Linear Algebra for Machine Learning

Disclaimer: Work in progress. Portions of these written materials are incomplete.

A bird's-eye view of **basic linear algebra** for machine learning

Introductory Level

Never taken linear algebra? Or know a little about the basics? Want to get a feel for how it's used in ML? *Then this video is for you.*

Non-technical

This is not a technical deep-dive.

Primary Goal

Hope to whet your appetite to learn more!

What’s on the menu for today?

Agenda

Data Representations

Use basic ideas of linear algebra to represent data in a way that computers can understand: **vectors.**

Vector Embeddings

Learn ways to choose these representations wisely via **matrix factorizations**.

Dimensionality Reduction

Deal with large-dimensional data using linear maps and their **eigenvectors and eigenvalues**.

Data Representations

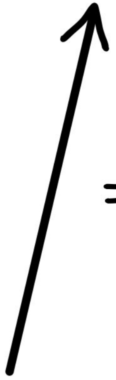
How can we represent **data** (images, text, user preferences, etc.) in a way that computers can understand?

Idea:

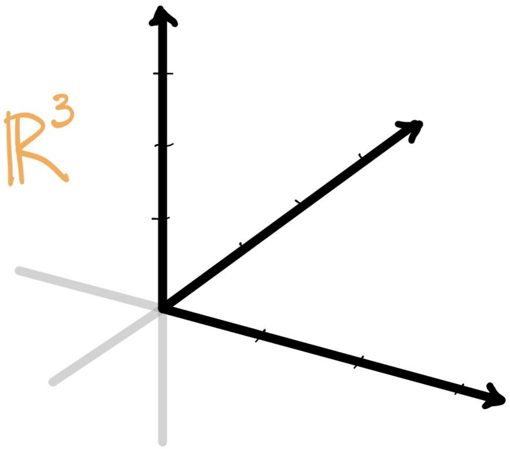
Organize information
into a **vector**

A **vector** is a 1-dimensional array of numbers. It has both a *magnitude* (length) and a *direction*.

The totality of all vectors with *n* entries is an ***n*-dimensional vector space**.

$v =$  $= \begin{bmatrix} -3 \\ 0.7 \\ 2 \end{bmatrix}$

$\mathbb{R}^3$



"3-dimensional space" consists of all vectors with 3 entries:

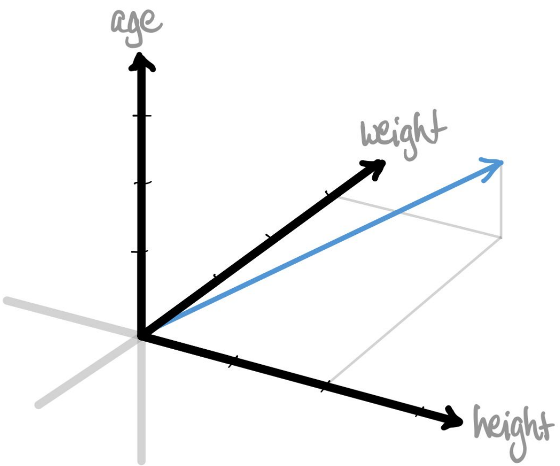
$$\begin{bmatrix} * \\ * \\ * \end{bmatrix}$$

In the context of machine learning…

A **feature vector** is a vector whose entries represent the "features" of some object.

$$p = \begin{bmatrix} 64 \\ 131 \\ 23 \end{bmatrix} \begin{matrix} \text{height} \\ \text{weight} \\ \text{age} \end{matrix}$$

"p" for "patient"

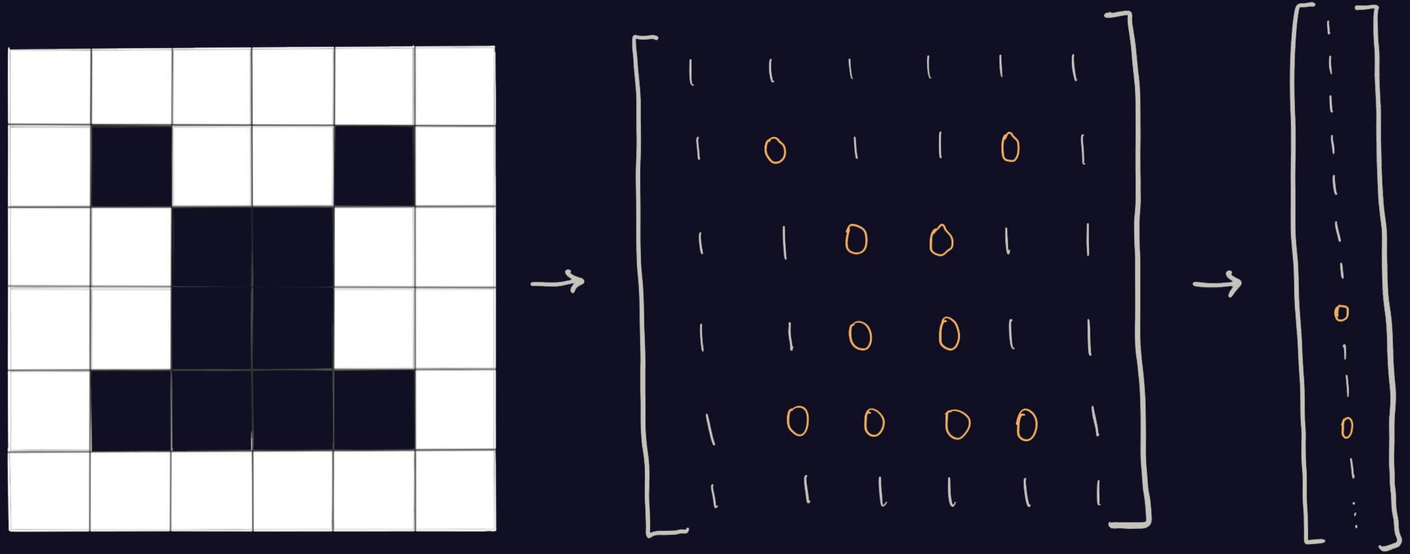


The vector space containing them is called **feature space**.

