Applications of the SVD

> The Singular Value Decomposition is the **"Swiss Army Knife"** and the **"Rolls Royce"** of matrix decompositions.
>
> – Diane O'Leary

Today we will concern ourselves with the "Swiss Army Knife" aspect of the SVD.
Our focus today will be on applications to data analysis.
So our matrices today will be data matrices.
(Rather than thinking of matrices as linear operators).
As a specific example, here is a typical data matrix. This matrix could be the result of measuring a collection of data objects, and noting a set of features for each object.

$$\overbrace{\begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}}^{n \text{ features}} \left\{ \begin{array}{l} \\ \\ \\ \\ \\ \end{array} \right. m \text{ data objects}$$

For example, rows could be people, and columns could be movie ratings.
Or rows could be documents, and columns could be words within the documents.
To start discussing the set of tools that SVD provides for analyzing data, let's remind ourselves what the SVD is.

Recap of SVD

Theorem. Let A be an $m \times n$ matrix with rank r. Then there exists an $m \times n$ matrix Σ whose diagonal entries are the first r singular values of A, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and there exist an $m \times m$ orthogonal matrix U and an $n \times n$ orthogonal matrix V such that

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

Today we'll work exclusively with the reduced SVD.
Here it is again, for the case where A is $m \times n$, and A has rank r.
In that case, the reduced SVD looks like this, with singular values on the diagonal of Σ:

$$m \left\{ \right. \overbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{a_1} & \mathbf{a_2} & \cdots & \mathbf{a_n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}}^{n} = \overbrace{\begin{bmatrix} \vdots & & \vdots \\ \vdots & & \vdots \\ \mathbf{u}_1 & \cdots & \mathbf{u}_r \\ \vdots & & \vdots \\ \vdots & & \vdots \end{bmatrix}}^{r} \times \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \times \begin{bmatrix} \cdots & \cdots & \mathbf{v}_1 & \cdots & \cdots \\ & & \vdots & & \\ \cdots & \cdots & \mathbf{v}_r & \cdots & \cdots \end{bmatrix}$$

$$\overset{m \times n}{A} = \overset{m \times r}{U} \quad \overset{r \times r}{\Sigma} \quad \overset{r \times n}{V^T}$$

Note that for the reduced version, the columns of U and V are orthogonal. This means that:

$$U^T U = I$$

and

$$V^T V = I.$$

(However, U and V are not square in this version, so they are not orthogonal matrices.)

Recall as well, that the route to the SVD starting by asking "What unit vector $\mathbf{x}$ maximizes $\|A\mathbf{x}\|$"?

We found that the answer is $\mathbf{v_1}$, the first row of V^T.

You should be able to see that the SVD of A^T is $V\Sigma U^T$.

So, we can make the corresponding observation that the unit vector that maximizes $\|A^T\mathbf{x}\|$ is $\mathbf{u_1}$, the first column of U.

Approximating a Matrix

To understand the power of SVD for analyzing data, it helps to think of it as a tool for **approximating one matrix by another, simpler, matrix.**

To talk about when one matrix **approximates** another, we need a "length" for matrices.

We will use the **Frobenius norm**.

The Frobenious norm is just the usual vector norm, treating the matrix as if it were a vector.

In other words, the definition of the Frobenius norm of A, denoted $\|A\|_F$, is:

$$\|A\|_F = \sqrt{\sum a_{ij}^2}.$$

The approximations we'll discuss are **low-rank** approximations.

Recall that the rank of a matrix A is the largest number of linearly independent columns of A.

Or, equivalently, the dimension of Col A.

Let's define the **rank-k approximation** to A:

When $k <$ Rank A, the rank-k approximation to A is the closest rank-k matrix to A, i.e.,

$$A^{(k)} = \arg \min_{\text{Rank } B=k} \|A - B\|_F.$$

Why is a rank-k approximation valuable?

The reason is that a rank-k matrix may take up **much** less space than the original A.

$$\overbrace{\text{m}\left\{\begin{bmatrix} \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{a_1} & \mathbf{a_2} & \dots & \mathbf{a_n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}\right.}^{\text{n}} = \overbrace{\begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \\ \sigma_1\mathbf{u}_1 & \sigma_k\mathbf{u}_k \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}}^{k} \times \begin{bmatrix} \dots & \dots & \mathbf{v}_1 & \dots & \dots \\ \dots & \dots & \mathbf{v}_k & \dots & \dots \end{bmatrix}$$

The rank-k approximation takes up space $(m + n)k$ while A itself takes space mn.

For example, if $k = 10$ and $m = n = 1000$, then the rank-k approximation takes space $20000/1000000 =$ 2% of A.

The key to using the SVD for matrix approximation is as follows:

The best rank-k approximation to any matrix can be found via the SVD.

In fact, for an $m \times n$ matrix A, the SVD does two things:

1. It gives the best rank-k approximation to A for **every** k up to the rank of A.
2. It gives the **distance** of the best rank-k approximation $A^{(k)}$ from A for each k.

