**Cosine Similarity – Understanding the math and how it works (with python codes)**

* [October 22, 2018](https://www.machinelearningplus.com/2018/10/22/)
* [Selva Prabhakaran](https://www.machinelearningplus.com/author/selva86/)

*Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.*

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

By the end of this tutorial you will know:

1. What is cosine similarity is and how it works?
2. How to compute cosine similarity of documents in python?
3. What is soft cosine similarity and how its different from cosine similarity?
4. When to use soft cosine similarity and how to compute it in python?

Cosine Similarity – Understanding the math and how it works. Photo by Matt Lamers

[1. Introduction](https://www.machinelearningplus.com/nlp/cosine-similarity/#1introduction)  
[2. What is Cosine Similarity and why is it advantageous?](https://www.machinelearningplus.com/nlp/cosine-similarity/#2whatiscosinesimilarityandwhyisitadvantageous)  
[3. Cosine Similarity Example](https://www.machinelearningplus.com/nlp/cosine-similarity/#3cosinesimilarityexample)  
[4. How to Compute Cosine Similarity in Python?](https://www.machinelearningplus.com/nlp/cosine-similarity/#4howtocomputecosinesimilarityinpython)  
[5. Soft Cosine Similarity](https://www.machinelearningplus.com/nlp/cosine-similarity/#5softcosinesimilarity) [6. Conclusion](https://www.machinelearningplus.com/nlp/cosine-similarity/#6conclusion)

**1. Introduction**

A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents.

But this approach has an inherent flaw.

That is, as the size of the document increases, the number of common words tend to increase even if the documents talk about different topics. The cosine similarity helps overcome this fundamental flaw in the ‘count-the-common-words’ or Euclidean distance approach.

**2. What is Cosine Similarity and why is it advantageous?**

**Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.**

Mathematically, **Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space.**

In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

As a similarity metric, how does cosine similarity differ from the number of common words?

When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word ‘cricket’ appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

**3. Cosine Similarity Example**

Let’s suppose you have 3 documents based on a couple of star cricket players – **Sachin Tendulkar and Dhoni.**

**Two of the documents (A) and (B) are from the wikipedia pages on the respective players and the third document (C) is a smaller snippet from Dhoni’s wikipedia page.**

<https://www.machinelearningplus.com/wp-content/uploads/2018/10/the_three_documents.png>

The Three Documents

As you can see, all three documents are connected by a common theme – the game of Cricket.

Our objective is to quantitatively estimate the similarity between the documents.

For ease of understanding, let’s consider only the top 3 common words between the documents: ‘Dhoni’, ‘Sachin’ and ‘Cricket’.

You would expect Doc B and Doc C, that is the two documents on Dhoni would have a higher similarity over Doc A and Doc B, because, Doc C is essentially a snippet from Doc B itself.

However, if we go by the number of common words, **the two larger documents will have the most common words and therefore will be judged as most similar, which is exactly what we want to avoid.**

The results would be more congruent when we use the cosine similarity score to assess the similarity.

Let me explain.

Let’s project the documents in a 3-dimensional space, where each dimension is a frequency count of either: ‘Sachin’, ‘Dhoni’ or ‘Cricket’. When plotted on this space, the 3 documents would appear something like this.

3d Projection

As you can see, Doc Dhoni\_Small and the main Doc Dhoni are oriented closer together in 3-D space, even though they are far apart by magnitiude.

It turns out, the closer the documents are by angle, the higher is the Cosine Similarity (Cos theta).

<https://www.machinelearningplus.com/wp-content/uploads/2018/10/Cosine-Similarity-Formula-1.png>Cosine Similarity Formula

As you include more words from the document, it’s harder to visualize a higher dimensional space. But you can directly compute the cosine similarity using this math formula. Enough with the theory. Let’s compute the cosine similarity with Python’s scikit learn.

**4. How to Compute Cosine Similarity in Python?**

We have the following 3 texts:  
**1. Doc Trump (A) :** Mr. Trump became president after winning the political election. Though he lost the support of some republican friends, Trump is friends with President Putin.

**2. Doc Trump Election (B) :** President Trump says Putin had no political interference is the election outcome. He says it was a witchhunt by political parties. He claimed President Putin is a friend who had nothing to do with the election.

**3. Doc Putin (C) :** Post elections, Vladimir Putin became President of Russia.

President Putin had served as the Prime Minister earlier in his political career.

Since, Doc B has more in common with Doc A than with Doc C, I would expect the Cosine between A and B to be larger than (C and B).

# Define the documents

doc\_trump = "Mr. Trump became president after winning the political election. Though he lost the support of some republican friends, Trump is friends with President Putin"

doc\_election = "President Trump says Putin had no political interference is the election outcome. He says it was a witchhunt by political parties. He claimed President Putin is a friend who had nothing to do with the election"

doc\_putin = "Post elections, Vladimir Putin became President of Russia. President Putin had served as the Prime Minister earlier in his political career"

documents = [doc\_trump, doc\_election, doc\_putin]

To compute the cosine similarity, you need the word count of the words in each document.

The CountVectorizer or the TfidfVectorizer from scikit learn lets us compute this.

The output of this comes as a sparse\_matrix.

On this, am optionally converting it to a pandas dataframe to see the word frequencies in a tabular format.

# Scikit Learn

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

# Create the Document Term Matrix

count\_vectorizer = CountVectorizer(stop\_words='english')

count\_vectorizer = CountVectorizer()

sparse\_matrix = count\_vectorizer.fit\_transform(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want to see the word frequencies.

doc\_term\_matrix = sparse\_matrix.todense()

df = pd.DataFrame(doc\_term\_matrix,

columns=count\_vectorizer.get\_feature\_names(),

index=['doc\_trump', 'doc\_election', 'doc\_putin'])

df

Doc-Term Matrix

Even better, I could have used the TfidfVectorizer() instead of CountVectorizer(), because it would have downweighted words that occur frequently across docuemnts.

Then, use cosine\_similarity() to get the final output.

It can take the document term matrix as a pandas dataframe as well as a sparse matrix as inputs.

# Compute Cosine Similarity

from sklearn.metrics.pairwise import cosine\_similarity

print(cosine\_similarity(df, df))

#> [[ 1. 0.48927489 0.37139068]

#> [ 0.48927489 1. 0.38829014]

#> [ 0.37139068 0.38829014 1. ]]

**5. Soft Cosine Similarity**

Suppose if you have another set of documents on a completely different topic, say ‘food’, you want a similarity metric that gives higher scores for documents belonging to the same topic and lower scores when comparing docs from different topics.

In such case, we need to consider the semantic meaning should be considered.

That is, words similar in meaning should be treated as similar.

For Example, ‘President’ vs ‘Prime minister’, ‘Food’ vs ‘Dish’, ‘Hi’ vs ‘Hello’ should be considered similar.

For this, converting the words into respective word vectors, and then, computing the similarities can address this problem.

Soft Cosines

Let’s define 3 additional documents on food items.

# Define the documents

doc\_soup = "Soup is a primarily liquid food, generally served warm or hot (but may be cool or cold), that is made by combining ingredients of meat or vegetables with stock, juice, water, or another liquid. "

doc\_noodles = "Noodles are a staple food in many cultures. They are made from unleavened dough which is stretched, extruded, or rolled flat and cut into one of a variety of shapes."

doc\_dosa = "Dosa is a type of pancake from the Indian subcontinent, made from a fermented batter. It is somewhat similar to a crepe in appearance. Its main ingredients are rice and black gram."

documents = [doc\_trump, doc\_election, doc\_putin, doc\_soup, doc\_noodles, doc\_dosa]

To get the word vectors, you need a word embedding model. Let’s download the FastText model using gensim’s downloader api.

import gensim

# upgrade gensim if you can't import softcossim

from gensim.matutils import softcossim

from gensim import corpora

import gensim.downloader as api

from gensim.utils import simple\_preprocess

print(gensim.\_\_version\_\_)

#> '3.6.0'

# Download the FastText model

fasttext\_model300 = api.load('fasttext-wiki-news-subwords-300')

To compute soft cosines, you need the dictionary (a map of word to unique id), the corpus (word counts) for each sentence and the similarity matrix.

# Prepare a dictionary and a corpus.

dictionary = corpora.Dictionary([simple\_preprocess(doc) for doc in documents])

# Prepare the similarity matrix

similarity\_matrix = fasttext\_model300.similarity\_matrix(dictionary, tfidf=None, threshold=0.0, exponent=2.0, nonzero\_limit=100)

# Convert the sentences into bag-of-words vectors.

sent\_1 = dictionary.doc2bow(simple\_preprocess(doc\_trump))

sent\_2 = dictionary.doc2bow(simple\_preprocess(doc\_election))

sent\_3 = dictionary.doc2bow(simple\_preprocess(doc\_putin))

sent\_4 = dictionary.doc2bow(simple\_preprocess(doc\_soup))

sent\_5 = dictionary.doc2bow(simple\_preprocess(doc\_noodles))

sent\_6 = dictionary.doc2bow(simple\_preprocess(doc\_dosa))

sentences = [sent\_1, sent\_2, sent\_3, sent\_4, sent\_5, sent\_6]

If you want the soft cosine similarity of 2 documents, you can just call the softcossim() function

# Compute soft cosine similarity

print(softcossim(sent\_1, sent\_2, similarity\_matrix))

#> 0.567228632589

But, I want to compare the soft cosines for all documents against each other. So, create the soft cosine similarity matrix.

import numpy as np

import pandas as pd

def create\_soft\_cossim\_matrix(sentences):

len\_array = np.arange(len(sentences))

xx, yy = np.meshgrid(len\_array, len\_array)

cossim\_mat = pd.DataFrame([[round(softcossim(sentences[i],sentences[j], similarity\_matrix) ,2) for i, j in zip(x,y)] for y, x in zip(xx, yy)])

return cossim\_mat

soft\_cosine\_similarity\_matrix(sentences)

Soft cosine similarity matrix

As one might expect, the similarity scores amongst similar documents are higher (see the red boxes).

**6. Conclusion**

Now you should clearly understand the math behind the computation of cosine similarity and how it is advantageous over magnitude based metrics like Euclidean distance.

Soft cosines can be a great feature if you want to use a similarity metric that can help in clustering or classification of documents. If you want to dig in further into natural language processing, the [gensim tutorial](https://www.machinelearningplus.com/nlp/gensim-tutorial/) is highly recommended.

**A One-Stop Shop for Principal Component Analysis**

[[](https://matthew-brems.medium.com/?source=post_page-----5582fb7e0a9c--------------------------------)](https://matthew-brems.medium.com/?source=post_page-----5582fb7e0a9c--------------------------------)

[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----5582fb7e0a9c--------------------------------)](https://towardsdatascience.com/?source=post_page-----5582fb7e0a9c--------------------------------)

[Matt Brems](https://matthew-brems.medium.com/?source=post_page-----5582fb7e0a9c--------------------------------)

·

Published in

[Towards Data Science](https://towardsdatascience.com/?source=post_page-----5582fb7e0a9c--------------------------------)

·

16 min read

·

Apr 17, 2017

At the beginning of the [textbook I used for my graduate stat theory](http://people.unica.it/musio/files/2008/10/Casella-Berger.pdf) class, the authors (George Casella and Roger Berger) explained in the preface why they chose to write a textbook:

“When someone discovers that you are writing a textbook, one or both of two questions will be asked. The first is “Why are you writing a book?” and the second is “How is your book different from what’s out there?” The first question is fairly easy to answer. You are writing a book because you are not entirely satisfied with the available texts.”