Common examples in machine learning

Images

In black and white images, **black and white pixels** correspond to 0s and 1s. Grayscale pixels are numbers between 0 and 255. Both assemble into a 1-dimensional array of numbers.



Words and Documents

Given a collection of documents (e.g. Wikipedia articles), assign to every word a vector whose i^{th} entry is the number of times the word appears in the i^{th} document.

$$|\text{dog}\rangle = \left[\begin{array}{c} 0 \\ 7 \\ 0 \\ 0 \\ 51 \\ \vdots \\ 0 \end{array}\right] \begin{array}{l} \text{wiki \#1} \\ \text{wiki \#2} \\ \text{wiki \#3} \\ \text{wiki \#4} \\ \text{wiki \#5} \\ \vdots \\ \text{wiki \# 54,000,000} \end{array}$$

Tip: These vectors can assemble into a large matrix, useful for **latent semantic analysis.**

Yes/No or Ratings

Given users and items (e.g. movies), vectors can indicate if a user has interacted with the item (1 = yes, 0 = no) or the user's ratings, say a number between 0 and 5.

$$\text{user 1} = \begin{array}{cl} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} & \begin{array}{l} \text{no} \\ \text{yes} \\ \text{no} \\ \text{no} \\ \vdots \\ \text{yes} \\ \text{no} \end{array} \end{array} \quad \underline{\text{or}} \quad \text{user 4} = \begin{array}{cl} \begin{bmatrix} 0 \\ 5 \\ 0 \\ 3 \\ \vdots \\ 0 \\ 2 \end{bmatrix} & \begin{array}{l} ? \\ \text{love} \\ ? \\ \text{like} \\ \vdots \\ ? \\ \text{dislike} \end{array} \end{array}$$

What about non-numerical data, such as language?

“One-Hot Encodings”

Assign to each word a vector with one 1 and 0s elsewhere. This is called a one-hot encoding (or a “standard basis vector”). For example, suppose our language only has four words:

$$\text{apple} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \text{cat} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \text{house} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \text{tiger} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Now imagine having *tens of thousands* of words....

Some drawbacks to consider:

- These vectors can be very **sparse**
  - A “sparse” vector is one with lots of zeros
  - 500,000 users and 1,000,000 movies

- Possible **lack of meaningful relationships** between vectors
  - One-hot encodings are never “similar.”
  - Similarity is measured by the *dot product*.

What’s the
“dot product”?

The dot product

Just like we can multiply numbers, we can also multiply vectors. This is called the **dot product**.

The product of numbers is another number.
The dot product of vectors is *not* another vector!

$$2 \times 5 = 10$$

numbers

vs

$$\begin{bmatrix}2\\1\end{bmatrix} \cdot \begin{bmatrix}1\\5\end{bmatrix} = 7$$

vectors

a number

The dot product

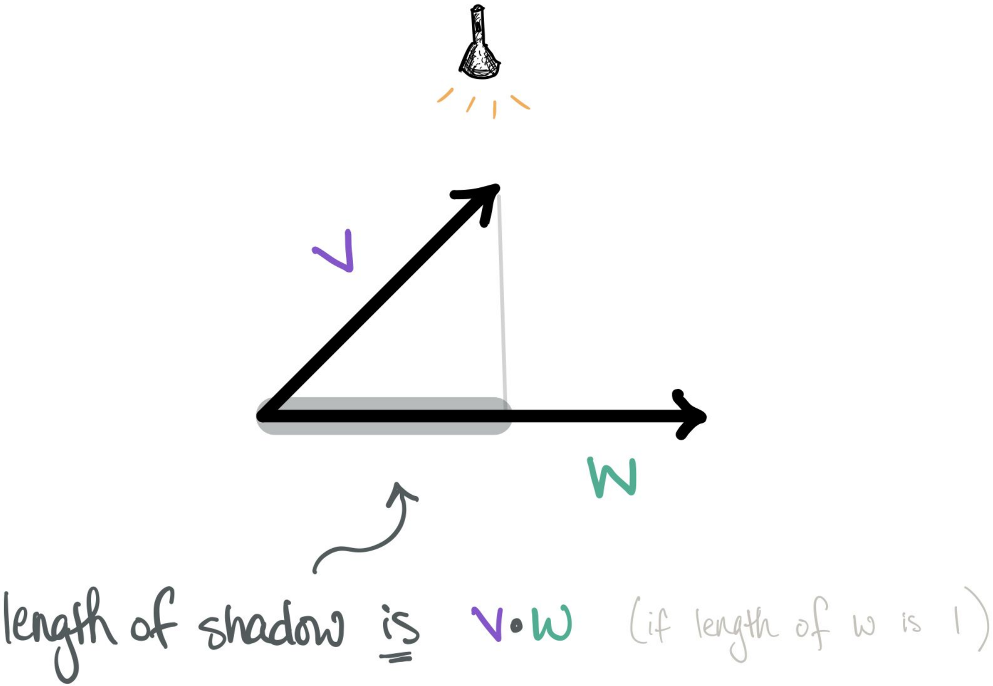
$$\begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 2 \\ -1 \end{bmatrix} = (1)(7) + (0)(2) + (3)(-1) = 4$$

What's the intuition?

The dot product represents the length of the "shadow" of one vector along another.

(Basic trigonometry)

This indicates how similar the two vectors are.



