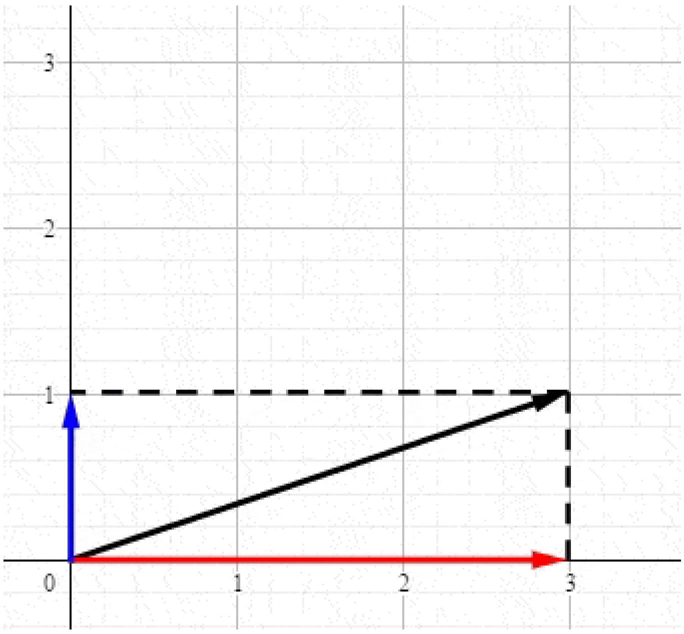
You Don't Know SVD (Singular Value Decomposition)

Truly Understanding SVD — The Intuitive Core Idea

Hussein Abdullatif Feb 21, 2019 · 6 min read ★

Back in elementary mechanics, you learned that any force vector can be decomposed into its components along the *x* and *y* axes:



> **Congratulations.** Now you know what singular value decomposition is.

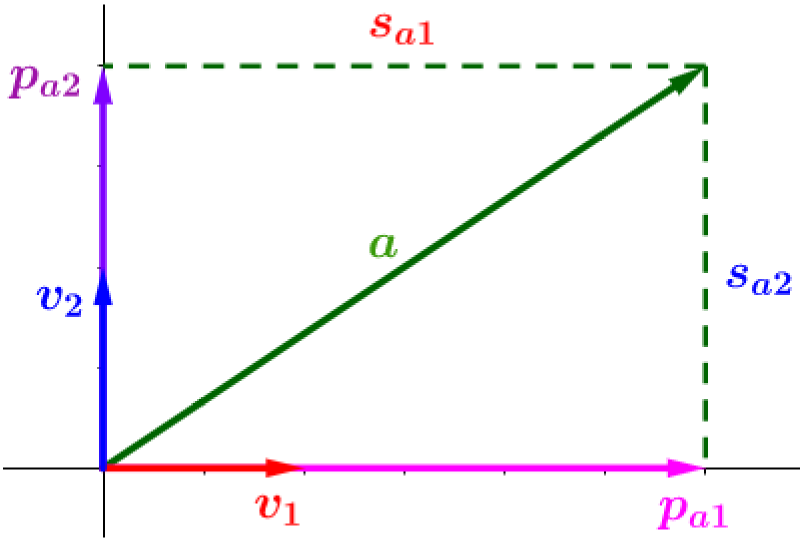
For it's disappointing that almost every tutorial of SVD **makes it more complicated than necessary, when the core idea is very simple.**

decided it may need a more deluxe name.

Let's see how this is the case.

. . .

When the vector (***a***) is decomposed, we get 3 pieces of information:



1. The **directions** of projection — the **unit** vectors (v_1 and v_2) **representing the directions** onto which we project (decompose). In the above they're the *x* and *y* axes, but can be any other orthogonal axes.

2. The **lengths** of projection (the **line segments** s_{a1} and s_{a2}) — which tell us how much of the vector is **contained** in each direction of projection (more of vector ***a*** is leaning on the direction v_1 than it is on v_2, hence $s_{a1}>s_{a2}$).