I apply the authors’ logic here. Principal component analysis (PCA) is an important technique to understand in the fields of statistics and data science… but when putting a lesson together for my [General Assembly](https://generalassemb.ly/) students, I found that the resources online were too technical, didn’t fully address our needs, and/or provided conflicting information. It’s safe to say that I’m not “entirely satisfied with the available texts” here.

As a result, I wanted to put together the “What,” “When,” “How,” and “Why” of PCA as well as links to some of the resources that can help to further explain this topic. Specifically, I want to present the rationale for this method, the math under the hood, some best practices, and potential drawbacks to the method.

While I want to make PCA as accessible as possible, the algorithm we’ll cover is pretty technical. Being familiar with some or all of the following will make this article and PCA as a method easier to understand: [matrix operations/linear algebra](https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab) (matrix multiplication, matrix transposition, matrix inverses, matrix decomposition, eigenvectors/eigenvalues) and statistics/machine learning (standardization, variance, covariance, independence, linear regression, feature selection). I’ve embedded links to illustrations of these topics throughout the article, but hopefully these will serve as a reminder rather than required reading to get through the article.

**What is PCA?**

Let’s say that you want to predict what the [gross domestic product](http://www.investopedia.com/terms/g/gdp.asp) (GDP) of the United States will be for 2017. You have lots of information available: the U.S. GDP for the first quarter of 2017, the U.S. GDP for the entirety of 2016, 2015, and so on. You have any publicly-available economic indicator, like the unemployment rate, inflation rate, and so on. You have U.S. Census data from 2010 estimating how many Americans work in each industry and [American Community Survey](https://www.census.gov/programs-surveys/acs/about.html) data updating those estimates in between each census. You know how many members of the House and Senate belong to each political party. You could gather stock price data, the number of [IPOs](https://en.wikipedia.org/wiki/Initial_public_offering) occurring in a year, and [how many CEOs](https://www.nytimes.com/2017/03/09/business/bloomberg-iger-business-executives-president.html?_r=0) [seem to be mounting a bid for public office](http://www.latimes.com/business/technology/la-fi-tn-zuckerberg-president-20170120-story.html). Despite being an overwhelming number of variables to consider, this *just scratches the surface*.

[TL;DR](http://www.urbandictionary.com/define.php?term=tl%3Bdr) — you have *a lot* of variables to consider.

If you’ve worked with a lot of variables before, you know this can present problems. Do you understand the relationships between each variable? Do you have so many variables that you are in danger of overfitting your model to your data or that you might be violating assumptions of whichever modeling tactic you’re using?

You might ask the question, “How do I take all of the variables I’ve collected and focus on only a few of them?” In technical terms, you want to “reduce the dimension of your feature space.” By reducing the dimension of your feature space, you have fewer relationships between variables to consider and you are less likely to overfit your model. (Note: This doesn’t immediately mean that overfitting, etc. are no longer concerns — but we’re moving in the right direction!)

Somewhat unsurprisingly, ***reducing*** the ***dimension*** of the feature space is called “***dimensionality reduction***.” There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

* Feature Elimination
* Feature Extraction

**Feature elimination** is what it sounds like: we reduce the feature space by eliminating features. In the GDP example above, instead of considering every single variable, we might drop all variables except the three we think will best predict what the U.S.’s gross domestic product will look like. Advantages of feature elimination methods include simplicity and maintaining interpretability of your variables.

As a disadvantage, though, you gain no information from those variables you’ve dropped. If we only use last year’s GDP, the proportion of the population in manufacturing jobs per the most recent American Community Survey numbers, and unemployment rate to predict this year’s GDP, we’re missing out on whatever the dropped variables could contribute to our model. By eliminating features, we’ve also entirely eliminated any benefits those dropped variables would bring.

**Feature extraction**, however, doesn’t run into this problem. Say we have ten independent variables. In feature extraction, we create ten “new” independent variables, where each “new” independent variable is a combination of each of the ten “old” independent variables. However, we create these new independent variables in a specific way and order these new variables by how well they predict our dependent variable.

You might say, “Where does the dimensionality reduction come into play?” Well, we keep as many of the new independent variables as we want, but we drop the “least important ones.” Because we ordered the new variables by how well they predict our dependent variable, we know which variable is the most important and least important. But — and here’s the kicker — because these new independent variables are combinations of our old ones, we’re still keeping the most valuable parts of our old variables, even when we drop one or more of these “new” variables!

Principal component analysis is a technique for *feature extraction* — so it combines our input variables in a specific way, then we can drop the “least important” variables while still retaining the most valuable parts of all of the variables! *As an added benefit, each of the “new” variables after PCA are all independent of one another.* This is a benefit because the [assumptions of a linear model](http://people.duke.edu/~rnau/testing.htm) require our independent variables to be independent of one another. If we decide to fit a linear regression model with these “new” variables (see “principal component regression” below), this assumption will necessarily be satisfied.

**When should I use PCA?**

1. Do you want to reduce the number of variables, but aren’t able to identify variables to completely remove from consideration?
2. Do you want to ensure your variables are independent of one another?
3. Are you comfortable making your independent variables less interpretable?

If you answered “yes” to all three questions, then PCA is a good method to use. If you answered “no” to question 3, you **should not** use PCA.

**How does PCA work?**

The section after this discusses *why* PCA works, but providing a brief summary before jumping into the algorithm may be helpful for context:

* We are going to calculate a matrix that summarizes how our variables all relate to one another.
* We’ll then break this matrix down into two separate components: direction and magnitude. We can then understand the “directions” of our data and its “magnitude” (or how “important” each direction is). The screenshot below, [from the setosa.io applet](http://setosa.io/ev/principal-component-analysis/), displays the two main directions in this data: the “red direction” and the “green direction.” In this case, the “red direction” is the more important one. We’ll get into why this is the case later, but given how the dots are arranged, can you see why the “red direction” looks more important than the “green direction?” (*Hint: What would fitting a line of best fit to this data look like?*)

A picture containing diagram, plot, text, line

Description automatically generated

Our original data in the xy-plane. ([Source](http://setosa.io/ev/principal-component-analysis/).)

* We will transform our original data to align with these important directions (which are combinations of our original variables). The screenshot below ([again from setosa.io](http://setosa.io/ev/principal-component-analysis/)) is the same exact data as above, but transformed so that the *x*- and *y*-axes are now the “red direction” and “green direction.” What would the line of best fit look like here?

A picture containing text, diagram, plot, screenshot

Description automatically generated

Our original data transformed by PCA. ([Source](http://setosa.io/ev/principal-component-analysis/).)

* While the visual example here is two-dimensional (and thus we have two “directions”), think about a case where our data has more dimensions. By identifying which “directions” are most “important,” we can compress or project our data into a smaller space by dropping the “directions” that are the “least important.” **By projecting our data into a smaller space, we’re reducing the dimensionality of our feature space… but because we’ve transformed our data in these different “directions,” we’ve made sure to keep all original variables in our model!**

Here, I walk through an algorithm for conducting PCA. I try to avoid being too technical, but it’s impossible to ignore the details here, so my goal is to walk through things as explicitly as possible. A deeper intuition of *why* the algorithm works is presented in the next section.

Before starting, you should have tabular data organized with *n* rows and likely *p+1* columns, where one column corresponds to your dependent variable (usually denoted ***Y***) and *p* columns where each corresponds to an independent variable (the matrix of which is usually denoted ***X***).

1. If a ***Y*** variable exists and is part of your data, then separate your data into ***Y*** and ***X***, as defined above — we’ll mostly be working with ***X***. (Note: if there exists no column for **Y**, that’s okay — skip to the next point!)
2. Take the matrix of independent variables ***X*** and, for each column, subtract the mean of that column from each entry. (This ensures that each column has a mean of zero.)
3. Decide whether or not to standardize. Given the columns of ***X***, are features with higher variance more important than features with lower variance, or is the importance of features independent of the variance? (In this case, importance means how well that feature predicts ***Y***.) **If the importance of features is independent of the variance of the features, then divide each observation in a column by that column’s standard deviation.** (This, combined with step 2, standardizes each column of *X* to make sure each column has mean zero and standard deviation 1.) Call the centered (and possibly standardized) matrix ***Z***.
4. Take the matrix ***Z***, [transpose it](https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_14.html), and multiply the transposed matrix by ***Z***. (Writing this out mathematically, we would write this as ***Z***ᵀ***Z***.) The resulting matrix is the [covariance matrix of ***Z***](http://www.itl.nist.gov/div898/handbook/pmc/section5/pmc541.htm), up to a constant.
5. (This is probably the toughest step to follow — stick with me here.) Calculate the eigenvectors and their corresponding eigenvalues of ***Z***ᵀ***Z***. This is quite easily done in most computing packages— in fact, the [eigendecomposition](https://cseweb.ucsd.edu/~dasgupta/291-unsup/lec7.pdf) of ***Z***ᵀ***Z*** is where we decompose ***Z***ᵀ***Z*** into ***PDP***⁻¹, where ***P*** is the matrix of eigenvectors and ***D*** is the diagonal matrix with eigenvalues on the diagonal and values of zero everywhere else. The eigenvalues on the diagonal of ***D*** will be associated with the corresponding column in ***P*** — that is, the first element of ***D*** is λ₁ and the corresponding eigenvector is the first column of ***P***. This holds for all elements in ***D*** and their corresponding eigenvectors in ***P***. We will always be able to calculate ***PDP***⁻¹ in this fashion. (Bonus: for those interested, we can always calculate ***PDP***⁻¹ in this fashion because ***Z***ᵀ***Z*** is a [symmetric](https://en.wikipedia.org/wiki/Symmetric_matrix), [positive semidefinite matrix](https://en.wikipedia.org/wiki/Positive-definite_matrix).)
6. Take the eigenvalues λ₁, λ₂, …, λ*p* and sort them from largest to smallest. In doing so, sort the eigenvectors in ***P*** accordingly. (For example, if λ₂ is the largest eigenvalue, then take the second column of ***P*** and place it in the first column position.) Depending on the computing package, this may be done automatically. Call this sorted matrix of eigenvectors ***P\****. (The columns of ***P\**** should be the same as the columns of ***P***, but perhaps in a different order.) **Note that these eigenvectors are independent of one another.**
7. Calculate ***Z\**** = ***ZP\****. This new matrix, ***Z\****, is a centered/standardized version of ***X*** but now each observation is a combination of the original variables, where the weights are determined by the eigenvector. **As a bonus, because our eigenvectors in *P\** are independent of one another, each column of *Z\** is also independent of one another!**

A screenshot of a graph

Description automatically generated with low confidence

An example from [setosa.io](http://setosa.io/ev/principal-component-analysis/) where we transform five data points using PCA. The left graph is our original data ***X****; the right graph would be our transformed data* ***Z\*****.*

Note two things in this graphic:

* The two charts show the exact same data, but the right graph reflects the original data transformed so that our axes are now the principal components.
* In both graphs, the principal components are perpendicular to one another. **In fact, every principal component will ALWAYS be** [**orthogonal**](https://en.wikipedia.org/wiki/Orthogonality#Statistics.2C_econometrics.2C_and_economics)(a.k.a. official math term for perpendicular) **to every other principal component.** (Don’t believe me? [Try to break the applet](http://setosa.io/ev/principal-component-analysis/)!)