Here’s something to think about:

Dot products between words are zero,
no matter how “similar” they are.

$$\text{apple} \cdot \text{cat} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = 0 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \text{tiger} \cdot \text{cat}$$

Solution?

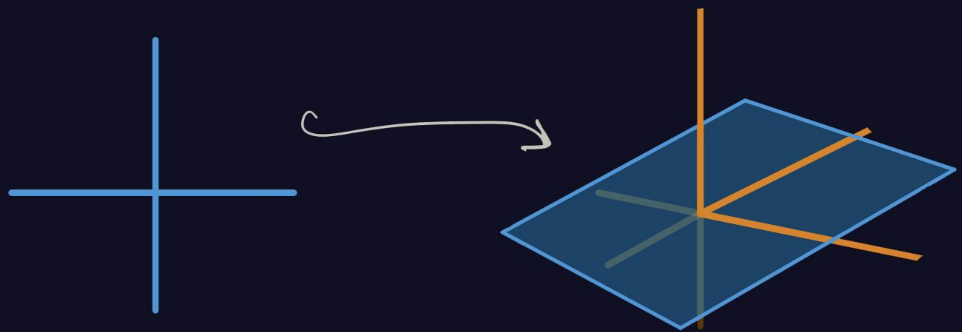
Look for better vector representations.

Vector Embeddings

Idea: An **embedding** of a vector is another vector in a smaller dimensional space.

replace $\begin{bmatrix} * \\ * \\ * \end{bmatrix}$ with $\begin{bmatrix} * \\ * \end{bmatrix}$

2D "embeds" into 3D



How to *find* embeddings?

Here are two ways:

1. Matrix factorizations

2. Neural networks

Matrix Factorizations

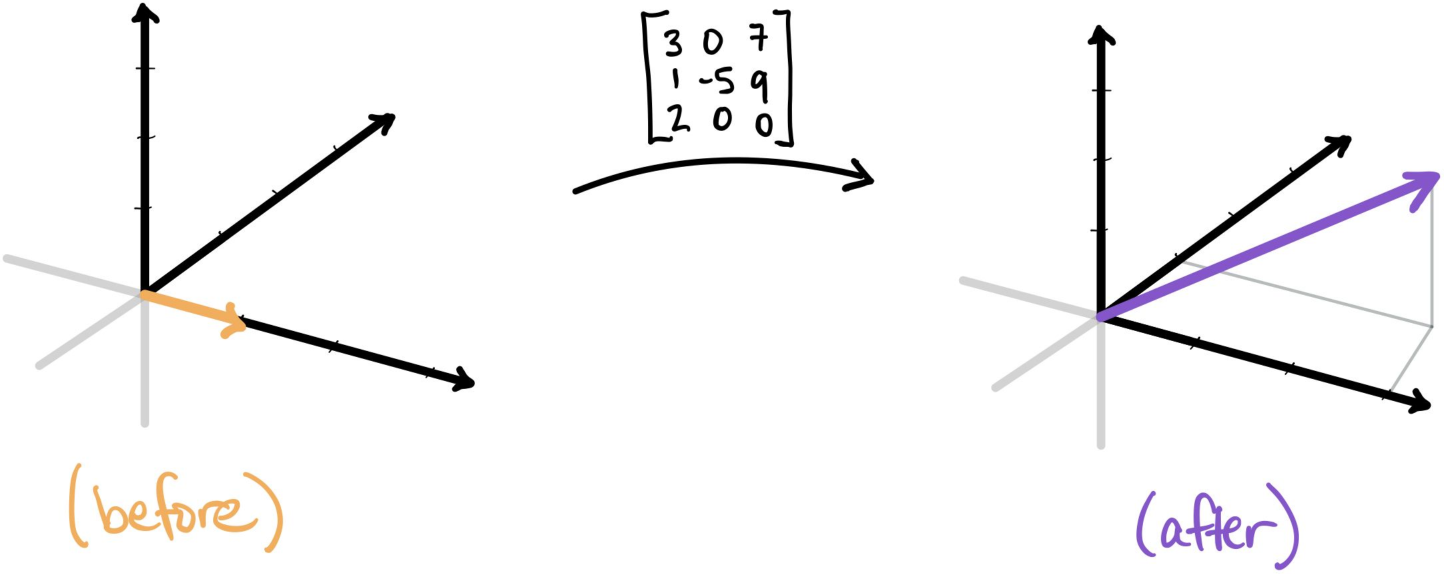
Let’s talk about **matrices** first.

A **matrix** is a 2-dimensional array of numbers. It represents a particular process of turning one vector into another: stretching, rotating, scaling, or something more complex.

$$\begin{bmatrix} 3 & 0 & 7 \\ 1 & -5 & 9 \\ 2 & 0 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{\text{input}} = \underbrace{\begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}}_{\text{output}}$$

More generally...

A matrix represents a **transformation** of an *entire* vector space to another (possibly of different dimensions.)



What is a “**matrix factorization**”?

Even if this phrase is new, the concept is familiar. To see this, let’s think back to numbers.

Earlier, we discussed multiplication:

- We can multiply **numbers** and get **number**.
- We can multiply **vectors** and get **number**.