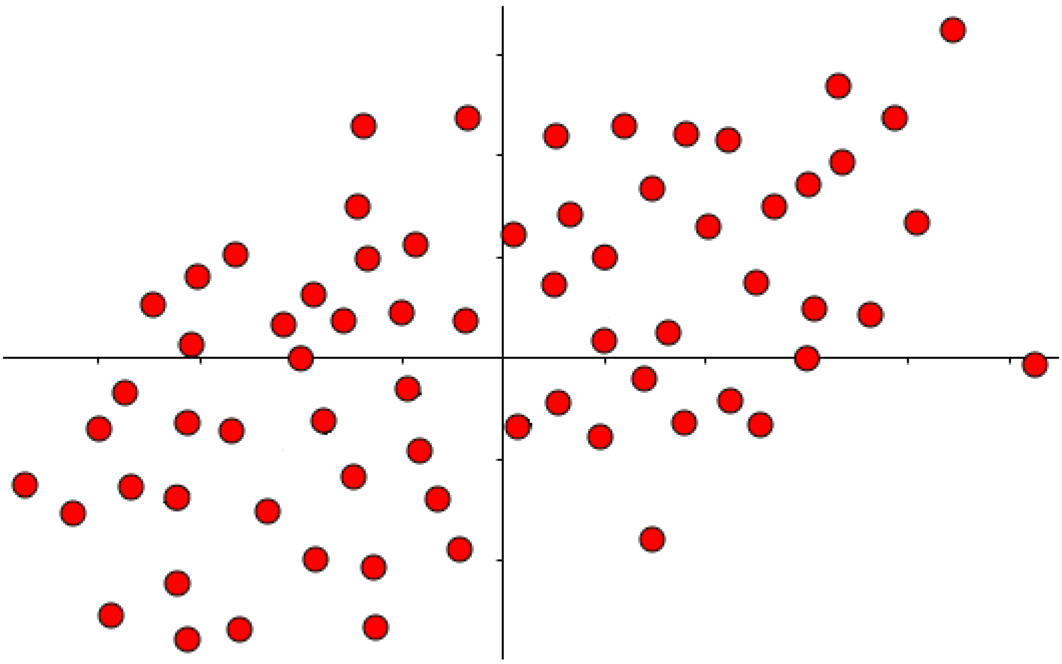
3. The **vectors** of projection (p_{a1} and p_{a2})—which are used to **reconstruct** the original vector ***a*** by adding them together (as a vector sum), and for which it's easy to verify that $p_{a1}=s_{a1}*v_1$ and $p_{a2}=s_{a2}*v_2$—**So they're redundant, as they can be deduced from the former 2 pieces.**

Critical Conclusion:

> Any vector can be expressed in terms of:

> 2. The lengths of projections onto them (s_{a1}, s_{a2}, ...).

All what SVD does is **extend this conclusion** to more than one vector (or point) and to all dimensions :



An example of a dataset (**a point can be considered a vector through the origin**).

Now it becomes a matter of knowing how to handle this mess.

. . .

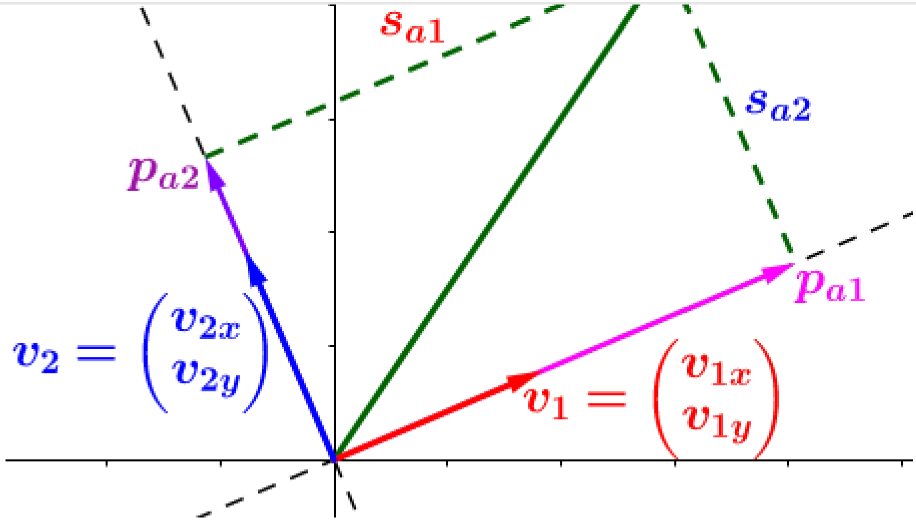
How To Handle This Mess

We can't handle that mess without first handling a single vector!