When we say "best", we mean in terms of Frobenius norm $\|A - A^{(k)}\|_F$,

and by distance we mean the same quantity, $\|A - A^{(k)}\|_F$.

How do we use SVD to find the best rank-k approximation to A?

In terms of the singular value decomposition,

the best rank-k approximation to A is formed by taking

- $U' =$ the k leftmost columns of U,
- $\Sigma' =$ the $k \times k$ upper left submatrix of Σ, and
- $(V')^T =$ the k upper rows of V^T,

and constructing

$$A^{(k)} = U'\Sigma'(V')^T.$$

The distance (in Frobenius norm) of the best rank-k approximation $A^{(k)}$ from A is equal to $\sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$.

Notice that this quantity is summing over the singular values **beyond** k.

What this means is that if, beyond some k, all of the singular values are small, then A **can be closely approximated by a rank-k matrix.**

Signal Compression

When working with measurement data, ie measurements of real-world objects, we find that data is often **approximately low-rank.**

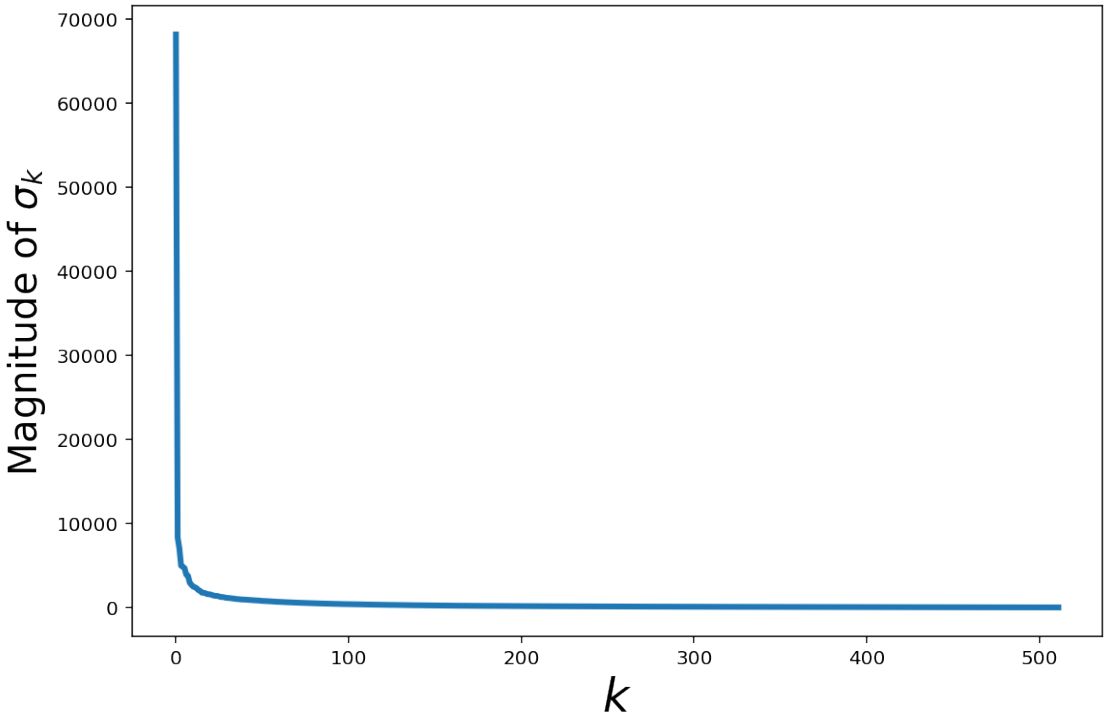
In other words, a matrix of measurements can often be well approximated by a low-rank matrix.

Classic examples include * measurements of human abilities - eg, psychology * measurements of human preferences – eg, movie ratings, social networks * images, movies, sound recordings * genomics, biological data * medical records * text documents

For example, here is a photo.

We can think of this as a 512×512 matrix A whose entries are grayscale values (numbers between 0 and 1).