Now:

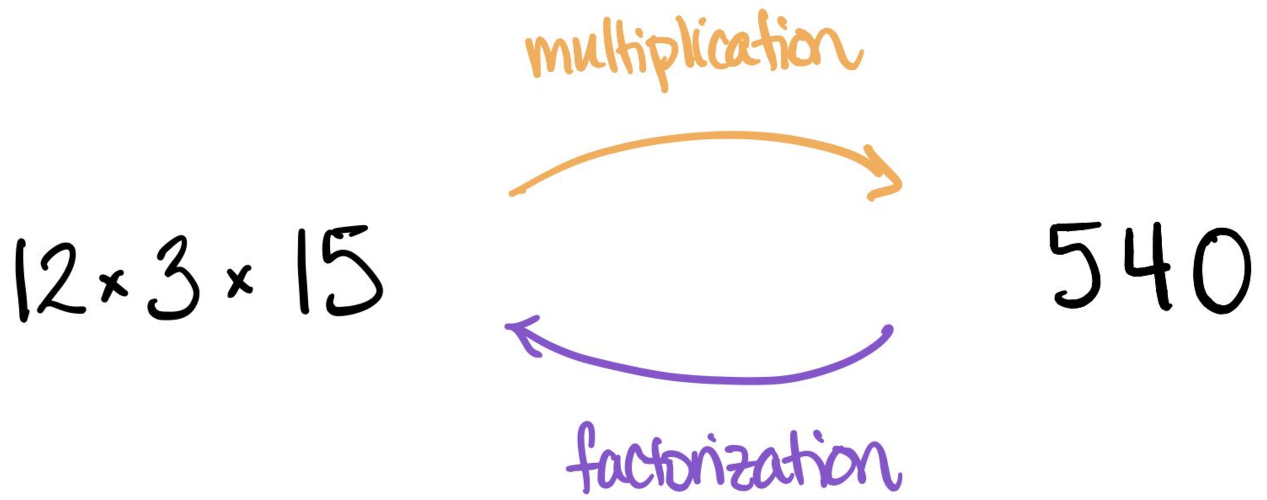
- We can multiply **matrices** and get a **matrix**.

$$\begin{bmatrix} 3 & 0 & 7 \\ 1 & -5 & 9 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -7 & 6 \\ -14 & 2 \\ 0 & 4 \end{bmatrix}$$

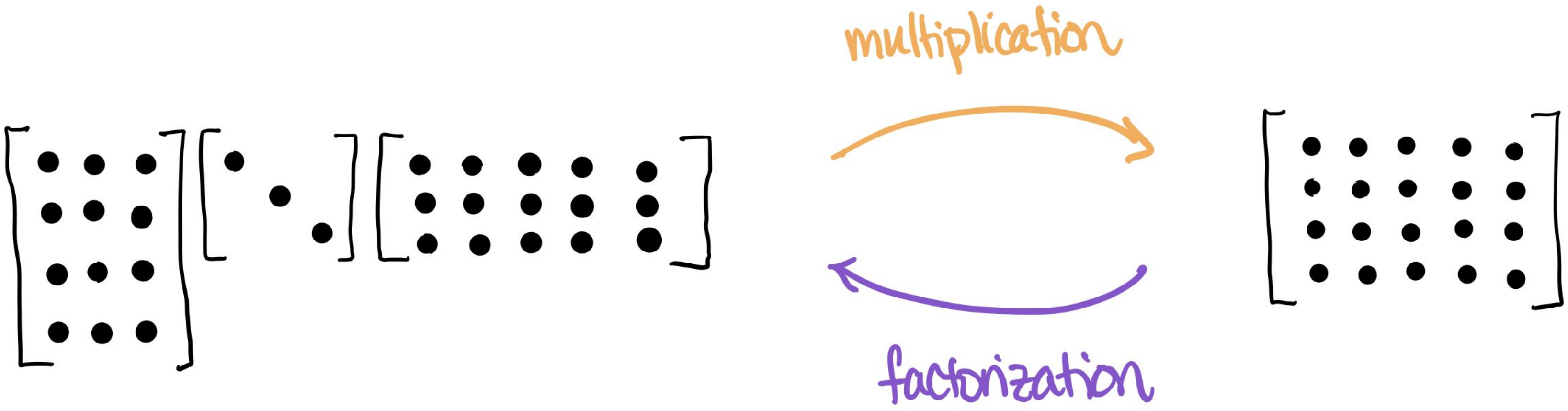
$$3 \times 3 \qquad 3 \times 2 \qquad\qquad 3 \times 2$$

sizes must match

Multiplication can be “undone.” This is called factorization.



Analogously, matrices can also be factored.



In general, factorization is **hard.**

Multiplying numbers is easier than trying to factor them. Modern cryptography is based on the difficulty of factorization.

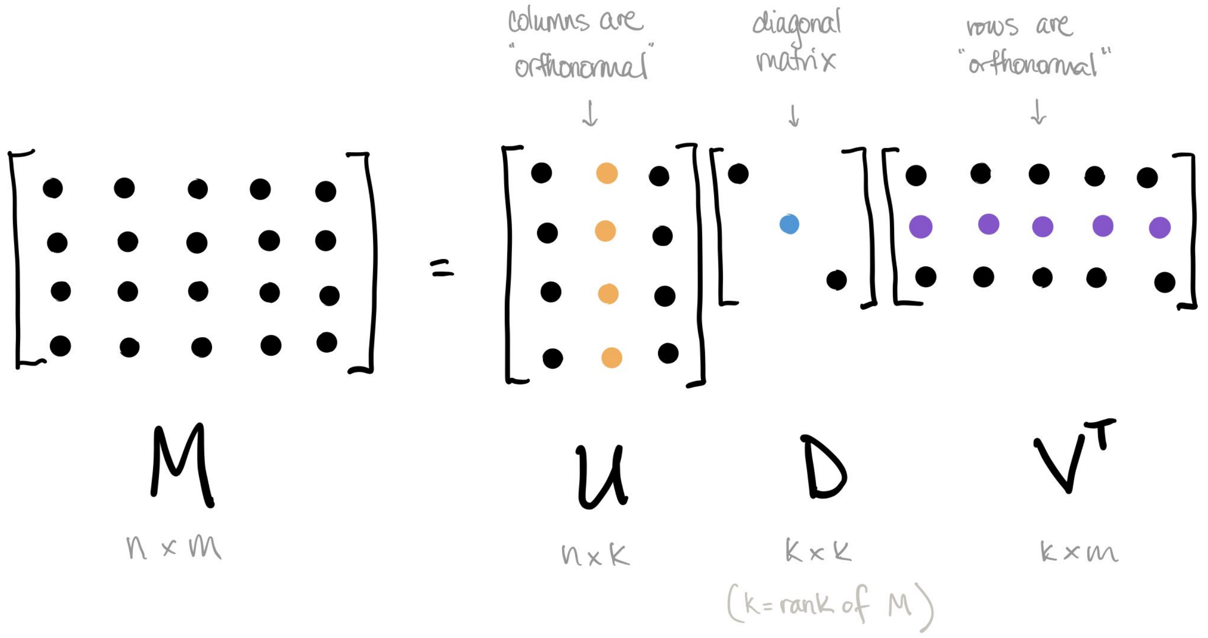
There is a nice result for matrices:

Every matrix can be factored!

This is a fundamental theorem in linear algebra.

Singular Value Decomposition

Every $n \times m$ matrix can be written as a product of three smaller matrices as below. This is called **singular value decomposition (SVD)**.



Singular Value Decomposition

SVD appears in *lots* of places:

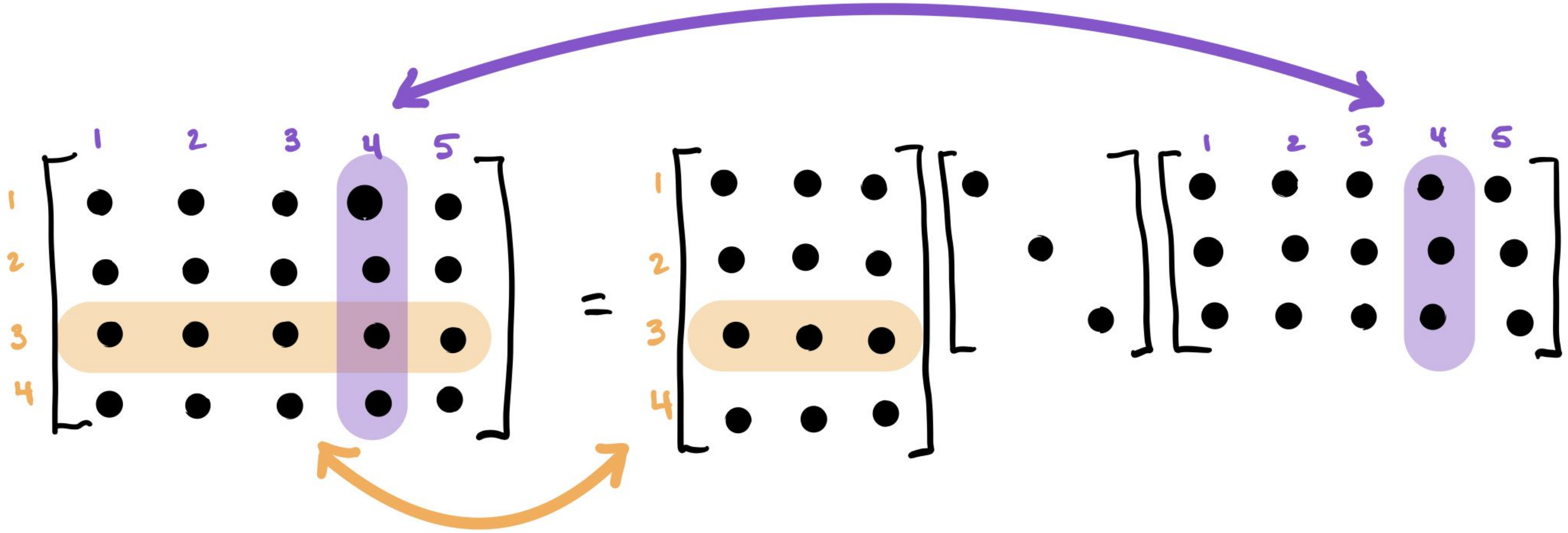
- Statistics (principal component analysis)
- Chemical physics
- Image processing
- Genomics
- Robotics
- Quantum physics (entanglement)
- Data embeddings / **vector embeddings**

For more, see "The Extraordinary SVD" by Martin and Porter (2012).

How do matrix factorizations give us vector embeddings?

Short answer

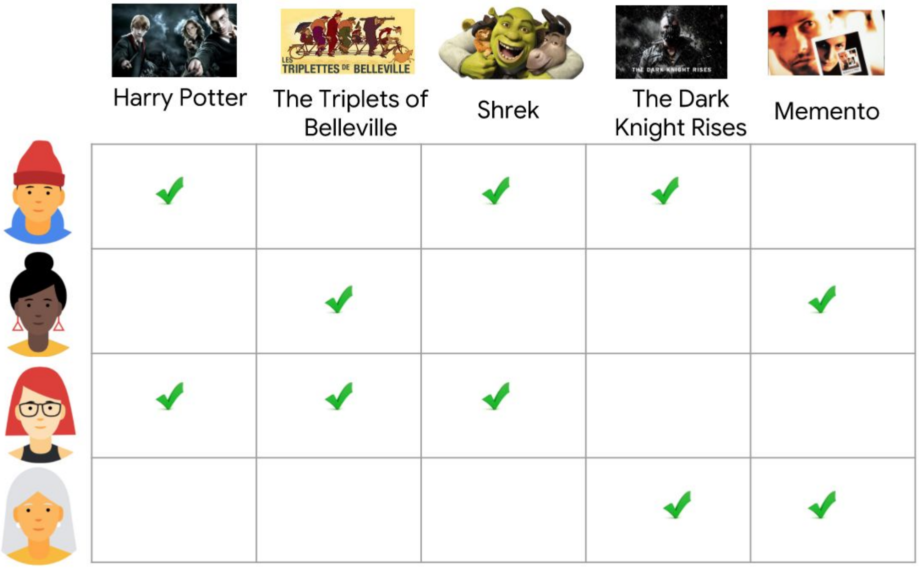
The columns/rows of the factors are candidates for embeddings.