If you look at many generalizations done in mathematics, you'll find they primarily utilize **matrices.**

> So we have to find a way to express the operation of vector decomposition using matrices.

It turns out to be a natural thing to do:



Same figure as before, but tilting the axes of projection to convince you they aren't confined to x and y. (a_x and a_y are the coordinates of vector **a**, put into a column matrix (aka column vector), as per convention. Same for v_1 and v_2).

We want to decompose (project) the vector ***a*** along unit vectors ***v*1** and ***v*2**.

You may already know (especially if you've watched this) that projection is done by the **dot product** — it gives us the **lengths** of projection (s_a1 and s_a2):

$$a^T \cdot v_1 = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix} = s_{a1}$$

$$a^T \cdot v_2 = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{2x} \\ v_{2y} \end{pmatrix} = s_{a2}$$

Projecting (a) onto v1 and v2.

But that's redundant. We can utilize the efficiency of matrices…

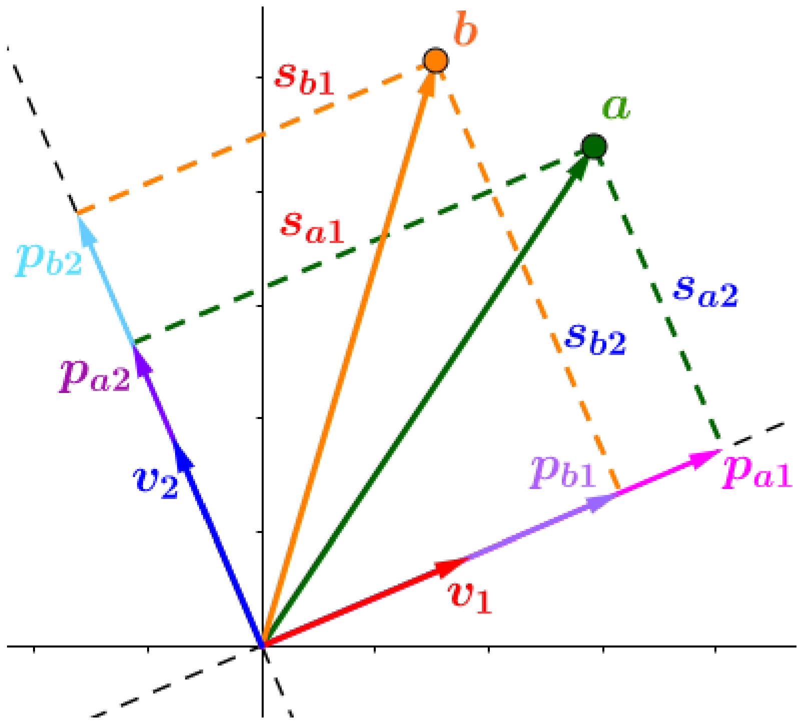
$$a^T \cdot V = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{pmatrix} = \begin{pmatrix} s_{a1} & s_{a2} \end{pmatrix}$$

…to write both equations in one go, by adding an **extra column** for each unit vector.

We can even add more points…

...by adding an **extra row** for each point. **S** is the matrix containing the lengths of projections.

Here's how it looks like, after adding that point ***b***:



It's now easy to generalize to any number of points and dimensions:

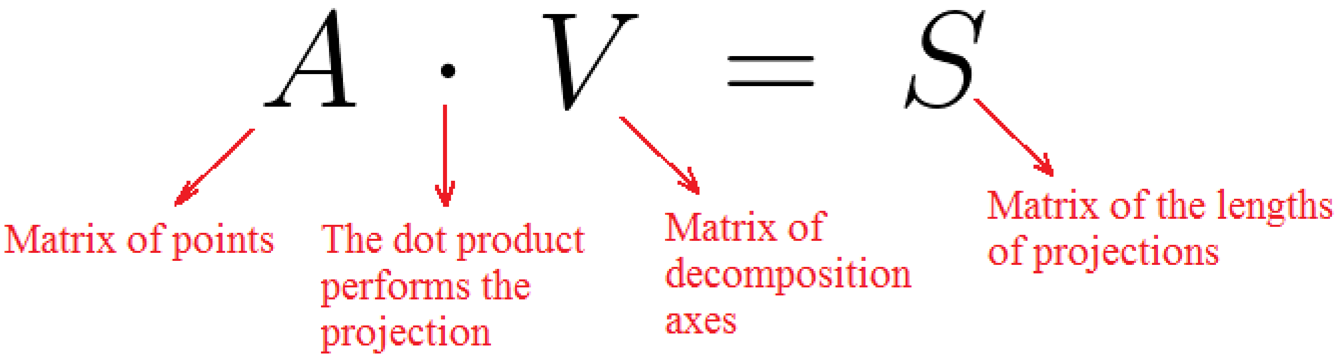
$$A \cdot V = \underbrace{\begin{pmatrix} a_x & a_y & \cdots \\ b_x & b_y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}}_{n \times d} \cdot \underbrace{\begin{pmatrix} v_{1x} & v_{2x} & \cdots \\ v_{1y} & v_{2y} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}}_{d \times d} = \underbrace{\begin{pmatrix} s_{a1} & s_{a2} & \cdots \\ s_{b1} & s_{b2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}}_{n \times d} = S$$

n = no. of points, **d** = no. of dimensions, **A** = matrix containing points, **V** = matrix containing the decomposition axes, **S** = matrix containing lengths of projection.

Mathematical elegance at its best.

$$a^T \cdot v_1 = \begin{pmatrix} a_x & a_y \end{pmatrix} \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix} = \begin{pmatrix} s_{a1} \end{pmatrix}$$

. . .



The dot product in this case is just **ordinary matrix multiplication.**

That’s exactly the same as saying:

$$A = S\, V^{-1} = S\, V^{T}$$

Because **V** contains orthonormal columns, its inverse = its transpose (property of orthogonal matrices).

Which is all what SVD says (remember the Critical Conclusion):

> Any set of vectors (A) can be expressed in terms of their lengths of projections (S) on some set of orthogonal axes (V).

However, we are not quite there yet. The conventional SVD formula says:

$$A = U\, \Sigma\, V^{T}$$

But that just means we want to see how:

$$S = U\, \Sigma$$

And that’s what we’re going to do.

. . .

$$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix}$$

A column vector containing the lengths of projections of each point on the 1st axis *v1*

A column vector containing the lengths of projections of each point on the 2nd axis *v2*

It turns out (for reasons to be seen later) that it’s best if we could **normalize** these column vectors, i.e. make them of **unit length**.

This is done by doing the equivalent of **dividing each column vector by its magnitude, but in matrix form**.

. . .

But first, a numerical example to see how this “division” thing is done.

$$M = \begin{pmatrix} 2 & 3 \\ 2 & 3 \end{pmatrix}$$

Let’s say we want to divide the ***1st*** column of ***M*** by **2**. We will surely have to **multiply by another matrix** to preserve the equality:

$$M = \begin{pmatrix} 1 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 2 & 3 \end{pmatrix}$$

It’s straightforward to verify that the unknown matrix is nothing more than the **identity matrix, with the *1st* element replaced by the divisor = 2**:

Dividing the ***2nd*** column by **3** now becomes a direct matter — just replace the ***2nd*** element of the identity matrix by **3**:

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 2 & 3 \end{pmatrix}$$

It should be obvious how this operation can be generalized to any matrix of any size.

. . .

We now want to apply the above “division” concept to the matrix ***S***.

To normalize the columns of ***S***, we divide them by their magnitude…

$$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix}$$

$$\text{Magnitude of 1st column} = \sigma_1 = \sqrt{(s_{a1})^2 + (s_{b1})^2}$$

$$\text{Magnitude of 2nd column} = \sigma_2 = \sqrt{(s_{a2})^2 + (s_{b2})^2}$$

…by doing with ***S*** what we did with ***M*** in the example above:

$$S = \begin{pmatrix} \frac{s_{a1}}{\sigma_1} & \frac{s_{a2}}{\sigma_2} \\ \\ \frac{s_{b1}}{\sigma_1} & \frac{s_{b2}}{\sigma_2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} u_{a1} & u_{a2} \\ u_{b1} & u_{b2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}$$

Finally…

$$A = U\,\Sigma\,V^T$$

Singular Value Decomposition (Compact or Thin version)

Of course, some fine details and rigorous mathematics were, justifiably, swept under the rug, in order to *not* distract from the core concept.