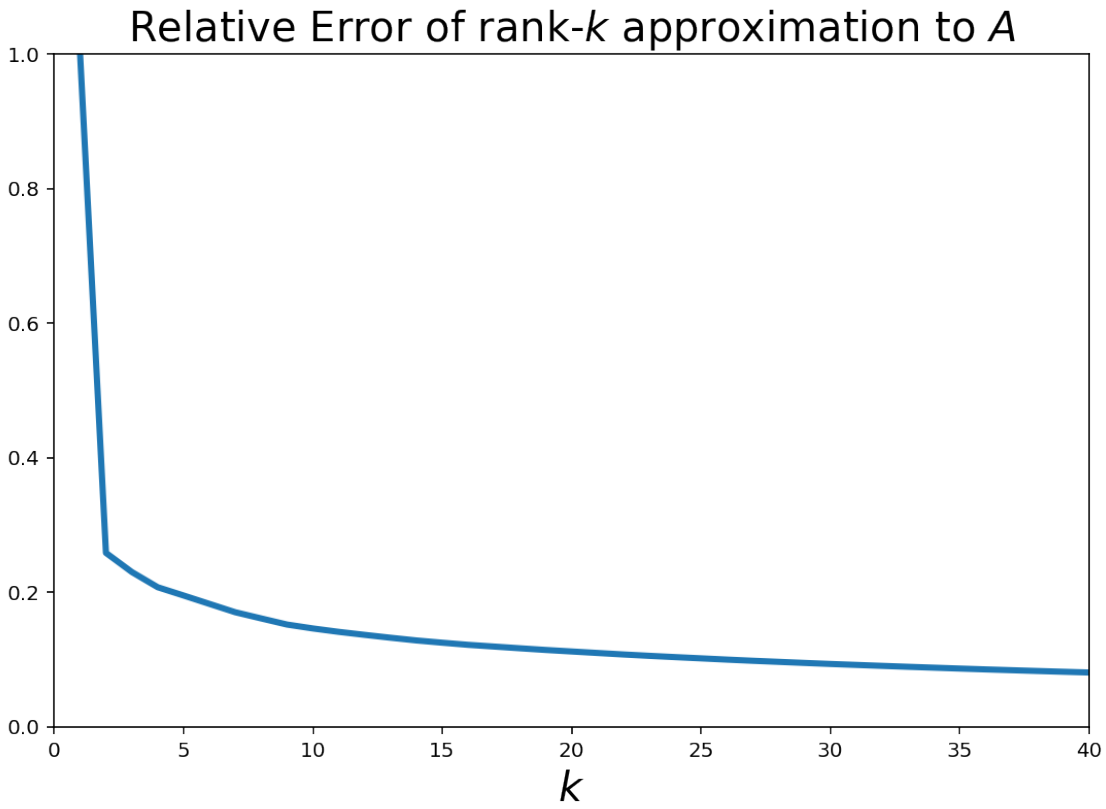
Let's look at the singular values of this matrix.
We compute $A = U\Sigma V^T$ and look at the values on the diagonal of Σ.
This is often called the matrix's "spectrum".



What is this telling us?
Most of the singular values of A are quite small.
Only the first few singular values are large – up to, say, $k = 40$.
Remember that the error we get when we use a rank-k approximation is

$$\sqrt{\sum_{i=k+1}^{r} \sigma_i^2}.$$

So we can use the singular values of A to compute the relative error over a range of possible approximations $A^{(k)}$.



This matrix A has rank of 512.

But the error when we approximate A by a rank 40 matrix is only around 10%.

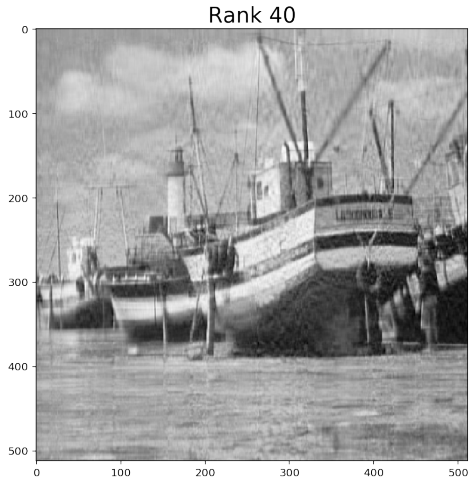
We say that the **effective** rank of A is low (perhaps 40).

Let's find the closest rank-40 matrix to A and view it.

We can do this quite easily using the SVD.

We simply construct our approximation of A using only the first 40 columns of U and top 40 rows of V^T.

[5]:
```
# construct a rank-n version of the boat
u, s, vt = np.linalg.svd(boat, full_matrices=False)
scopy = s.copy()
rank = 40
scopy[rank:]=0
boatApprox = u @ np.diag(scopy) @ vt
#
plt.figure(figsize=(18,9))
plt.subplot(1,2,1)
plt.imshow(boatApprox,cmap = cm.Greys_r)
plt.title('Rank {}'.format(rank), size=20)
plt.subplot(1,2,2)
plt.imshow(boat,cmap = cm.Greys_r)
plt.title('Rank 512', size=20)
plt.subplots_adjust(wspace=0.5);
```



Note that the rank-40 boat takes up only $40/512$ = **8% of the space of the original image!**

This general principle is what makes image, video, and sound compression effective.

When you * watch HDTV, or * listen to an MP3, or * look at a JPEG image,

these signals have been compressed using the fact that they are **effectively low-rank** matrices.

As you can see from the example of the boat image, it is often possible to compress such signals enormously, leading to an immense savings of storage space and transmission bandwidth.

In fact the entire premise of the show "Silicon Valley" is based on this fact :)

Dimensionality Reduction

Another way to think about what we just did is "dimensionality reduction".