**Because our principal components are orthogonal to one another, they are statistically linearly independent of one another… which is why our columns of *Z\** are linearly independent of one another!**

8. Finally, we need to determine how many features to keep versus how many to drop. There are three common methods to determine this, discussed below and followed by an explicit example:

* **Method 1**: We arbitrarily select how many dimensions we want to keep. Perhaps I want to visually represent things in two dimensions, so I may only keep two features. This is use-case dependent and there isn’t a hard-and-fast rule for how many features I should pick.
* **Method 2**: Calculate the [proportion of variance explained](https://stats.stackexchange.com/questions/22569/pca-and-proportion-of-variance-explained) (briefly explained below) for each feature, pick a threshold, and add features until you hit that threshold. (For example, if you want to explain 80% of the total variability possibly explained by your model, add features with the largest explained proportion of variance until your proportion of variance explained hits or exceeds 80%.)
* **Method 3**: This is closely related to Method 2. Calculate the [proportion of variance explained](https://stats.stackexchange.com/questions/22569/pca-and-proportion-of-variance-explained) for each feature, sort features by proportion of variance explained and plot the cumulative proportion of variance explained as you keep more features. (This plot is called a [scree plot](http://ba-finance-2013.blogspot.com/2012/09/scree-plots-interpretation-and.html), shown below.) One can pick how many features to include by identifying the point where adding a new feature has a significant drop in variance explained relative to the previous feature, and choosing features up until that point. (I call this the “find the elbow” method, as looking at the “bend” or “elbow” in the scree plot determines where the biggest drop in proportion of variance explained occurs.)

Because each eigenvalue is roughly the importance of its corresponding eigenvector, the proportion of variance explained is the sum of the eigenvalues of the features you kept divided by the sum of the eigenvalues of all features.

# Principal Component Analysis

### Explained Visually

By [Victor Powell](http://twitter.com/vicapow)

with text by [Lewis Lehe](http://twitter.com/lewislehe)

Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize.

# 2D example

First, consider a dataset in only two dimensions, like (height, weight). This dataset can be plotted as points in a plane. But if we want to tease out variation, PCA finds a new coordinate system in which every point has a new (x,y) value. The axes don't actually mean anything physical; they're combinations of height and weight called "principal components" that are chosen to give one axes lots of variation.

Drag the points around in the following visualization to see PC coordinate system adjusts.

original data set

0246810x0246810youtput from PCA-6-4-20246pc1-6-4-20246pc2

PCA is useful for eliminating dimensions. Below, we've plotted the data along a pair of lines: one composed of the x-values and another of the y-values.

If we're going to only see the data along one dimension, though, it might be better to make that dimension the principal component with most variation. We don't lose much by dropping PC2 since it contributes the least to the variation in the data set.

0246810x0246810y-6-4-20246pc1-6-4-20246pc2

# 3D example

With three dimensions, PCA is more useful, because it's hard to see through a cloud of data. In the example below, the original data are plotted in 3D, but you can project the data into 2D through a transformation no different than finding a camera angle: rotate the axes to find the best angle. To see the "official" PCA transformation, click the "Show PCA" button. The PCA transformation ensures that the horizontal axis PC1 has the most variation, the vertical axis PC2 the second-most, and a third axis PC3 the least. Obviously, PC3 is the one we drop.

-10-50510pc1-10-50510pc2-10-50510x-10-50510y-10-50510z-10-50510pc1-10-50510pc2-10-50510pc3

# Eating in the UK (a 17D example)

Original example from Mark Richardson's class notes [Principal Component Analysis](http://people.maths.ox.ac.uk/richardsonm/SignalProcPCA.pdf)

What if our data have way more than 3-dimensions? Like, **17** dimensions?! In the table is the average consumption of 17 types of food in grams per person per week for every country in the UK.

The table shows some interesting variations across different food types, but overall differences aren't so notable. Let's see if PCA can eliminate dimensions to emphasize how countries differ.

3755724514721055419314711027202536854881983601374156135472671494664120993674103314358635518733415061394585324214621036218412295756617175041822033715721474757322715821036423516011378742658035702033651256

175EnglandN IrelandScotlandWalesAlcoholic drinksBeveragesCarcase meatCerealsCheeseConfectioneryFats and oilsFishFresh fruitFresh potatoesFresh VegOther meatOther VegProcessed potatoesProcessed VegSoft drinksSugars

Here's the plot of the data along the first principal component. Already we can see something is different about Northern Ireland.

-300-200-1000100200300400500pc1EnglandWalesScotlandN Ireland

Now, see the first and second principal components, we see Northern Ireland a major outlier. Once we go back and look at the data in the table, this makes sense: the Northern Irish eat way more grams of fresh potatoes and way fewer of fresh fruits, cheese, fish and alcoholic drinks. It's a good sign that structure we've visualized reflects a big fact of real-world geography: Northern Ireland is the only of the four countries not on the island of Great Britain. (If you're confused about the differences among England, the UK and Great Britain, see: [this video](https://www.youtube.com/watch?v=rNu8XDBSn10).)

-300-200-1000100200300400500-400-300-200-1000100200300400pc1pc2EnglandWalesScotlandN Ireland

A screen shot of a graph

Description automatically generated with medium confidence

Scree Plot for Genetic Data. ([Source](http://www.improvedoutcomes.com/docs/WebSiteDocs/PCA/Creating_a_Scree_Plot.htm).)

Consider this scree plot for genetic data. (Source: [here](http://www.improvedoutcomes.com/docs/WebSiteDocs/PCA/Creating_a_Scree_Plot.htm).) The red line indicates the proportion of variance explained by each feature, which is calculated by taking that principal component’s eigenvalue divided by the sum of all eigenvalues. The proportion of variance explained by including only principal component 1 is λ₁/(λ₁ + λ₂ + … + λ*p*), which is about 23%. The proportion of variance explained by including only principal component 2 is λ₂/(λ₁ + λ₂ + … + λ*p*), or about 19%.

The proportion of variance explained by including both principal components 1 and 2 is (λ₁ + λ₂)/(λ₁ + λ₂ + … + λ*p*), which is about 42%. This is where the yellow line comes in; the yellow line indicates the cumulative proportion of variance explained if you included all principal components up to that point. For example, the yellow dot above PC2 indicates that including principal components 1 and 2 will explain about 42% of the total variance in the model.

Now let’s go through some examples:

* Method 1: We arbitrarily select a number of principal components to include. Suppose I wanted to keep five principal components in my model. In the genetic data case above, these five principal components explains about 66% of the total variability that would be explained by including all 13 principal components.
* Method 2: Suppose I wanted to include enough principal components to explain 90% of the total variability explained by all 13 principal components. In the genetic data case above, I would include the first 10 principal components and drop the final three variables from ***Z\****.
* Method 3: Here, we want to “find the elbow.” In the scree plot above, we see there’s a big drop in proportion of variability explained between principal component 2 and principal component 3. In this case, we’d likely include the first two features and drop the remaining features. As you can see, this method is a bit subjective as “elbow” doesn’t have a mathematically precise definition and, in this case, we’d include a model that explains only about 42% of the total variability.

*(Note: Some scree plots will have the size of eigenvectors on the Y axis rather than the proportion of variance. This leads to equivalent results, but requires the user to manually calculate the proportion of variance.* [*An example of this can be seen here*](http://documentation.statsoft.com/STATISTICAHelp.aspx?path=Glossary/GlossaryTwo/S/ScreePlotScreeTest)*.)*

**Once we’ve dropped the transformed variables we want to drop, we’re done! That’s PCA.**

**But, like, \*why\* does PCA work?**

While PCA is a very technical method relying on in-depth linear algebra algorithms, it’s a relatively intuitive method when you think about it.

* First, the covariance matrix ***Z***ᵀ***Z*** is a matrix that contains estimates of how every variable in ***Z*** relates to every other variable in ***Z***. Understanding how one variable is associated with another is quite powerful.
* Second, eigenvalues and eigenvectors are important. Eigenvectors represent directions. Think of plotting your data on a multidimensional scatterplot. Then one can think of an individual eigenvector as a particular “direction” in your scatterplot of data. Eigenvalues represent magnitude, or importance. Bigger eigenvalues correlate with more important directions.
* Finally, we make an assumption that more variability in a particular direction correlates with explaining the behavior of the dependent variable. Lots of variability usually indicates signal, whereas little variability usually indicates noise. Thus, the more variability there is in a particular direction is, theoretically, indicative of something important we want to detect. (The [setosa.io PCA applet](http://setosa.io/ev/principal-component-analysis/) is a great way to play around with data and convince yourself why it makes sense.)

Thus, PCA is a method that brings together:

1. A measure of how each variable is associated with one another. (Covariance matrix.)
2. The directions in which our data are dispersed. (Eigenvectors.)
3. The relative importance of these different directions. (Eigenvalues.)

PCA combines our predictors and allows us to drop the eigenvectors that are relatively unimportant.

**Are there extensions to PCA?**

Yes, more than I can address here in a reasonable amount of space. The one I’ve most frequently seen is [principal component regression](https://onlinecourses.science.psu.edu/stat857/node/157), where we take our untransformed ***Y*** and regress it on the subset of ***Z\**** that we didn’t drop. (This is where the independence of the columns of ***Z\**** comes in; by regressing ***Y*** on ***Z\****, we know that the required independence of independent variables will necessarily be satisfied. However, we will need to still check our other assumptions.)

The other commonly-seen variant I’ve seen is [kernel PCA](https://en.wikipedia.org/wiki/Kernel_principal_component_analysis).

**Conclusion**

I hope you found this article helpful! Check out some of the resources below for more in-depth discussions of PCA. Let me know what you think, especially if there are suggestions for improvement.

I’ve been told that a Chinese translation of this article has been made available [here](https://www.jqr.com/article/000429). (Thanks,

[Jakukyo Friel](https://medium.com/u/c2d21d069ac?source=post_page-----5582fb7e0a9c--------------------------------)

!)

I want to offer many thanks to my friends [Ritika Bhasker](https://medium.com/@mostlyinane), [Joseph Nelson](https://medium.com/@josephofiowa), and [Corey Smith](https://www.linkedin.com/in/corey-smith-03203b78/) for their suggestions and edits. You should check Ritika and Joseph out on Medium — their posts are far more entertaining than mine. (Corey is too focused on not getting his Ph.D. research scooped to have a Medium presence.)

I also want to give a **huge** h/t to the [setosa.io applet](http://setosa.io/ev/principal-component-analysis/) for its visual and intuitive display of PCA.

Edit: Thanks to [Michael Matthews](https://www.linkedin.com/in/michael-j-matthews/) for noticing a typo in the formula for ***Z\**** in Step 7 above. He correctly pointed out that ***Z\**** = ***ZP\****, not ***Z***ᵀ***P\****. Thanks also to [Chienlung Cheung](https://medium.com/@chienlungcheung) for noticing another typo in Step 8 above and noted that I had conflated “eigenvector” with “eigenvalue” in one line.