Example

Think back to users and movies.

Imagine a **user-by-movie matrix** obtained by stacking users' data vectors side by side. (Checkmarks are 1s and empty cells are 0s or negatives.)



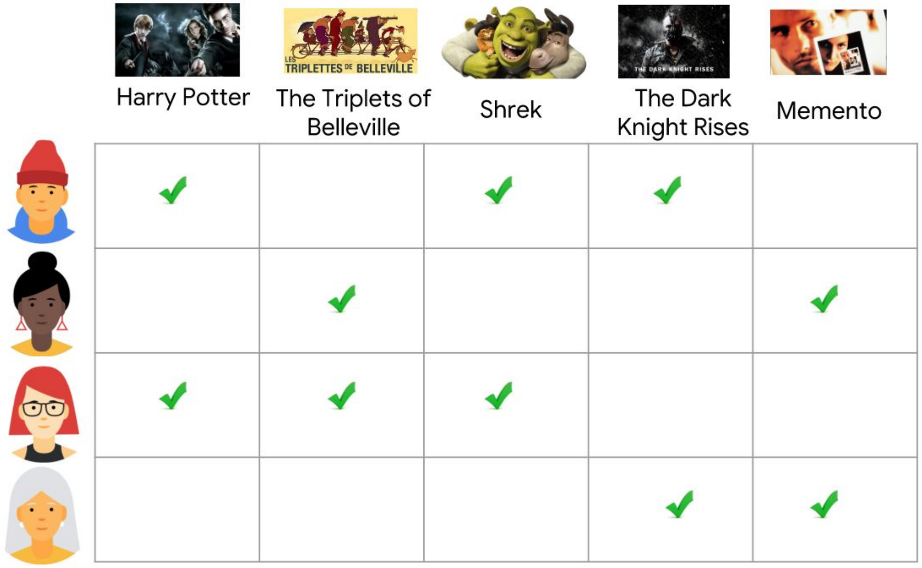
Images from Google Developers ML Crash Course on Collaborative Filtering.

For example, user #3 and *Shrek* correspond to these vectors:

$$\text{user3} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \text{Shrek} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

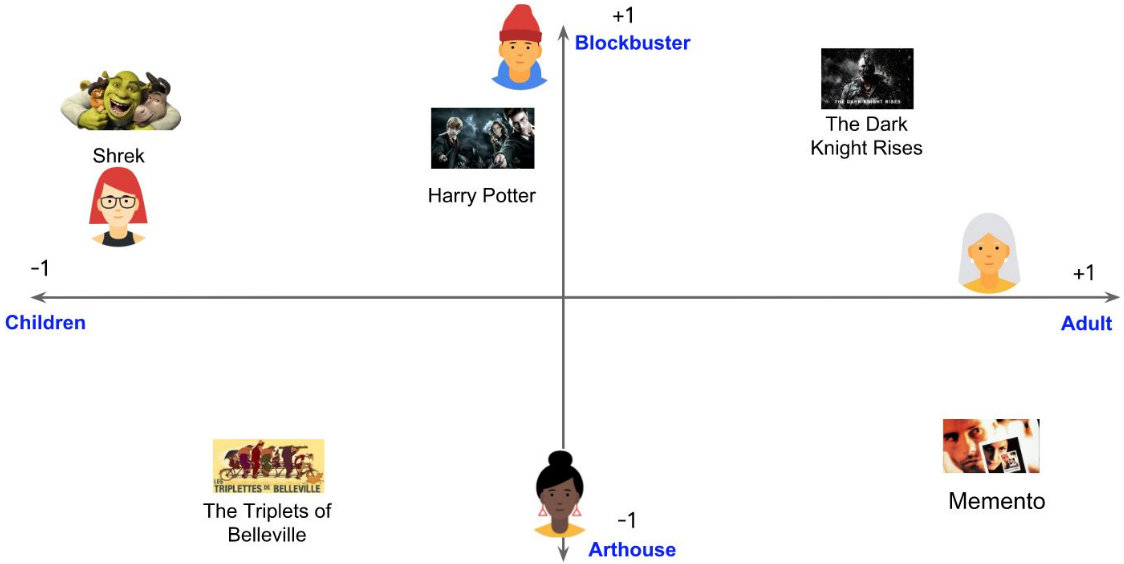
There are **features** in the data.

- Some users are **kids**, others are adults.
- Some movies are for kids, others are for adults.
- Some movies are **blockbusters**, others are arthouse movies.



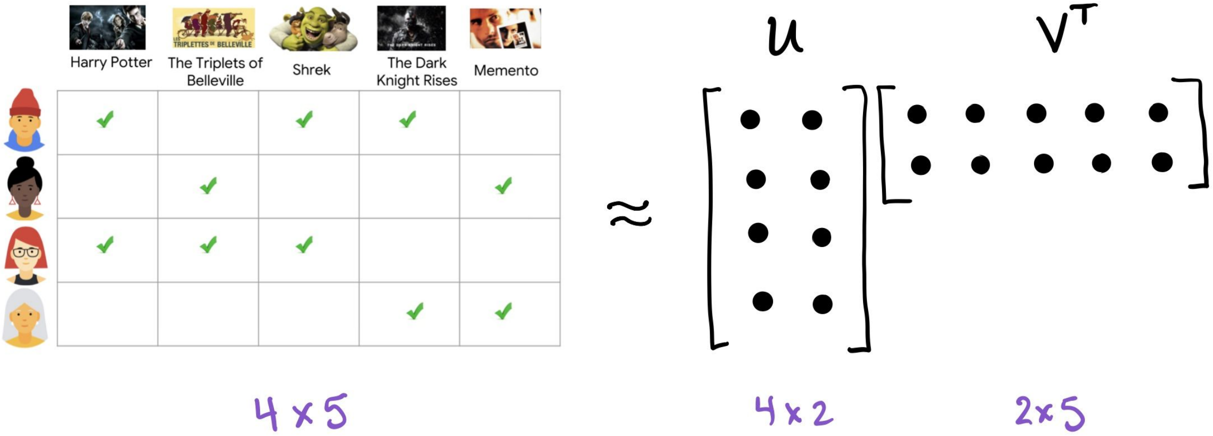
Goal

Find *new*, smaller-dimensional vector representations that capture these features. For example, a 2-dimensional “latent” space.