. . .

Interpretation

Let's talk about this ***U*** and **Σ** …

$$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix} = \begin{pmatrix} u_{a1} & u_{a2} \\ u_{b1} & u_{b2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = U\,\Sigma$$

Lengths of projections on $v1$

Lengths of projections on v2

Lengths of projections on $v1$, but divided by σ1 to become a unit vector

Lengths of projections on $v2$, but divided by σ2 to become a unit vector

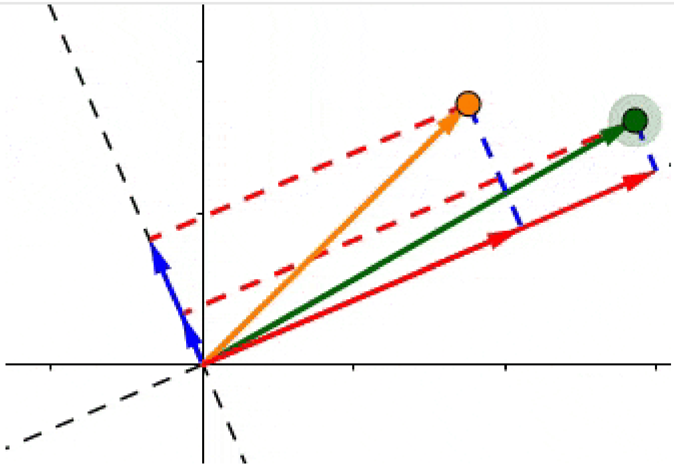
Hmm?

What about the sigmas? Why did we burden ourselves with normalizing ***S*** to find them?

We've already seen that (σ_i) is the **square root of the sum of squared projection lengths,** of all points, onto the *i*th unit vector $\boldsymbol{v_i}$.

What does that mean?

$$\sigma_1 = 1.99$$
$$\sigma_2 = 0.48$$



Red segments = projections on v1. Blue segments = projections on v2. **The closer the points to a specific axis of projection, the larger the value of the corresponding** σ**.**

Since the sigmas contain, in their definition, the sum of projection lengths onto a specific axis, **they represent how close all the points are to that axis.**

E.g. if $\sigma_1 > \sigma_2$, then most points are closer to $\boldsymbol{v_1}$ than $\boldsymbol{v_2}$, and vice versa.

That's of immense utility in the myriad applications of SVD.

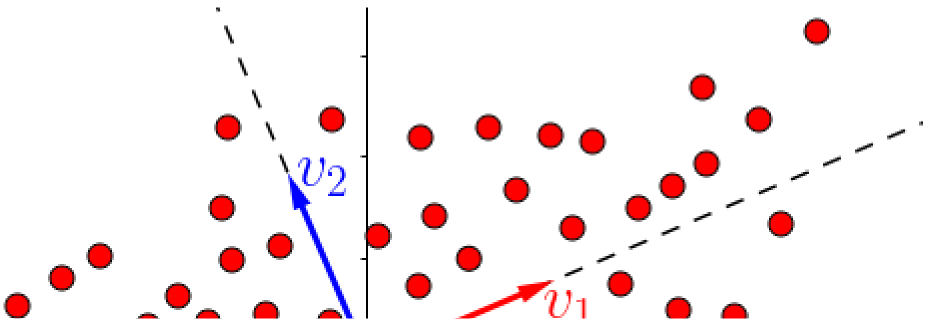
. . .

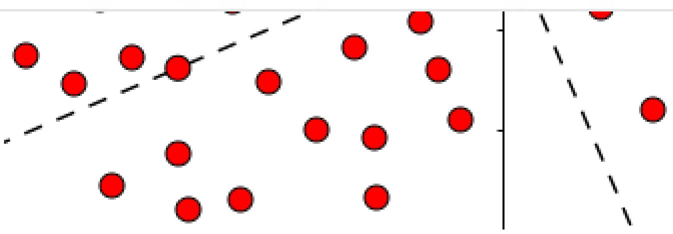
The Main Application

The algorithms of finding the SVD of a matrix don't choose the projection directions (columns of matrix *V*) randomly.

> They choose them to be the Principal Components of the dataset (matrix A).