**Machine Learning — Singular Value Decomposition (SVD) & Principal Component Analysis (PCA)**

[[](https://jonathan-hui.medium.com/?source=post_page-----1d45e885e491--------------------------------)](https://jonathan-hui.medium.com/?source=post_page-----1d45e885e491--------------------------------)

[Jonathan Hui](https://jonathan-hui.medium.com/?source=post_page-----1d45e885e491--------------------------------)

·

16 min read

·

Mar 6, 2019

A picture containing jigsaw puzzle, art, board game, indoor

Description automatically generated

Photo by [Sheldon Nunes](https://unsplash.com/@sheldonnunes?utm_source=medium&utm_medium=referral)

In machine learning (ML), some of the most important linear algebra concepts are the singular value decomposition (SVD) and principal component analysis (PCA). With all the raw data collected, how can we discover structures? For example, with the interest rates of the last 6 days, can we understand its composition to spot trends?

A picture containing text, font, screenshot, number

Description automatically generated

This becomes even harder for high-dimensional raw data. It is like finding a needle in a haystack. SVD allows us to extract and untangle information. In this article, we will detail SVD and PCA. We assume you have [basic](https://medium.com/@jonathan_hui/machine-learning-linear-algebra-a5b1658f0151) linear algebra knowledge including rank and [eigenvectors](https://medium.com/@jonathan_hui/machine-learning-linear-algebra-eigenvalue-and-eigenvector-f8d0493564c9). If you experience difficulties in reading this article, I will suggest refreshing those concepts first. At the end of the article, we will answer some questions in the interest rate example above. This article also contains optional sections. Feel free to skip it according to your interest level.

**Misconceptions (optional for beginners)**

I realize a few common questions that non-beginners may ask. Let me address the elephant in the room first. Is PCA dimension reduction? PCA reduces dimension but it is far more than that. I like the Wiki description (but if you don’t know PCA, this is just gibberish):

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

From a simplified perspective, PCA transforms data linearly into new properties that are not correlated with each other. For ML, positioning PCA as feature extraction may allow us to explore its potential better than dimension reduction.

What is the difference between SVD and PCA? SVD gives you the whole nine-yard of diagonalizing a matrix into [special matrices](https://medium.com/@jonathan_hui/machine-learning-linear-algebra-special-matrices-c750cd742dfe) that are easy to manipulate and to analyze. It lay down the foundation to untangle data into independent components. PCA skips less significant components. Obviously, we can use SVD to find PCA by truncating the less important basis vectors in the original SVD matrix.

**Matrix diagonalization**

In the [article](https://medium.com/@jonathan_hui/machine-learning-linear-algebra-eigenvalue-and-eigenvector-f8d0493564c9) on eigenvalue and eigenvectors, we describe a method to decompose an n × n square matrix *A* into



For example,

A picture containing text, font, line, diagram

Description automatically generated

A matrix can be diagonalized if *A* is a square matrix and *A* has n linearly independent eigenvectors. Now, it is time to develop a solution for all matrices using SVD.

**Singular vectors & singular values**

The matrix ***AAᵀ*** and ***AᵀA*** are very special in linear algebra. Consider any m × n matrix ***A***, we can multiply it with *Aᵀ* to form ***AAᵀ*** and ***AᵀA*** separately. These matrices are

* symmetrical,
* square,
* at least positive semidefinite (eigenvalues are zero or positive),
* both matrices have the same positive eigenvalues, and
* both have the same rank *r* as *A*.

In addition, the covariance matrices that we often use in ML are in this form. Since they are symmetric, we can choose its eigenvectors to be orthonormal (perpendicular to each other with unit length) — this is a fundamental property for [symmetric matrices](https://medium.com/@jonathan_hui/machine-learning-linear-algebra-special-matrices-c750cd742dfe).

A picture containing text, font, line, diagram

Description automatically generated

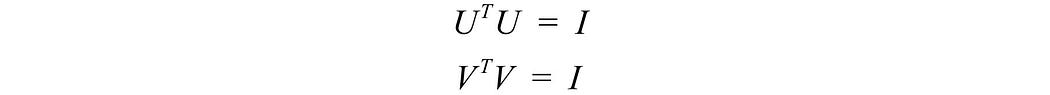
Let’s introduce some terms that frequently used in SVD. We name the eigenvectors for ***AAᵀ*** as *uᵢ* and ***AᵀA*** as *vᵢ* here and call these sets of eigenvectors *u* and *v* the **singular vectors** of *A*. Both matrices have the same positive eigenvalues. The square roots of these eigenvalues are called **singular values**.

Not too many explanations so far but let’s put everything together first and the explanations will come next. We concatenate vectors *uᵢ* into *U* and *vᵢ* into *V* to form orthogonal matrices.

A picture containing sketch, diagram, drawing, white

Description automatically generated

Since these vectors are orthonormal, it is easy to prove that *U* and *V* obey



**SVD**

Let’s start with the hard part first. SVD states that **any** matrix *A* can be factorized as:



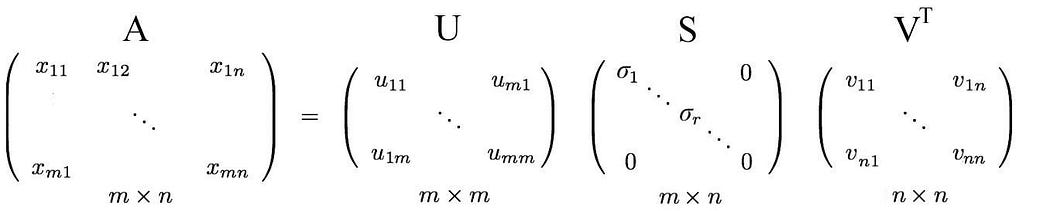
where *U* and *V* are orthogonal matrices with orthonormal eigenvectors chosen from ***AAᵀ*** and ***AᵀA*** respectively. ***S*** is a diagonal matrix with *r* elements equal to the root of the positive eigenvalues of *AAᵀ* or *Aᵀ A* (both matrics have the same positive eigenvalues anyway). The diagonal elements are composed of singular values.

A picture containing text, handwriting, sketch, line

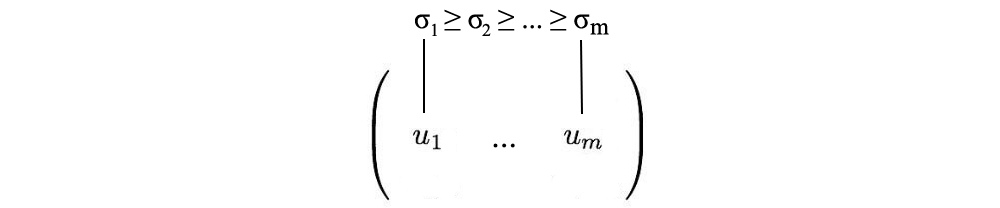
Description automatically generated

i.e. an m× n matrix can be factorized as:





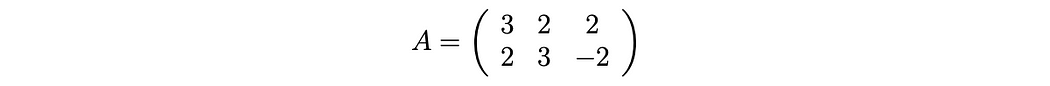
We can arrange eigenvectors in different orders to produce *U* and *V*. To standardize the solution, we order the eigenvectors such that vectors with higher eigenvalues come before those with smaller values.



Comparing to eigendecomposition, SVD works on non-square matrices. *U* and *V* are invertible for any matrix in SVD and they are orthonormal which we love it. Without proof here, we also tell you that singular values are more numerical stable than eigenvalues.

**Example** ([Source](http://www.d.umn.edu/~mhampton/m4326svd_example.pdf) of the example)

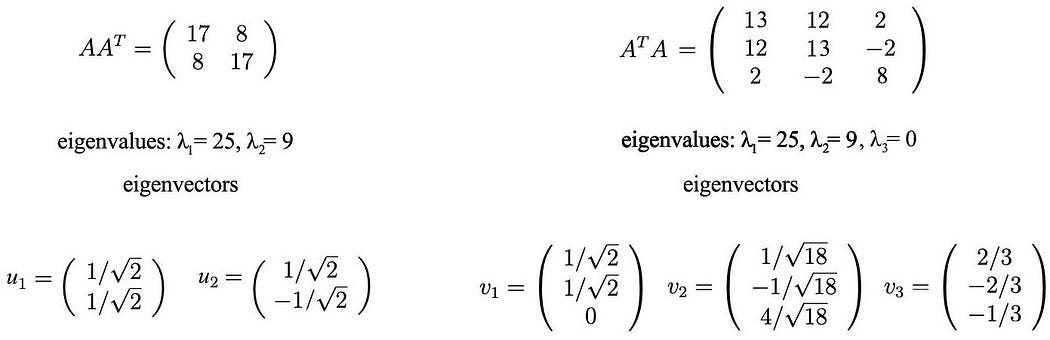
Before going too far, let’s demonstrate it with a simple example. This will make things very easy to understand.



We calculate:



These matrices are at least positive semidefinite (all eigenvalues are positive or zero). As shown, they share the same positive eigenvalues (25 and 9). The figure below also shows their corresponding eigenvectors.



The singular values are the square root of positive eigenvalues, i.e. 5 and 3. Therefore, the SVD composition is

A picture containing text, font, handwriting, white

Description automatically generated

**Proof (optional)**

To proof SVD, we want to solve *U, S,* and *V* with:

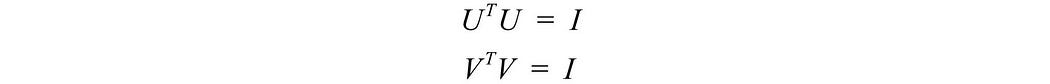
A picture containing text, white, font, black

Description automatically generated

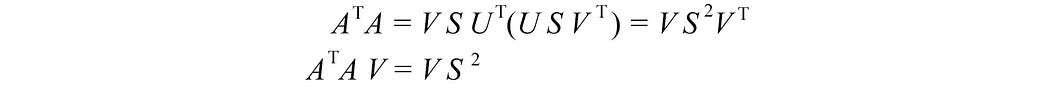
We have 3 unknowns. Hopefully, we can solve them with the 3 equations above. The transpose of *A* is



Knowing



We compute *AᵀA*,



The last equation is equilvant to the eigenvector definition for the matrix (*AᵀA*)*.* We just put all eigenvectors in a matrix.

A picture containing text, diagram, font, white

Description automatically generated

with *VS²* equals

A picture containing sketch, typography

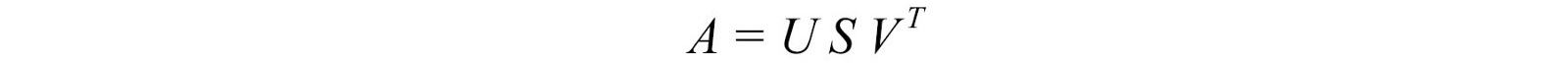
Description automatically generated

*V* hold all the eigenvectors *vᵢ* of *AᵀA* and *S* hold the square roots of all eigenvalues of *AᵀA*. We can repeat the same process for *AAᵀ* and come back with a similar equation.