Consider this common situation:

$$m \text{ objects} \left\{ \overbrace{\begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{bmatrix}}^{n \text{ features}} \right. = \overbrace{\begin{bmatrix} \vdots & & \vdots \\ \vdots & & \vdots \\ \mathbf{u}_1 & \cdots & \mathbf{u}_k \\ \vdots & & \vdots \\ \vdots & & \vdots \end{bmatrix}}^{k} \times \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix} \times \begin{bmatrix} \dots & \dots & \mathbf{v}_1 & \dots \\ & & \vdots & \\ \dots & \dots & \mathbf{v}_k & \dots \end{bmatrix}$$

The U matrix has a row for each data object.

Notice that the original data objects had n features, but each row of U only has k entries.

Despite that, a row of U can still provide most of the information in the corresponding row of A

(To see that, note that we can approximately recover the original row by simply multiplying the row of U by ΣV^T).

So we have **reduced the dimension** of our data objects – from n down to k – without losing much of the information they contain.

Principal Component Analysis

This kind of dimensionality reduction can be done in an **optimal** way.

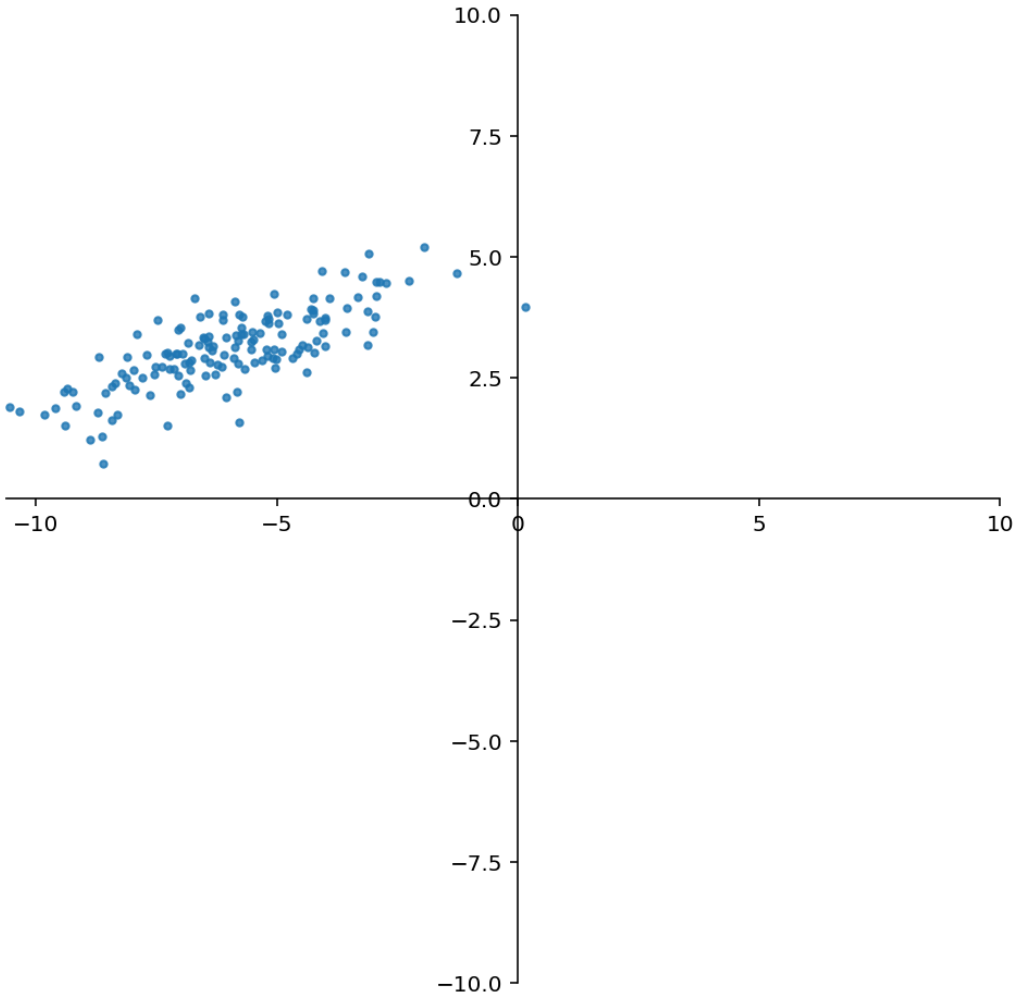
The method for doing it is called **Principal Component Analysis** (or PCA).

What does **optimal** mean in this context?

Here we use a statistical criterion: a dimensionality reduction that captures the maximum **variance** in the data.

Here is a classic example.

Consider the points below, which live in $\mathbb{R}^2$.



Now, although the points are in $\mathbb{R}^2$, they seem to show effective low-rank.

That is, it might not be a bad approximation to replace each point by a point in a 1-D dimensional space, that is, along a line.

What line should we choose? We will choose the line such that the **sum of the distances of the points to the line is minimized.**

The points, projected on this line, will capture the maximum variance in the data (because the remaining errors are minimized).