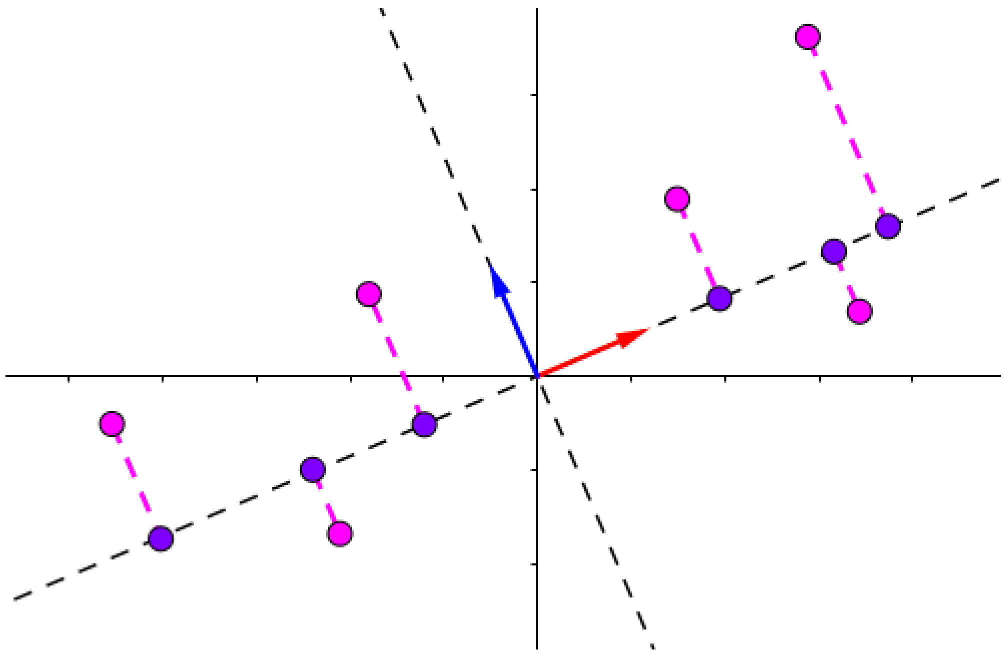
If you've read my first article, you know very well what the principal components are…





...they're the lines of largest variation (largest variance).

You also know from the same that the goal of **dimensionality reduction** is to **project the dataset on the line (or plane) of largest variance**:



Magenta: points before projection. Violet: points after projection (reduced dimensionality).

Now the act of projecting the dataset using SVD becomes a snap, since **all the points are *already projected*** (decomposed) on all the principal components (the $\boldsymbol{v_i}$ unit vectors):

$$\begin{pmatrix} a_x & a_y & \cdots \\ b_x & b_y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} = A = SV^T = \begin{pmatrix} \boxed{\begin{matrix} s_{a1} \\ s_{b1} \\ \vdots \end{matrix}} & \boxed{\begin{matrix} s_{a2} \\ s_{b2} \\ \vdots \end{matrix}} & \begin{matrix} \cdots \\ \cdots \\ \ddots \end{matrix} \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} v_{1x} \\ v_{1y} \\ \vdots \end{matrix}} & \boxed{\begin{matrix} v_{2x} \\ v_{2y} \\ \vdots \end{matrix}} & \begin{matrix} \cdots \\ \cdots \\ \ddots \end{matrix} \end{pmatrix}^T$$

The dataset

The projection lengths of the dataset on 1st principal component

The projection lengths of the dataset on 2nd principal component

1st principal component

2nd principal component

$$A' = SV^T = \begin{pmatrix} s_{a1} & \cancel{s_{a2}} & \cancel{\cdots} \\ s_{b1} & \cancel{s_{b2}} & \cancel{\cdots} \\ \vdots & \cancel{\vdots} & \cancel{\ddots} \end{pmatrix} \begin{pmatrix} v_{1x} & \cancel{v_{2x}} & \cancel{\cdots} \\ v_{1y} & \cancel{v_{2y}} & \cancel{\cdots} \\ \vdots & \cancel{\vdots} & \cancel{\ddots} \end{pmatrix}^T$$

…all we have to do is remove all columns not related to the 1st principal component. The projected dataset in now **A’**.

Multiplying the two matrices (S and V^T above) results in the matrix A′ containing the projected points (violet) in the last graph.

· · ·

And that’s it…for now.

Machine Learning Data Science Svd Intuition Tutorial