A picture containing sketch, white, diagram, line

Description automatically generated

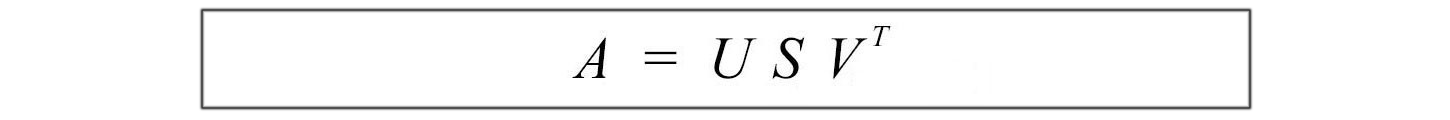
Now, we just solve *U, V* and *S* for



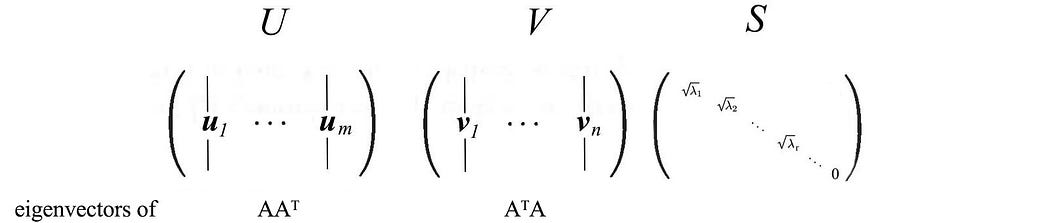
and prove the theorem.

**Recap**

The following is a recap of SVD.

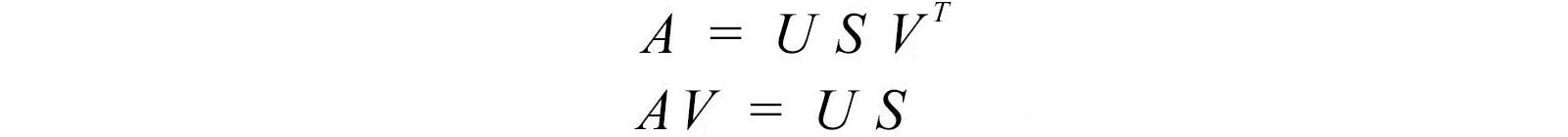


where



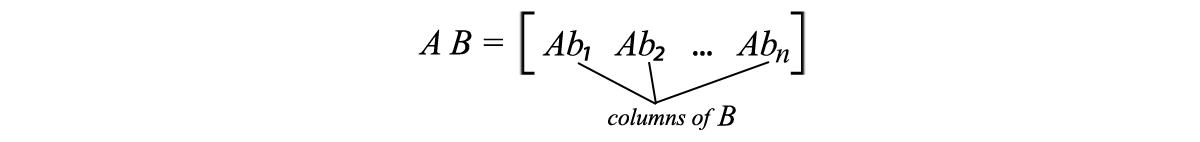
**Reformulate SVD**

Since matrix *V* is orthogonal, *VᵀV* equals *I*. We can rewrite the SVD equation as:



This equation establishes an important relationship between *uᵢ* and *vᵢ*.

Recall

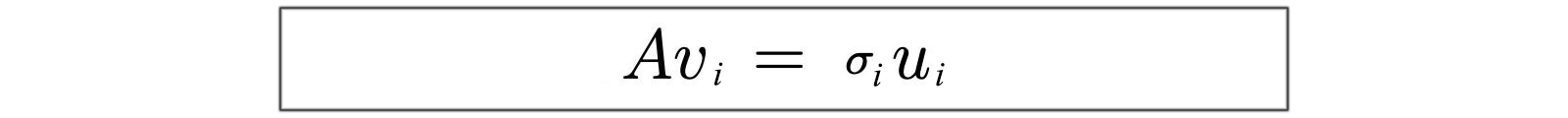


Apply *AV = US*,

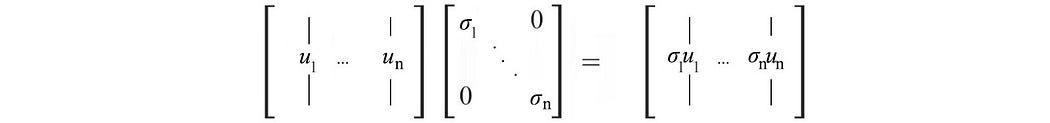
A picture containing text, handwriting, font, sketch

Description automatically generated

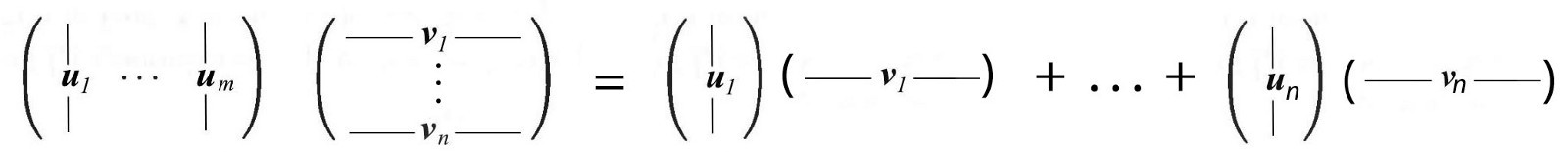
This can be generalized as



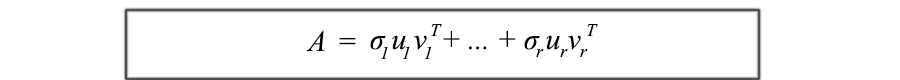
Recall,



and

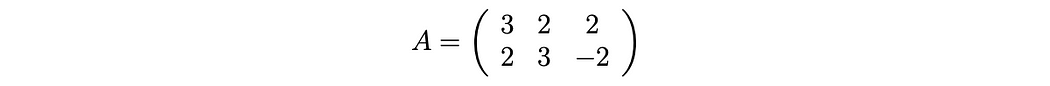


The SVD decomposition can be recognized as a series of outer products of *uᵢ* and *vᵢ.*



This formularization of SVD is the key to understand the components of *A*. It provides an important way to break down an m × n array of entangled data into *r* components. Since *uᵢ* and *vᵢ* are unit vectors, we can even ignore terms (*σᵢuᵢvᵢᵀ*) with very small singular value *σᵢ*. (We will come back to this later.)

Let’s first reuse the example before and show how it works.



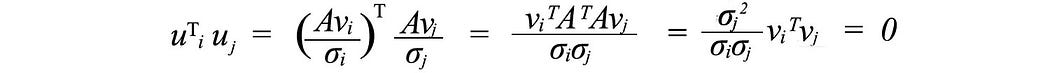
The matrix *A* above can be decomposed as

A picture containing handwriting, text, font, white

Description automatically generated

**Column space, row space, left nullspace and nullspace (Optional-for advanced users)**

Next, we will take a look at what *U* & *V* composed of. Let’s say *A* is an m × n matrix of rank *r*. *AᵀA* will be an n× n symmetric matrix. All symmetric matrices can choose n orthonormal eigenvectors *vⱼ*. Because of *Avᵢ = σᵢuᵢ* and *vⱼ* are orthonormal eigenvectors of *AᵀA,* we can calculate the value of *uᵢ*ᵀ*uⱼ* as



It equals zero. i.e. *uᵢ* and *uⱼ* are orthogonal with each other. As shown previously, they are also eigenvectors of *AAᵀ.*

From *Avᵢ = σuᵢ*, we can recognize that *uᵢ* is a column vector of *A.*

A picture containing text, font, white, line

Description automatically generated

Because *A* has a rank of *r*, we can choose these *r* *uᵢ* vectors to be orthonormal. So what are the remaining m - r orthogonal eigenvectors for *AAᵀ?* Since left nullspace of *A* is orthogonal to the column space, it is very natural to pick them as the remaining eigenvector. (The left nullsapce *N*(*Aᵀ)* is the space span by *x* in *Aᵀx=0*.) A similar argument will work for the eigenvectors for *AᵀA.* Therefore,

A screenshot of a computer

Description automatically generated with low confidence

To get back to the former SVD equation from

A picture containing text, font, line, white

Description automatically generated

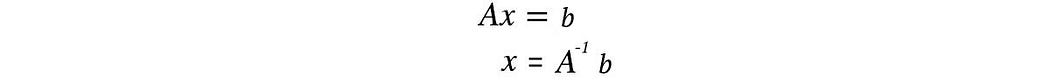
We simply put back the eigenvectors in the left nullspace and nullspace.

A picture containing text, handwriting, font, line

Description automatically generated

**Moore-Penrose Pseudoinverse**

For a linear equation system, we can compute the inverse of a square matrix *A* to solve *x*.



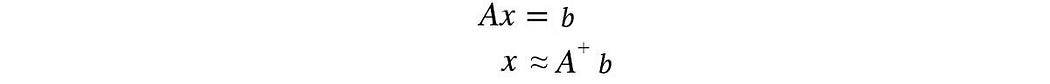
But not all matrices are invertible. Also, in ML, it will be unlikely to find an exact solution with the presence of noise in data. Our objective is to find the model that best fit the data. To find the best-fit solution, we compute a pseudoinverse



which minimizes the least square error below.



And the solution for *x* can be estimated as,



In a linear regression problem, ***x*** is our linear model, ***A*** contains the training data and *b* contains the corresponding labels. We can solve *x* by

A picture containing text, font, receipt, white

Description automatically generated

A picture containing text, design

Description automatically generated

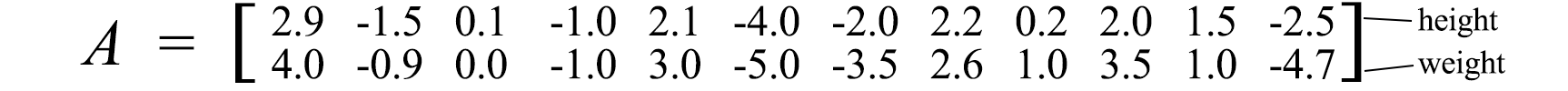
Here is an example.

A picture containing text, font, receipt, algebra

Description automatically generated

**Variance & covariance**

In ML, we identify patterns and relationship. How do we identify the correlation of properties in data? Let’s start the discussion with an example. We sample the height and weight of 12 people and compute their means. We zero-center the original values by subtracting them with its mean. For example, Matrix *A* below holds the adjusted zero-centered height and weight.

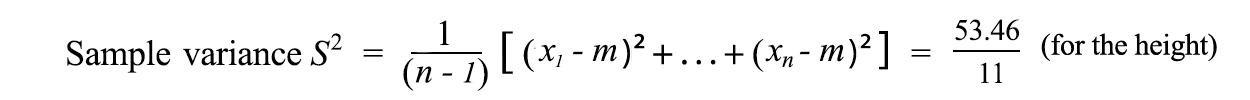


As we plot the data points, we can recognize height and weight are positively related. But how can we quantify such a relationship?