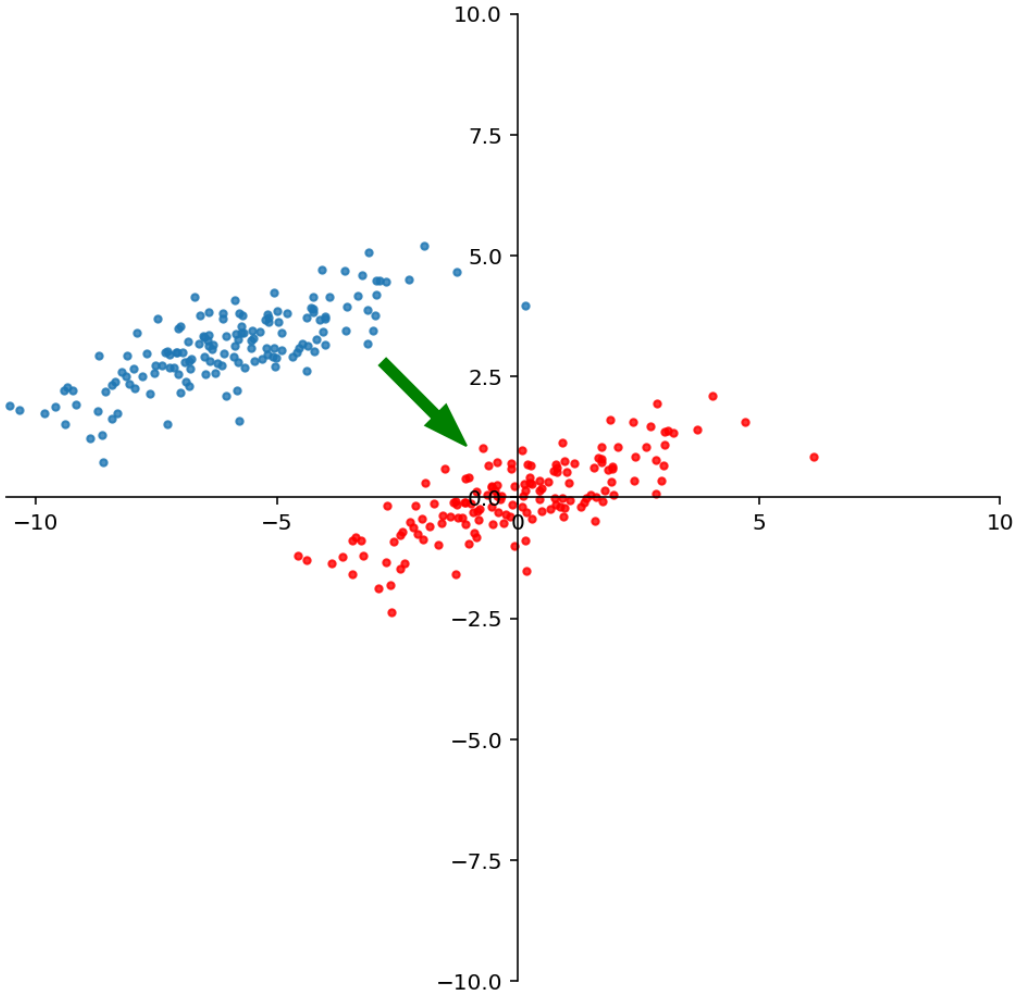
What would happen if we used SVD at this point, and kept only rank-1 approximation to the data?

This would be the 1-D **subspace** that approximates the data best in Frobenius norm.

