actually explain this behavior by referring to the explained variance ratio. We show the explained variance ratio plot here again for your reference:



As a reminder, we can interpret the **explained variance ratio** of a principal component as the percentage of information that the principal component preserves. So for example, the first principal component captures roughly 80% of the information contained in the original data point. The **explained variance ratios** will always add up to one, and so if we select both PC1 and PC2, then the total **explained variance ratio** would equal 100%, which means that there is no information loss!

#### NOTE

In the context of machine learning, PCA is more often used for dimensionality reduction rather than remapping the data points in a new coordinate system as we have done in this section.

#### Using only PC1

Let us now look at the second scenario where we select only PC1. Since we ignore PC2, we can discard the second column of  $\mathbf{X}'$ , so  $\mathbf{X}'$  would look like the following:

$$\mathbf{X}' = \begin{pmatrix} 1.87 \\ 0.68 \\ 0.25 \\ 0.47 \\ -1.31 \\ -1.95 \end{pmatrix}$$

Similarly, we can ignore the second eigenvector of  $\mathbf{W}^T$ , which represents PC2. Therefore  $\mathbf{W}^T$  would contain only the first eigenvector:

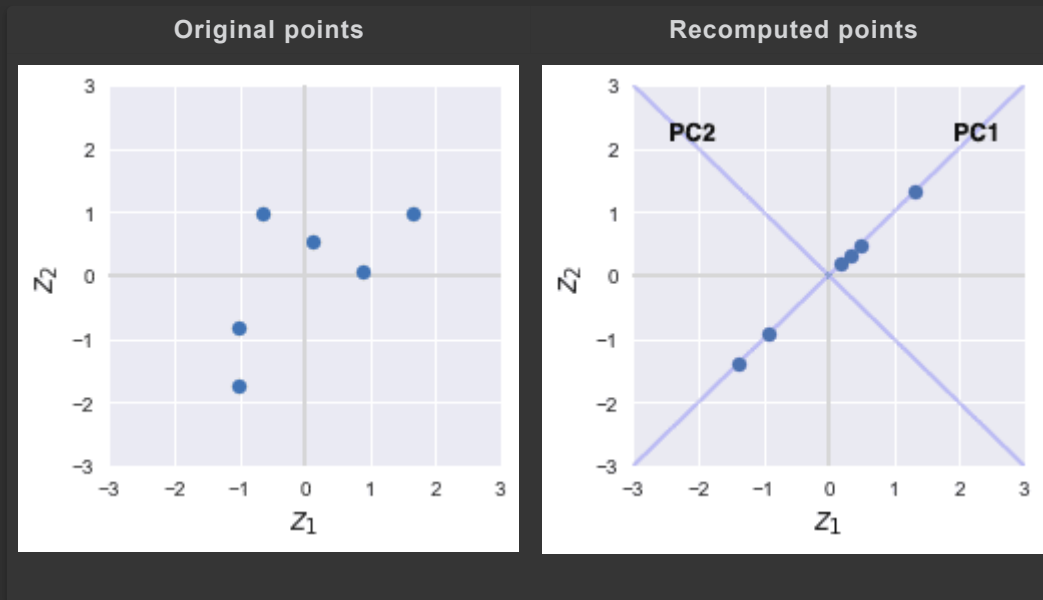
$$\begin{aligned} \mathbf{W}^T &= (- \quad \mathbf{u}_1 \quad -) \\ &= (0.71 \quad 0.71) \end{aligned}$$

Here,  $\mathbf{X}' \in \mathbb{R}^{6 \times 1}$  and  $\mathbf{W}^T \in \mathbb{R}^{1 \times 2}$  so  $\mathbf{Z} \in \mathbb{R}^{6 \times 2}$ , as we see below:

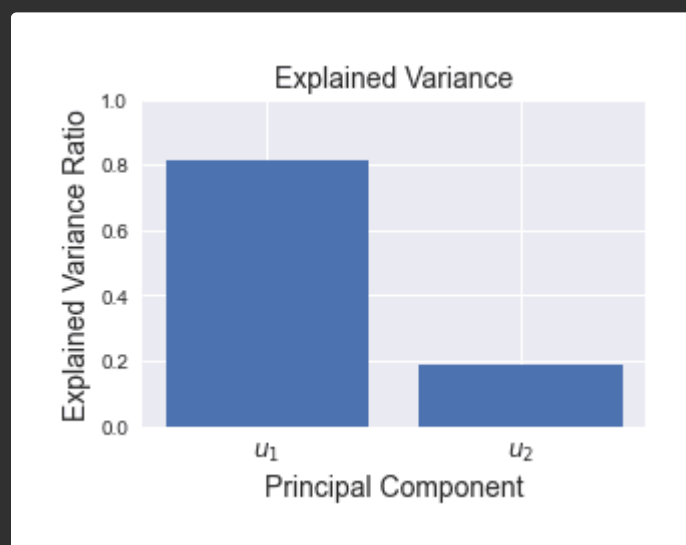
$$\mathbf{Z} = \mathbf{X}'\mathbf{W}^T$$

$$\approx \begin{pmatrix} 1.32 & 1.32 \\ 0.48 & 0.48 \\ 0.18 & 0.18 \\ 0.33 & 0.33 \\ -0.93 & -0.93 \\ -1.38 & -1.38 \end{pmatrix}$$

Let's plot  $\mathbf{Z}$  to visualize the retrieved data points:



Can you see how the recomputed data points have retained their variation along PC1 but now ignores the variation along PC2? It is impossible to reconstruct the original data points if we only select a subset of principal components because some information will be inevitably lost during dimensionality reduction. In fact, we can quantify how much information is lost using the concept of **explained variance ratio** again:



By only using PC1, we preserve 80% of the information contained in the original data points!

## NOTE

In order to retrieve the original data points back before standardization, we can simply rearrange formula (1) for standardization to make  $x$  the subject:

$$x_j^{(i)} = z_j^{(i)} \sigma_j + \bar{x}_j$$

Where:

- $x_j^{(i)}$  is the  $i$ -th original data point in feature  $j$
- $z_j^{(i)}$  is the  $i$ -th scaled data point in feature  $j$
- $\sigma_j$  is the standard deviation of the original values in feature  $j$
- $\bar{x}_j$  is the mean of the original values in feature  $j$

## Supplementary information about PCA

### Mathematical proof that the projected points are centered around the origin

Recall that the variance of the projected points  $\mathbf{P}$  is computed like so:

$$\mathbb{V}(\mathbf{P}) = \frac{1}{n} \sum_{i=1}^n (\|\mathbf{p}^{(i)}\| - \mu_P)^2$$

Where:

- $n$  is the number of data points
- $\|\mathbf{p}^{(i)}\|$  is the length from the origin to the projected point  $\mathbf{p}^{(i)}$
- $\mu_P$  is the mean of the projected points

We have claimed in the [previous section](#) [↗](#) that, just like the standardized points, the projected points are also centered around the origin. Therefore, the above equation can be reduced to the following:

$$\mathbb{V}(\mathbf{P}) = \frac{1}{n} \sum_{i=1}^n (\|\mathbf{p}^{(i)}\|)^2$$

In this section, we will prove this claim.

Firstly, we've standardized our dataset in the first step such that the mean of every feature is zero:

$$\frac{1}{n} \sum_{i=1}^n z_j^{(i)} = 0 \quad \Rightarrow \quad \sum_{i=1}^n z_j^{(i)} = 0 \quad (10)$$

Using (10), let us prove that the mean vector  $\mu_P$  of the projected points  $\mathbf{p}^{(i)}$  is actually just the origin:

Continue reading the full version at [here](#) for free without registration!