A picture containing line, plot, diagram, number

Description automatically generated

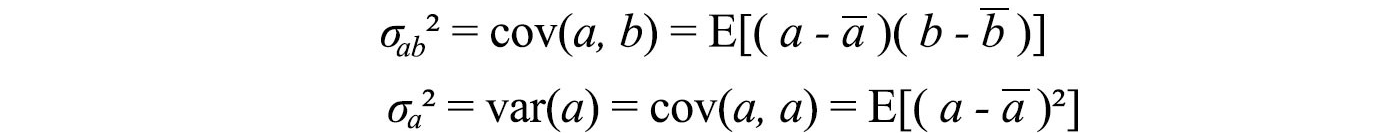
First, how does a property vary? We probably learn the variance from high school. Let’s introduce its cousin. **Sample variance** is defined as :



Note, it is divided by *n-1* instead of *n* in the **variance**. With a limited size of the samples, the sample mean is biased and correlated with the samples. The average square distance from this mean will be smaller than that from the general population. The sample covariance *S*², divided by *n-1,* compensates for the smaller value and can be proven to be an unbiased estimate for variance *σ*². (The proof is not very important so I will simply provide a link for the [proof](https://en.wikipedia.org/wiki/Variance#Sample_variance) here.)

**Covariance matrices**

Variance measures how a variable varies between itself while covariance is between two variables (*a* and *b*).



We can hold all these possible combinations of covariance in a matrix called the **covariance matrix** Σ.

A picture containing text, font, white, handwriting

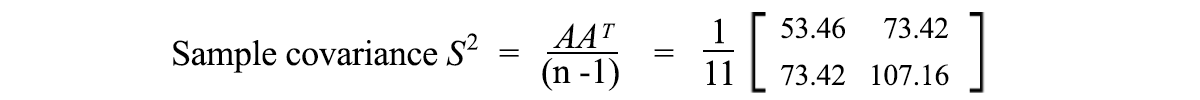
Description automatically generated

We can rewrite this in a simple matrix form.

A picture containing text, font, white, black and white

Description automatically generated

The diagonal elements hold the variances of individual variables (like height) and the non-diagonal elements hold the covariance between two variables. Let’s compute the sample covariance now.



A picture containing text, font, screenshot, graphics

Description automatically generated

The positive sample covariance indicates weight and height are positively correlated. It will be negative if they are negatively correlated and zero if they are independent.

A picture containing text, screenshot, line, diagram

Description automatically generated

**Covariance matrix & SVD**

We can use SVD to decompose the sample covariance matrix. Since *σ₂* is relatively small compared with *σ₁*, we can even ignore the *σ₂* term. When we train an ML model, we can perform a linear regression on the weight and height to form a new property rather than treating them as two separated and correlated properties (where entangled data usually make model training harder).

A picture containing text, screenshot, black, font

Description automatically generated

*u₁* has one significant importance. It is the principal component of *S*.

A screen shot of a graph

Description automatically generated with low confidence

There are a few properties about a sample covariance matrix under the context of SVD:

* The total variance of the data equals the trace of the sample covariance matrix S which equals the sum of squares of S’s singular values. Equipped with this, we can calculate the ratio of variance lost if we drop smaller *σᵢ* terms*.* This reflects the amount of information lost if we eliminate them.



* The first eigenvector *u₁* of *S* points to the most important direction of the data. In our example, it quantifies the typical ratio between weight and height.

A picture containing line, diagram, plot, text

Description automatically generated

Perpendicular least squares

* The error, calculated as the sum of the perpendicular squared distance from the sample points to *u₁*, is the minimum when SVD is used.

**Property**

Covariance matrices are not only symmetric but they are also positive semidefinite. Because variance is positive or zero, *uᵀVu* below is always greater or equal zero. By the energy test, *V* is positive semidefinite.

A picture containing text, black, font, screenshot

Description automatically generated

Therefore,



Often, after some linear transformation *A*, we want to know the covariance of the transformed data. This can be calculated with the transformation matrix *A* and the covariance of the original data.

A picture containing text, font, white, diagram

Description automatically generated

**Correlation matrix**

A correlation matrix is a scaled version of the covariance matrix. A correlation matrix standardizes (scale) the variables to have a standard deviation of 1.

A picture containing text, screenshot, font, line

Description automatically generated

Correlation matrix will be used if variables are in scales of very different magnitudes. Bad scaling may hurt ML algorithms like gradient descent.

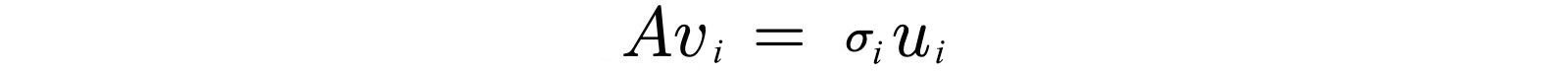
**Visualization**

So far, we have a lot of equations. Let’s visualize what SVD does and develop the insight gradually. SVD factorizes a matrix *A* into *USVᵀ.* Applying *A* to a vector *x* (*Ax*) can be visualized as performing a rotation (*Vᵀ)*, a scaling (*S*) and another rotation (*U*) on *x*.

A picture containing diagram, line, text, plot

Description automatically generated

As shown above, the eigenvector *vᵢ* of *V* is transformed into:



Or in the full matrix form

A picture containing text, font, line, diagram

Description automatically generated

demonstrate for r = m < n

**Insight of SVD**

As described before, the SVD can be formulated as

A picture containing text, screenshot, font, line

Description automatically generated

Since *uᵢ* and *vᵢ* have unit length, the most dominant factor in determining the significance of each term is the singular value *σᵢ*. We purposely sort *σᵢ* in the descending order. If the eigenvalues become too small, we can ignore the remaining terms (+ *σᵢuᵢvᵢᵀ + …*).

A picture containing text, handwriting, font, white

Description automatically generated

This formularization has some interesting implications. For example, we have a matrix contains the return of stock yields traded by different investors.

A screenshot of a computer

Description automatically generated with low confidence

As a fund manager, what information can we get out of it? Finding patterns and structures will be the first step. Maybe, we can identify the combination of stocks and investors that have the largest yields. SVD decompose an n × n matrix into *r* components with the singular value *σᵢ* demonstrating its significant. Consider this as a way to extract entangled and related properties into fewer principal directions with no correlations.



If data is highly correlated, we should expect many *σᵢ* values to be small and can be ignored.

A picture containing diagram, line, plot, font

Description automatically generated

In our previous example, weight and height are highly related. If we have a matrix containing the weight and height of 1000 people, the first component in the SVD decomposition will dominate. The *u₁* vector indeed demonstrates the ratio between weight and height among these 1000 people as we discussed before.

A picture containing diagram, line, plot, text

Description automatically generated

**Principal Component Analysis (PCA)**

Technically, SVD extracts data in the directions with the highest variances respectively. PCA is a linear model in mapping *m*-dimensional input features to *k*-dimensional latent factors (*k* principal components). If we ignore the less significant terms, we remove the components that we care less but keep the principal directions with the highest variances (largest information).

A picture containing text, handwriting, font, sketch

Description automatically generated

Consider the 3-dimensional data points that displayed as blue dots below. It can be approximated by a plane easily.

A picture containing diagram, screenshot, line, drawing

Description automatically generated

[Source](https://www.mathworks.com/help/stats/regress.html)

You may quickly realize that we can use SVD to find the matrix *W*. Consider the data points below that lie on a 2-D space.

A picture containing line, diagram, plot

Description automatically generated

SVD selects a projection that maximizes the variance of their output. Hence, PCA will pick the blue line over the green line if it has a higher variance.

A picture containing line, diagram, plot, design

Description automatically generated

As indicated below, we keep the eigenvectors that have the top *k*th highest singular value.

A picture containing handwriting, text, font

Description automatically generated

**Interest rate**

Let’s illustrate the concept deeper by retracing an example [here](http://math.mit.edu/~gs/linearalgebra/) with the interest rate data originated from the US Treasurer Department. The basis points for 9 different interest rates (from 3 months, 6 months, … to 20 years) over 6 consecutive business days are collected with *A* stored the difference from the previous date. ***A*** also has its elements subtracted by its mean over this period already. i.e. it is zero-centered (across its row).

A picture containing text, font, screenshot, number

Description automatically generated

The sample covariance matrix equals *S = AAᵀ/(5–1)*.

A picture containing text, screenshot, font, black

Description automatically generated

Now we have the covariance matrix *S* that we want to factorize. The SVD decomposition is

A screenshot of a computer

Description automatically generated with medium confidence

From the SVD decomposition, we realize that we can focus on the first three principal components.

A picture containing text, screenshot, font, number

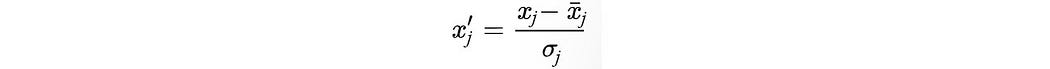
Description automatically generated

As shown, the first principal component is related to a weighted average of the daily change for all maturity lengths. The second principal component adjusts the daily change sensitive to the maturity length of the bond. (The third principal component is likely the curvature — a second-degree derivative.)

We understand the relationship between the interest rate change and maturity well in our daily life. So the principal components reconfirm what we believe how interest rates behave. But when we are presented with unfamiliar raw data, PCA is very helpful to extract the principal components of your data to find the underneath information structure. This may answer some questions on how to find a needle in a haystack.

**Tips**

Scale features before performing SVD.



Say, we want to retain 99% variance, we can choose *k* such that

A picture containing text, white, font, sketch

Description automatically generated

[How Are Principal Component Analysis and Singular Value Decomposition Related? - go to homepage](https://intoli.com/)

* [Home](https://intoli.com/)
* [Features](https://intoli.com/features/)
* [Pricing](https://intoli.com/pricing/)
* [Team](https://intoli.com/team/)
* [Blog](https://intoli.com/blog/)
* [Contact](https://intoli.com/contact/)

# How Are Principal Component Analysis and Singular Value Decomposition Related?

By [Andre Perunicic](mailto:andre@intoli.com) | August 23, 2017

[Follow @prncc](https://github.com/prncc) [65](https://github.com/prncc/followers) [Star](https://github.com/intoli/intoli-article-materials) [264](https://github.com/intoli/intoli-article-materials/stargazers)

# Introduction

Principal Component Analysis, or PCA, is a well-known and widely used technique applicable to a wide variety of applications such as dimensionality reduction, data compression, feature extraction, and visualization. The basic idea is to project a dataset from many correlated coordinates onto fewer uncorrelated coordinates called principal components while still retaining most of the variability present in the data.

Singular Value Decomposition, or SVD, is a computational method often employed to calculate principal components for a dataset. Using SVD to perform PCA is efficient and numerically robust. Moreover, the intimate relationship between them can guide our intuition about what PCA actually does and help us gain additional insights into this technique.

In this post, I will explicitly describe the mathematical relationship between SVD and PCA and highlight some benefits of doing so. If you have used these techniques in the past but aren’t sure how they work internally this article is for you. By the end you should have an understanding of the motivation for PCA and SVD, and hopefully a better intuition about how to effectively employ them.