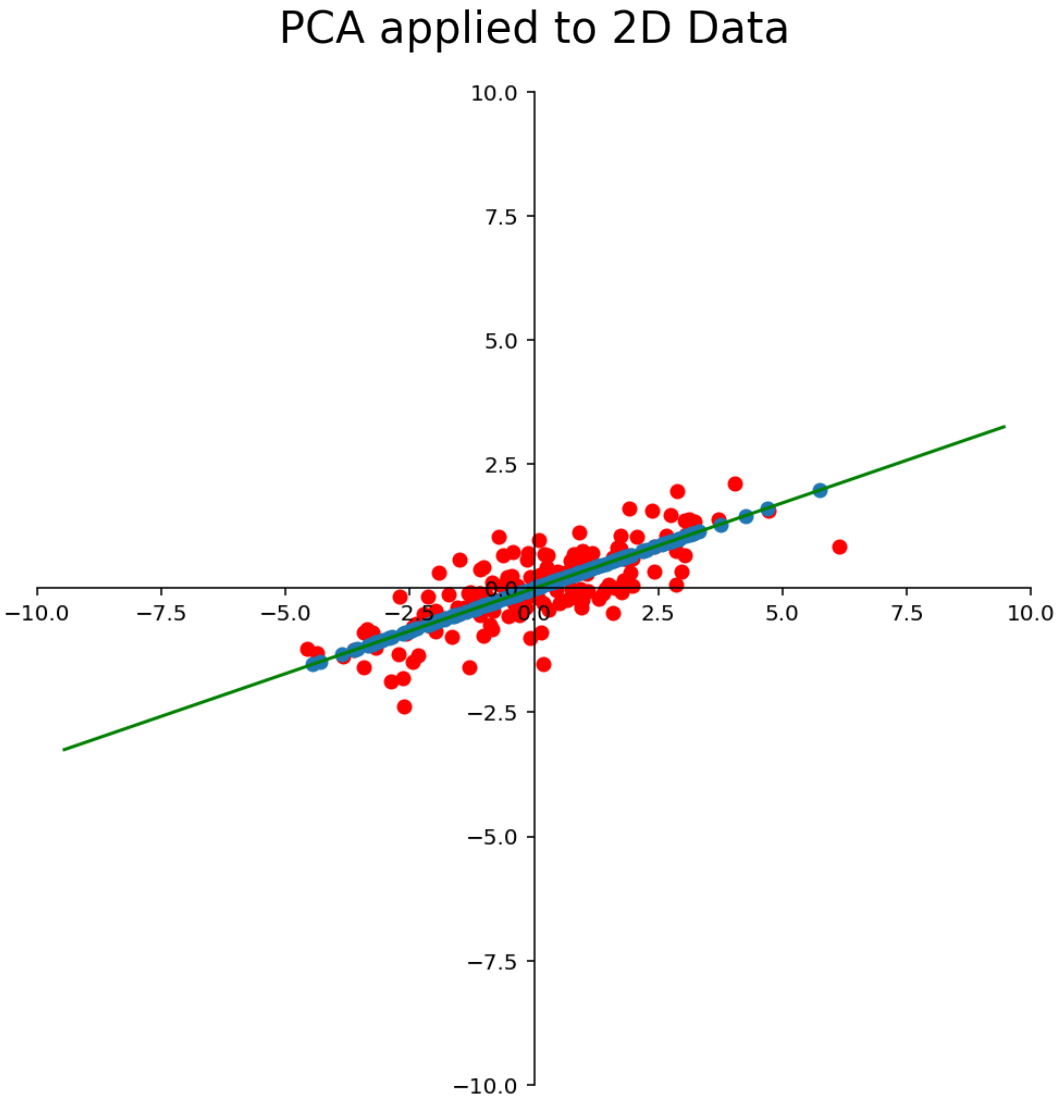
However the variance in the data is defined with respect to the data mean, so we need to mean-center the data first, before using SVD.

That is, without mean centering, SVD finds the best 1-D subspace, not the best line though the data (which might not pass through the origin).

So to capture the best line through the data, we first move the data points to the origin:



Now we use SVD to construct the best 1-D approximation of the mean-centered data:



This method is called **Principal Component Analysis.**

In summary, PCA consists of:

1. Mean center the data, and
2. Reduce the dimension of the mean-centered data via SVD.

It winds up constructing the **best low dimensional approximation of the data** in terms of variance.

This is equivalent to projecting the data onto the subspace that captures the maximum variance in the data.

That is, each point is replaced by a point in k dimensional space such that the total error (distances between points and their replacements) is minimized.

Visualization using PCA

I'll now show an extended example to give you a sense of the power of PCA.

Let's analyze some really high-dimensional data: **documents.**

A common way to represent documents is using the bag-of-words model.

In this matrix, rows are documents, columns are words, and entries count how many time a word appears in a document.

This is called a *document-term matrix.*

$$\overbrace{\phantom{\begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \end{bmatrix}}}^{n \text{ terms}}$$
$$m \text{ documents} \left\{ \begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{bmatrix} \right.$$

We are touching on a broad topic, called Latent Semantic Analysis, which is essentially the application of linear algebra to document analysis.

You can learn about Latent Semantic Analysis in other courses in data science or natural language processing.

Our text documents are going to be posts from certain discussion forums called "newsgroups".

We will collect posts from three groups: `comp.os.ms-windows.misc`, `sci.space`, and `rec.sport.baseball`.

I am going to skip over some details. However, all the code is in this notebook, so you can explore it on your own if you like.

```
[9]: from sklearn.datasets import fetch_20newsgroups
     categories = ['comp.os.ms-windows.misc', 'sci.space', 'rec.sport.baseball']
     news_data = fetch_20newsgroups(subset='train', categories=categories)
     from sklearn.feature_extraction.text import TfidfVectorizer
     vectorizer = TfidfVectorizer(stop_words='english', min_df=4, max_df=0.8)
     dtm = vectorizer.fit_transform(news_data.data).todense()
```

```
[10]: print('The size of our document-term matrix is {}'.format(dtm.shape))
```

```
The size of our document-term matrix is (1781, 9409)
```

So we have 1781 documents, and there 9409 different words that are contained in the documents. We can think of each document as a vector in 9409-dimensional space.