# Encoding Data as a Matrix

Before we discuss how to actually perform PCA it will be useful to standardize the way in which data is encoded in matrix form. Since PCA and SVD are linear techniques, this will allow us to manipulate the data using linear transformations more easily.

It’s simplest to start with an example, so suppose that you are given the width w

, height h, and length l measurements of n=1000 rectangular boxes. We can encode the measurements of box i as a (wi,hi,li) tuple called a sample. Each sample is a vector in d=3

dimensions, since there are 3 numbers describing it. Because vectors are typically written horizontally, we transpose the vectors to write them vertically:

xi=⎛⎜⎝wihili⎞⎟⎠andxTi=(wi,hi,li).

To pack the data into a single object we simply stack the samples as rows of a data matrix,

X=⎛⎜ ⎜ ⎜ ⎜ ⎜⎝w1h1l1w2h2l2⋮w1000h1000l1000⎞⎟ ⎟ ⎟ ⎟ ⎟⎠.

The columns of this matrix correspond to a single coordinate, i.e., all the measurements of a particular type.

In the general case, we are working with a d

-dimensional dataset comprised of n samples. Instead of using letters like h, w, and l to denote the different coordinates, we simply enumerate the entries of each data-vector so that xTi=(xi1,…,xid)

. Before placing these vectors into a data matrix, we will actually subtract the mean of the data

μ=1nn∑i=1xi=(1nn∑i=1xi1,…,1nn∑i=1xid)T

from each sample for later convenience. We then use these zero-centered vectors as rows of the matrix

X=⎛⎜ ⎜ ⎜ ⎜ ⎜ ⎜⎝xT1−μTxT2−μT⋮xTn−μT⎞⎟ ⎟ ⎟ ⎟ ⎟ ⎟⎠.

See Figure 1 for a visual illustration of the case n=1000

, d=2

. Placing data into a matrix is particularly convenient because it allows us to write the sample covariance around the mean using matrices as

S=1n−1n∑i=1(xi−μ)(xi−μ)T=1n−1XTX.

Dividing by n−1

is a typical way to [correct for the bias](https://en.wikipedia.org/wiki/Bessel%27s_correction) introduced by using the sample mean instead of the true population mean. The covariance matrix will take center stage as we work through understanding principal components and singular value decomposition.

#### What Is the Covariance Matrix?

The j

-th column of X is nothing but the j-th coordinate that our zero-centered dataset is encoded in. The jk-th entry of the product 1n−1XTX is therefore given as the (scaled) dot product of the j-th column of X, denoted x∙,j, and the k-th column of X, denoted x∙,k

. That is,

1n−1x∙,j⋅x∙,k=1n−1xT∙,jx∙,k=1n−1n∑i=1xijxik.

When k=j

this gives us the variance of the data along the k

-th coordinate axis, and otherwise we obtain a measure of how much the two coordinates vary together.

To see that ∑ni=1(xi−μ)(xi−μ)T=XTX

is a short exercise in matrix multiplication. Note that the term on the left is a sum of products of vectors represented as d×1 and 1×d matrices, producing a result of size d×d

.

# Principal Component Analysis

One of the first papers to introduce PCA as its known today was [published in 1933 by Hotelling](https://babel.hathitrust.org/cgi/pt?id=wu.89097139406;view=1up;seq=5). The author’s motivation was to transform a set of possibly correlated variables into “some more fundamental set of independent variables … which determine the values [the original variables] will take.” Coming from psychology, his first choice for naming them was “factors,” but given that this term already had a meaning in mathematics he called the reduced set of variables “components” and the technique to find them “the method of principal components.” These components are chosen sequentially in a way that lets their “contributions to the variances of [the original variables] have as great a total as possible.”

Mathematically, the goal of Principal Component Analysis, or PCA, is to find a collection of k≤d

unit vectors vi∈Rd (for i∈1,…,k

) called Principal Components, or PCs, such that

1. the variance of the dataset projected onto the direction determined by vi

 is maximized and

 vi is chosen to be orthogonal to v1,…,vi−1

1. .

Now, the projection of a vector x∈Rd

onto the line determined by any vi is simply given as the dot product vTix. This means that the variance of the dataset projected onto the first Principal Component v1

can be written as

1n−1n∑i=1(vT1xi−vT1μ)2=vT1Sv1.

To actually find v1

we have to maximize this quantity, subject to the additional constraint that ∥v1∥=1

. Solving this optimization problem using [the method of Lagrange multipliers](https://en.wikipedia.org/wiki/Lagrange_multiplier) turns out to imply that

Sv1=λ1v1,

which just means that v1

is an eigenvector of the covariance matrix S. In fact, since ∥v1∥=vT1v1=1 we also conclude that the corresponding eigenvalue is exactly equal to the variance of the dataset along v1

, i.e.,

vT1Sv1=λ1.

You can continue this process by projecting the data onto a new direction v2

while enforcing the additional constraint that v1⊥v2, then onto v3 while enforcing v3⊥v1,v2

, and so on.

The end result is that the first k

principal components of X correspond exactly to the eigenvectors of the covariance matrix S

ordered by their eigenvalues. Moreover, the eigenvalues are exactly equal to the variance of the dataset along the corresponding eigenvectors.

### Intoli Smart Proxies

If you're doing serious web scraping, then using proxies is a must. Our [Intoli Smart Proxies service](https://intoli.com/beta/) makes it easy to stop getting blocked by bot mitigation services. Your requests are intelligently routed through clean residential IPs where they are likely to succeed, failed requests are automatically retried, and you can even make requests through remote browsers preloaded with customizizations that make it hard to detect your scraper. Enter your email address below to get access to the same tooling that we use for all of our own web scraping!

Top of Form

Bottom of Form

You may have noticed that this result suggests that there exists a full set of orthonormal eigenvectors for S over R. Indeed, since S is a real symmetric matrix, meaning that S=ST, the [Real Spectral Theorem](http://www.math.lsa.umich.edu/~speyer/417/SpectralTheorem.pdf) implies exactly that. This is a non-trivial result which we will make use of later in the article, so let’s expand on it a little bit. Consider the case of k=d<n. Taking k=d principal components may seem like a strange choice if our purpose is to understand X through a lower dimensional subspace, but doing so allows us to construct a d×d matrix V whose columns are the eigenvectors of S and which therefore diagnoalizes S

, i.e.,

S=VΛVT=r∑i=1λivivTi,

where Λ=diag(λ1,…,λd)

and r=rank(X). In other words, the principal components are the columns of a rotation matrix and form the axes of a new basis which can be thought of as “aligning” with the dataset X. Of course, this image works best when X

is blobby or looks approximately normal.

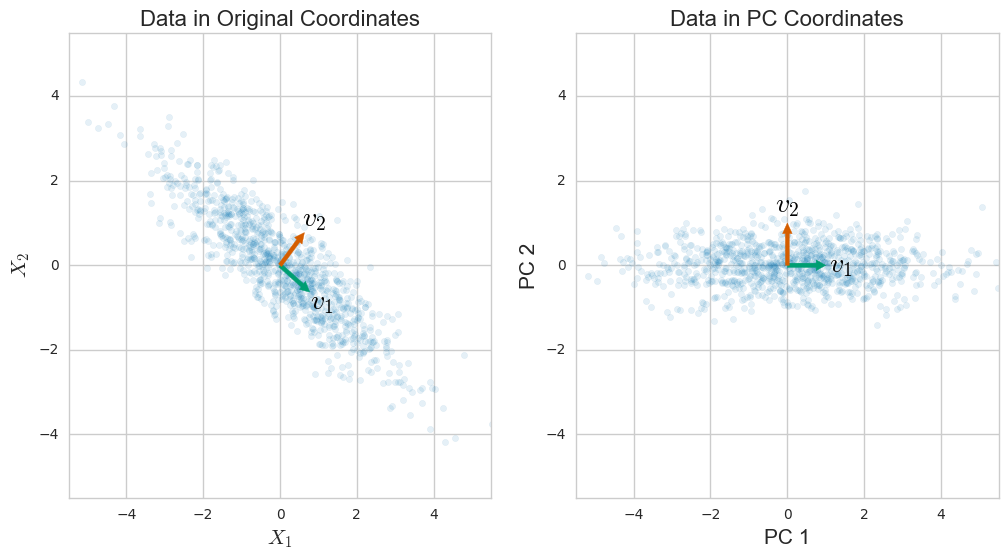


Figure 1: The original and uncorrelated view of 1000 samples drawn from a multivariate Gaussian.

### Looking for a job at an exciting startup?

We have some close personal friends building an awesome tutoring platform, and they're looking to hire. If this sounds like something you might be interested in, then [shoot us an email](mailto:contact@intoli.com) and we'll get you in touch!

# Singular Value Decomposition

Singular Value Decomposition is a matrix factorization method utilized in many numerical applications of linear algebra such as PCA. This technique enhances our understanding of what principal components are and provides a robust computational framework that lets us compute them accurately for more datasets.

Let’s start with a review of SVD for an arbitrary n×d

matrix A. SVD is motivated by the fact that when viewed as a linear transformation, A maps the unit sphere Sd⊂Rd to a (hyper)ellipse in Rn. Let’s consider an example with n=d=2

in order to more easily visualize this fact.

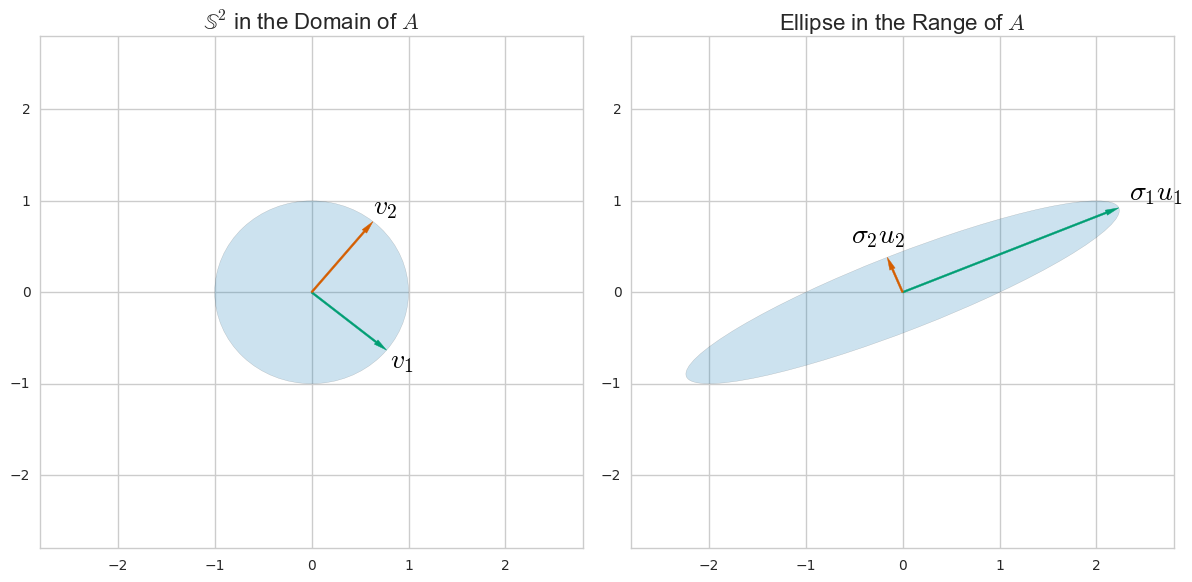


Figure 2: Action of *A=(1201)*

on the unit sphere in *R2*

.