Let us apply PCA to the document-term matrix.

Our data matrix is in sparse form (most entries are zero).

First, we mean center the data.

Note that `dtm` is a sparse matrix, but once it is mean centered it is not sparse any longer.

```
[11]: centered_dtm = dtm - np.mean(dtm, axis=0)
```

Now we compute the SVD of the mean-centered data:

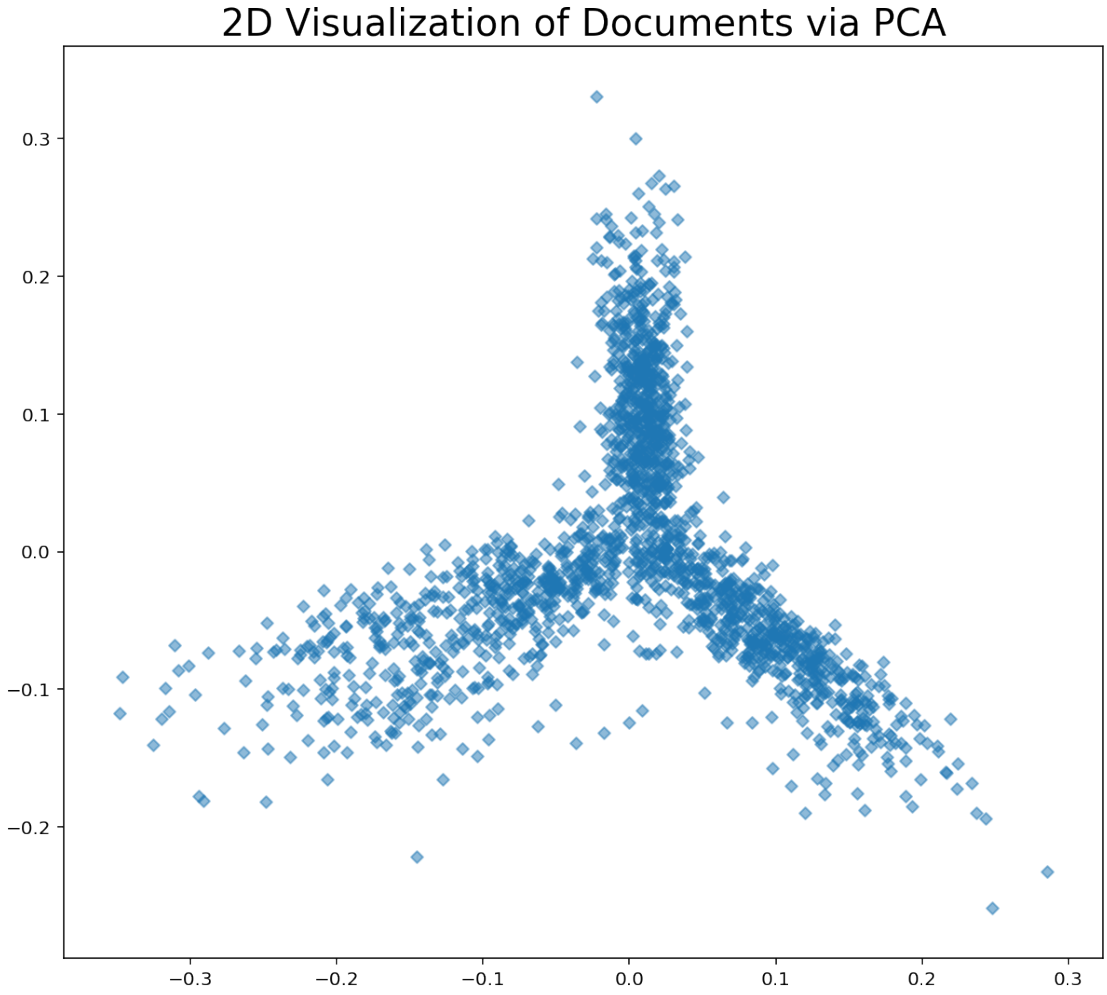
```
[12]: u, s, vt = np.linalg.svd(centered_dtm)
```

Now, we use PCA to visualize the set of documents.