From this figure we can extract a few useful definitions which hold for arbitrary dimensions with a few extra caveats:

* The lengths σi

of the semi-axes of the ellipse ASd∈Rn are the singular values of A

 .

 The unit vectors ui along the semi-axes of the ellipse are called the “left” singular vectors of A

 .

 The unit vectors vi such that Avi=σiui are called the “right” singular vectors of A

* .

Proving the existence and uniqueness of the singular values and singular vectors is a bit beyond the scope of this article, but the proof is relatively straightforward and can be found in any book on numerical linear algebra (see below for a reference). The two-dimensional image in Figure 2 hopefully provides enough guidance for the result to be intuitive, however.

### Get Help from Our Data Experts

Do you need to make sense of your data? Whether it’s variable importance identification, dimensionality reduction, or other data analysis tasks our experts can guide you in the right direction.

What may not be immediately apparent from the picture is that depending on n,d and r=rank(A), some of the left singular vectors may “collapse” to zero. This happens when the matrix A does not have full rank for instance, since then its range must be a subspace of Rn with dimension r<d. In general, there are exactly r=rank(A)

singular values and the same number of left singular vectors.

By stacking the vectors vi

and ui into columns of matrices ˆV and ˆU, respectively, we can write the relations Avi=σiui

as

AˆV=ˆUˆΣ,

where ˆΣ=diag(σ1,…,σr)

. By padding ˆΣ with zeros and adding arbitrary orthonormal columns to ˆU and ˆV

we obtain a more convenient factorization

AV=UΣ.

Since V

is unitary – that is, it has unit length, orthonormal columns – it follows that V−1=VT, so multiplying by VT gives us the singular value decomposition of A

:

A=UΣVT.

Things got a bit busy here, so here’s a visual summary of this result.

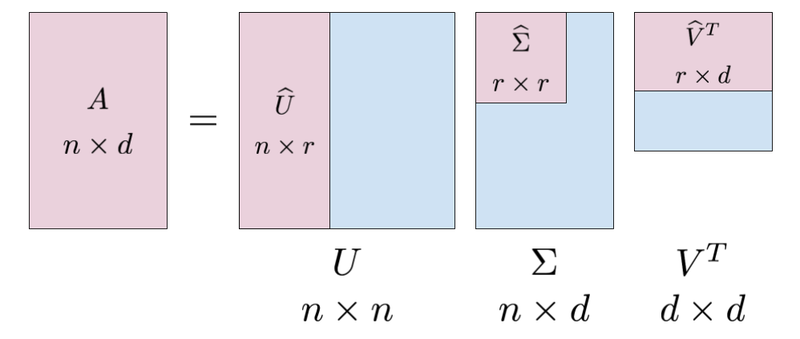


Figure 3: Components of the singular value decomposition of a matrix *A*

. The internal areas correspond to the stacked vectors and singular values motivated by Figure 2.

Note that this result is in the end consistent with the ellipse picture above since the effect of the three transformations is to switch to a new basis using VT

, stretch axes to get a (hyper)ellipse using Σ, and rotate and reflect using U

(which doesn’t affect the shape of the ellipse).

# Relation Between SVD and PCA

Since any matrix has a singular value decomposition, let’s take A=X

and write

X=UΣVT.

We have so far thought of A

as a linear transformation, but there’s nothing preventing us from using SVD on a data matrix. In fact, note that from the decomposition we have

XTX=(UΣVT)T(UΣVT)=VΣTUTUΣVT=V(ΣTΣ)VT,

which means that XTX

and ΣTΣ are similar. Similar matrices have the same eigenvalues, so the eigenvalues λi of the covariance matrix S=1n−1XTX are related to the singular values σi of the matrix X

via

σ2i=(n−1)λi,

for i∈{1,…,r}

, where as usual r=rank(A)

.

To fully relate SVD and PCA we also have to describe the correspondence between principal components and singular vectors. For the right singular vectors we take

ˆVT=⎛⎜ ⎜ ⎜⎝vT1⋮vTr⎞⎟ ⎟ ⎟⎠

where vi

are the principal components of X

. For the left singular vectors we take

ui=1√(n−1)λiXvi.

Before proving that these choices are correct, let’s verify that they at least make intuitive sense. From Figure 2 we can see that the right singular vectors vi

are an orthonormal set of d-dimensional vectors that span the row space of the data. Since X is zero centered we can think of them as capturing the spread of the data around the mean in a sense reminiscent of PCA. We also see that the column space of X is generated by the left singular vectors ui. The column space of a data matrix is a summary of the different coordinates of the data, so it makes sense that we’ve chosen ui to be a (scaled) projection of X into the direction of the i

-th principal component. These observations are encouraging but imprecise, so let’s actually prove that our choices are correct algebraically. The result will follow from this general fact.

Fact. Let A=UΣVT

be the singular decomposition of the real matrix A

. Then,

A=r∑i=1σiuivTj.

Proof. Let’s prove this statement in a small example to simplify the notation. The general case will follow analogously. So, let vi=(v1i,v2i)T

for i=1,2 and ui=(u1i,u2i,u3i) for i=1,2,3

. That is,

UΣVT=⎛⎜ ⎜⎝u1u2⎞⎟ ⎟⎠(σ100σ2)(vT1vT2)=⎛⎜⎝u11u12u21u22u31u32⎞⎟⎠(σ100σ2)(v11v21v12v22)

Next, split up the matrix Σ

so that

UΣVT=U(σ1000)VT+U(000σ2)VT

and tackle the terms individually. One way to rewrite the first term is

U(σ1000)VT=⎛⎜⎝u11σ10u21σ10u31σ10⎞⎟⎠(v11v21v12v22)=σ1u1vT1,

where the last step follows because the entries highlighted in blue (the second column of the first matrix and the second row of VT

) do not affect the result of the matrix multiplication. A similar calculation shows that the second term is σ2u2vT2, which proves the claim for the given example. Note that in general some of the singular values could be 0, which makes the sum go only up to r=rank(A). □

Let’s apply this fact to X

. We have that

UΣVT=r∑i=1σiuivi=r∑i=1√(n−1)λi1√(n−1)λiXvivTi=Xr∑i=1vivTi=X,

where the last step follows from I=VTV=∑ri=1vivTi

. We have thus established the connection between SVD and PCA.

# A Few Benefits

Let’s explore a few consequences of this correspondence. From the definition of U

we already know that it projects and slightly scales the data onto the principal components. It is more satisfying, however, to derive this result as a general consequence of being able to write X=UΣVT, so let’s turn out attention to that. Multiplying the SVD of X by V

we obtain

XV=UΣVTV=UΣ.

From the left hand side we see that the i

-th column of this matrix is the projection

zi=Xvi=((x1−μ)Tvi,…,(xn−μ)Tvi)T

of the data onto the i

-th PC. From the right hand side we see that this corresponds to zi=uiσi

, which is what we were trying to recover.

We can extract another useful morsel by thinking about the scaling of U

. Recall that each singular value corresponds to the length of a semi-axis of an ellipse that describes the column space of X. By dividing out the singular value (and multiplying by √n−1

to deal with the fact that we started with the covariance) we obtain a transformed dataset which is in some sense spherical:

Y=√n−1UΣ−1.

This process produces a unit variance dataset in a way that can be difficult to achieve by just centering and scaling.

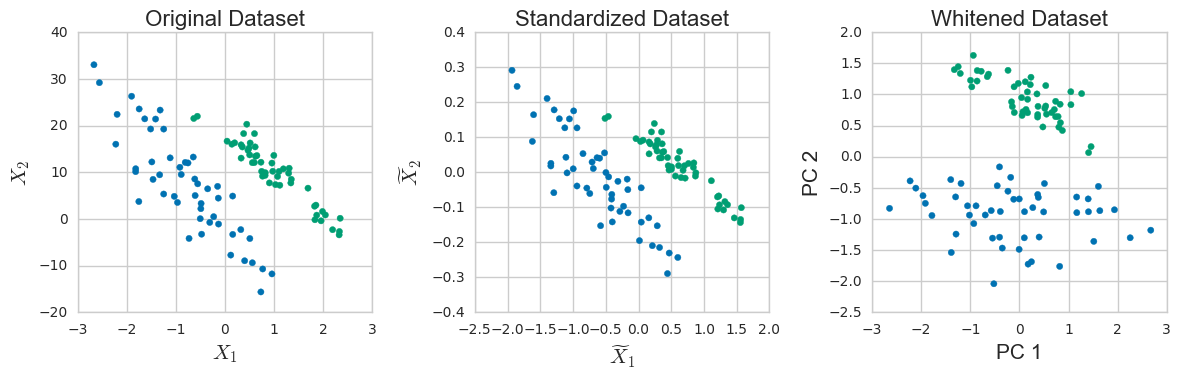


Figure 4: Whitening can be used to easily alleviate the effects of coordinates given in different units. In the left plot you can see that *X2*

is on a different scale than *X1*

. The middle plot depicts the centered and scaled dataset. In the right plot you see the results of whitening.

As a final remark, let’s discuss the numerical advantages of using SVD. A basic approach to actually calculating PCA on a computer would be to perform the eigenvalue decomposition of XTX

directly. It turns out that doing so would introduce some potentially serious numerical issues that could be avoided by using SVD.

The problem lies in the fact that on a computer real numbers are represented with finite precision. Since each number is stored in a finite amount of memory there is necessarily a gap between consecutive representable numbers. For double precision floating point numbers, for instance, the relative error between a real number and its closest floating point approximation is on the order of ε≈10−16

. Algorithms that take this limitation into account are called backward stable. The basic idea is that such an algorithm produces the correct output for an input value that’s within ε

of the value you actually asked about.

If ~σi

is the output of a backward stable algorithm calculating singular values for X and σi

is the true singular value, it can be shown that

|~σi−σi|=O(ε∥X∥),

where ∥X∥

is a reasonable measure of the size of X. On the other hand, a backward stable algorithm that calculates the eigenvalues li of XTX

can be shown to satisfy

|~li−li|=O(ε∥XTX∥)=O(ε∥X∥2),

which when given in terms of the singular values of X

becomes

|~σi−σi|=O(ε∥X∥2σi).

The end result here is that if you are interested in relatively small singular values, e.g. σi≪∥X∥

, working directly with X will produce much more accurate results than working with XTX

. Luckily, SVD lets us do exactly that.

# Conclusion

There is obviously a lot more to say about SVD and PCA. If you’d like to know more about ways to extend these ideas and adapt them to different scenarios, a few of the references that I have found useful while writing this article are [Numerical Linear Algebra](http://bookstore.siam.org/ot50/) by Trefethen and Bau (1997, SIAM) and [Principal Component Analysis](http://www.springer.com/us/book/9780387954424) by Jolliffe (2002, Springer).

If you need a custom data sourcing, transformation and analysis solution, [get in touch with us](https://intoli.com/contact/) to see how our experts can help you make the most out of your data.