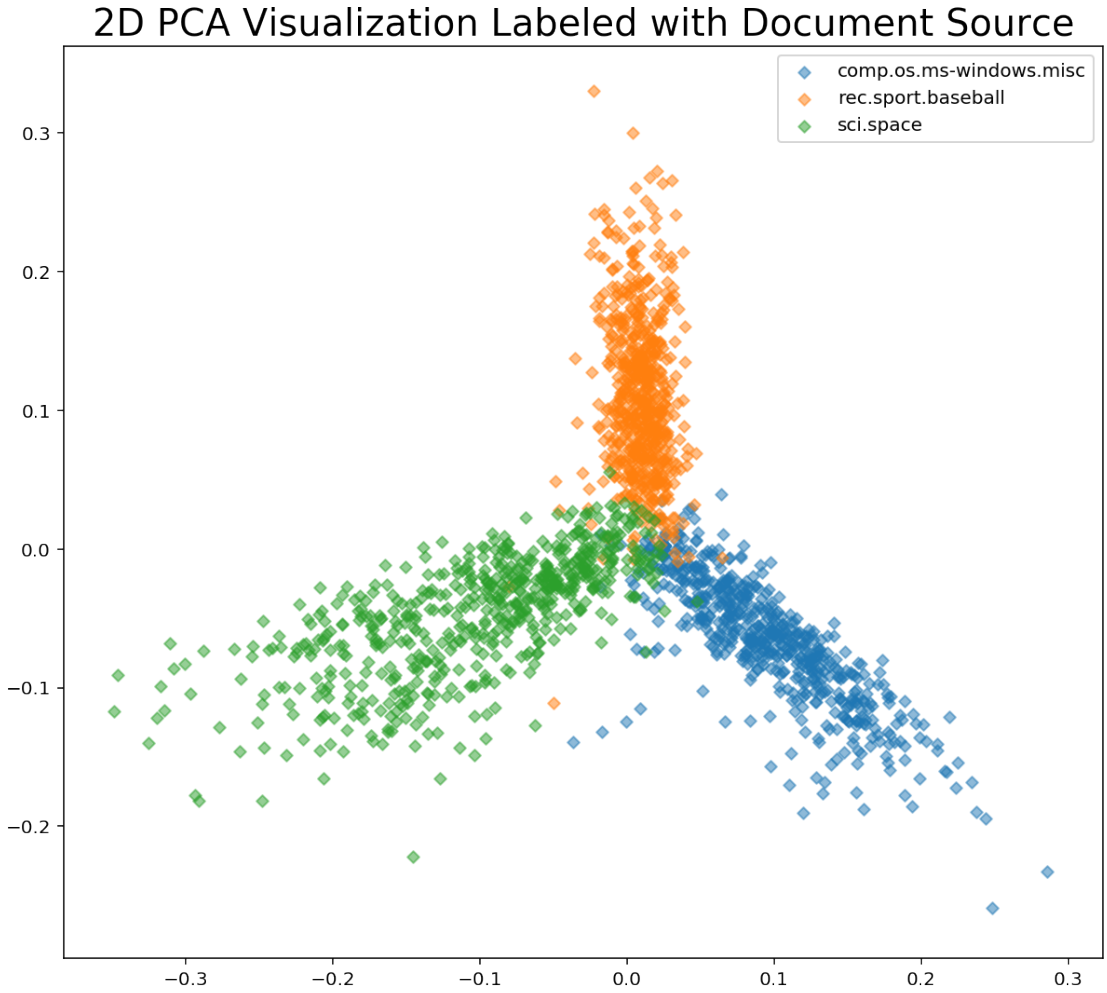
Our visualization will be in two dimensions.

This is pretty extreme – we are taking points in 9409-dimensional space and projecting them into a space of only two dimensions!

```
[13]: Xk = u @ np.diag(s)
      fig, ax = plt.subplots(1,1,figsize=(10,9))
      plt.scatter(np.ravel(Xk[:,0]), np.ravel(Xk[:,1]),
                          s=20, alpha=0.5, marker='D')
      plt.title('2D Visualization of Documents via PCA', size=20);
```



This visualization shows that our collection of documents has considerable internal structure.
In particular, based on word frequency, it appears that there are three general groups of documents.
As you might guess, this is because the discussion topics of the document sets are different:



Wrapup

We have reached the end!

Of course, this is not really the end ... more like the beginning.

If we had more time, we'd talk about linear algebra informs the study of graphs, the methods of machine learning, data mining, and many more topics.

So this is just where we have to stop.

We have looked at the richness of linear algebra from many angles.

We have seen that the simple linear system $A\mathbf{x} = \mathbf{b}$ leads to a whole collection of interesting questions, questions that have unfolded step by step over the course of the semester.

But we have also seen that we can extract the idea of matrix out of a linear system, and consider it as an object in its own right.

Considered on their own, matrices can be seen as linear operators, giving us tools for computer graphics and the solution of dynamical systems and linear equations.

We have also seen that matrices can be seen as data objects, whose linear algebraic properties expose useful facts about the data.

There are many courses you can go on to from here, which will rely on your understanding of linear algebra:

- CS 391 Fundamentals of Data Science
- CS 440 Artificial Intelligence
- CS 470 Computer Systems Performance Analysis
- CS 480 Computer Graphics
- CS 505 Intro to Natural Language Processing
- CS 506 Tools for Data Science
- CS 530 Advanced Algorithms
- CS 531 Advanced Optimization Algorithms
- CS 533 Spectral Methods
- CS 558 Machine Learning
- CS 565 Data Mining
- CS 581 Computational Fabrication
- CS 591 Deep Learning
- CS 591 Compressive Sensing
- CS 591 Natural Language Understanding

In each of these you will use and build on your knowledge of linear algebra.
Enjoy!