How?

Use **SVD-like techniques** to find “smaller” matrices *U* and *V* whose product is *close* to the original matrix. Columns and rows of *U* and *V* are candidates for embeddings.



For instance, see the Netflix Grand Prize and *Matrix Factorization Techniques for Recommender Systems* (2009) by Koren, Bell, and Volinsky.

V^T is a feature × movie matrix

	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

≈

V^T:

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

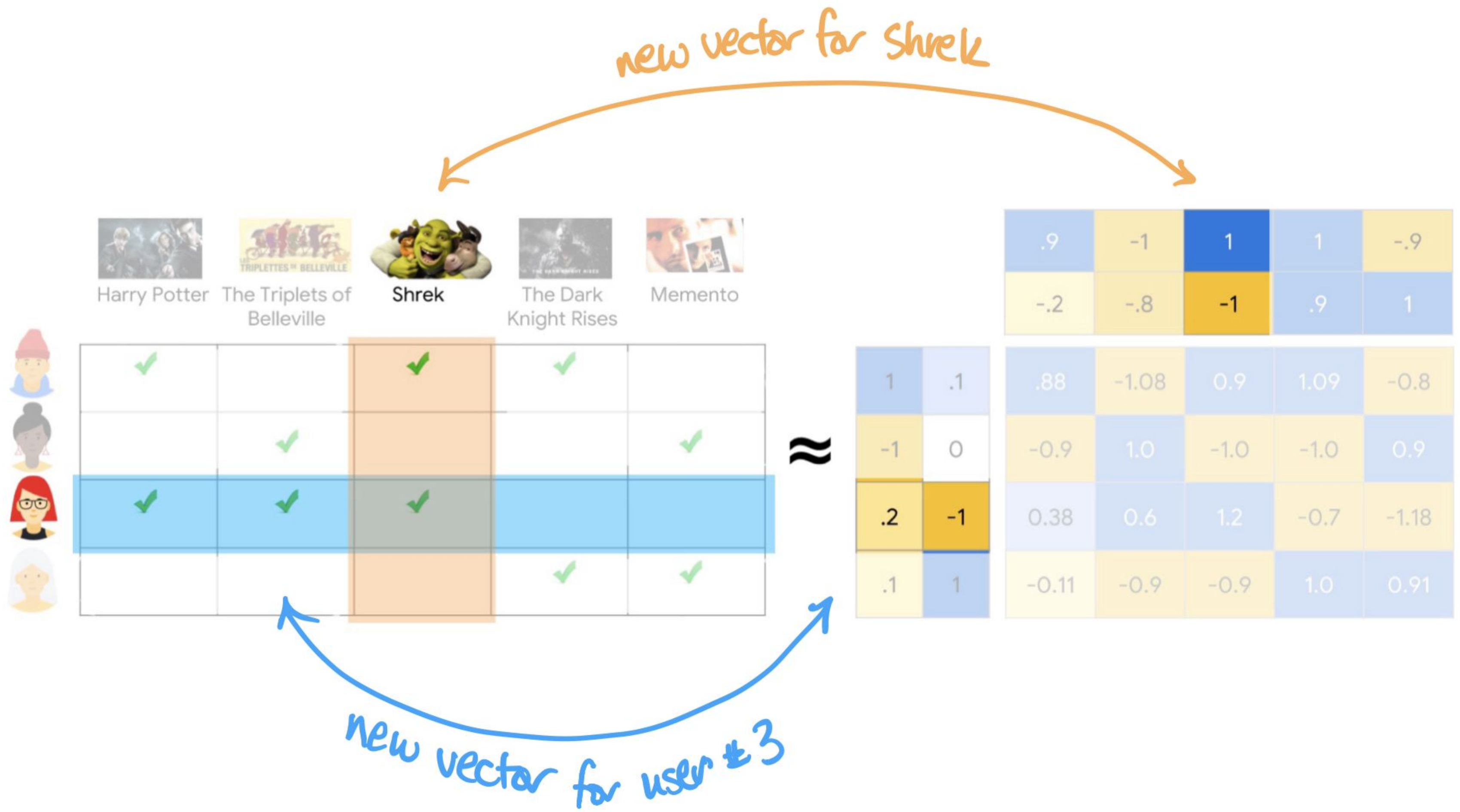
U:

1	.1
-1	0
.2	-1
.1	1

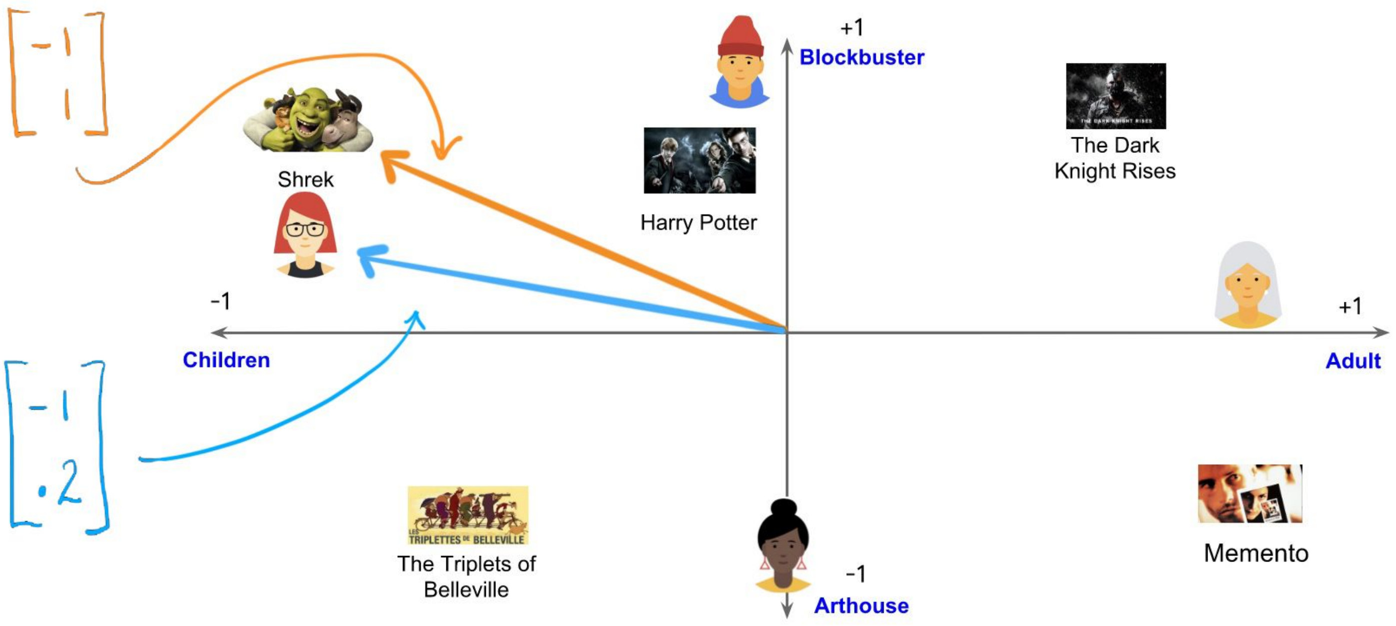
UV^T:

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

U is a user × feature matrix


new vector for shrek
Harry Potter
The Triplets of Belleville
Shrek
The Dark Knight Rises
Memento
≈
.9 -1 1 1 -.9
-.2 -.8 -1 .9 1
1 .1
-1 0
.2 -1
.1 1
.88 -1.08 0.9 1.09 -0.8
-0.9 1.0 -1.0 -1.0 0.9
0.38 0.6 1.2 -0.7 -1.18
-0.11 -0.9 -0.9 1.0 0.91
new vector for user #3

These vectors are close. In particular, the “shadow” of orange vector onto blue is pretty large. (Remember, the **dot product** is a measure of similarity!)



How to *find* embeddings?

Here are two key ways:

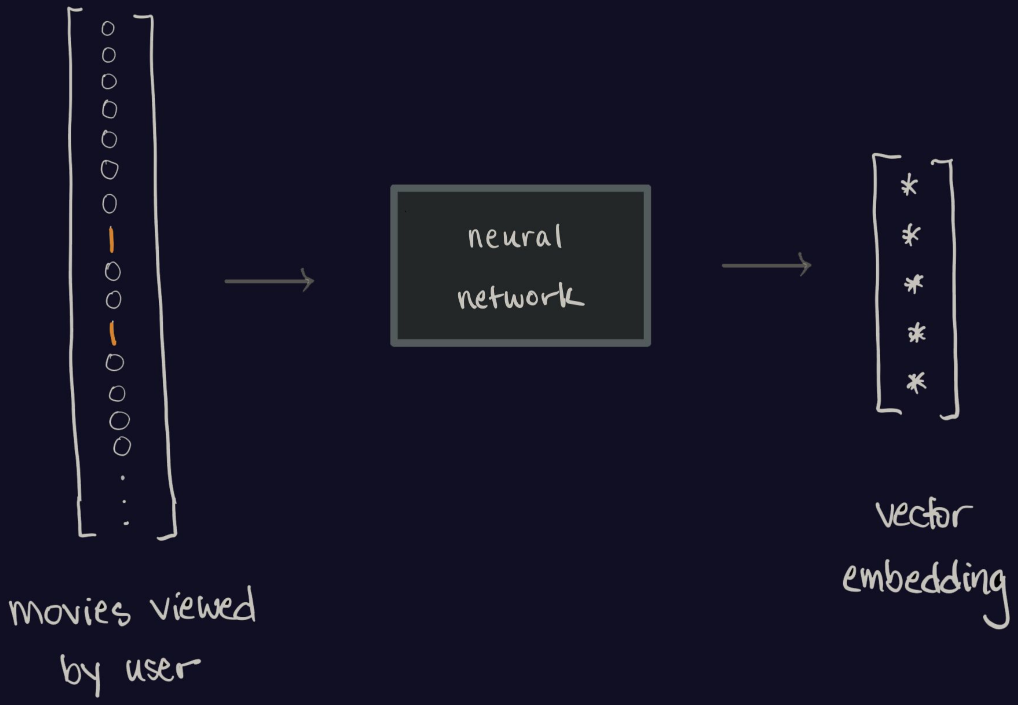
1. Matrix factorizations ✓
2. Neural networks

Neural Networks

CliffsNotes Version

Feed data vector into a neural network. The output is vector embedding.

Under the hood: matrix multiplication *plus more*.



In either case,
the goal is the same:

“Compress” high-dimensional data into a smaller-dimensional, more meaningful subspace.

This should be done in a way that doesn’t lose too much information.

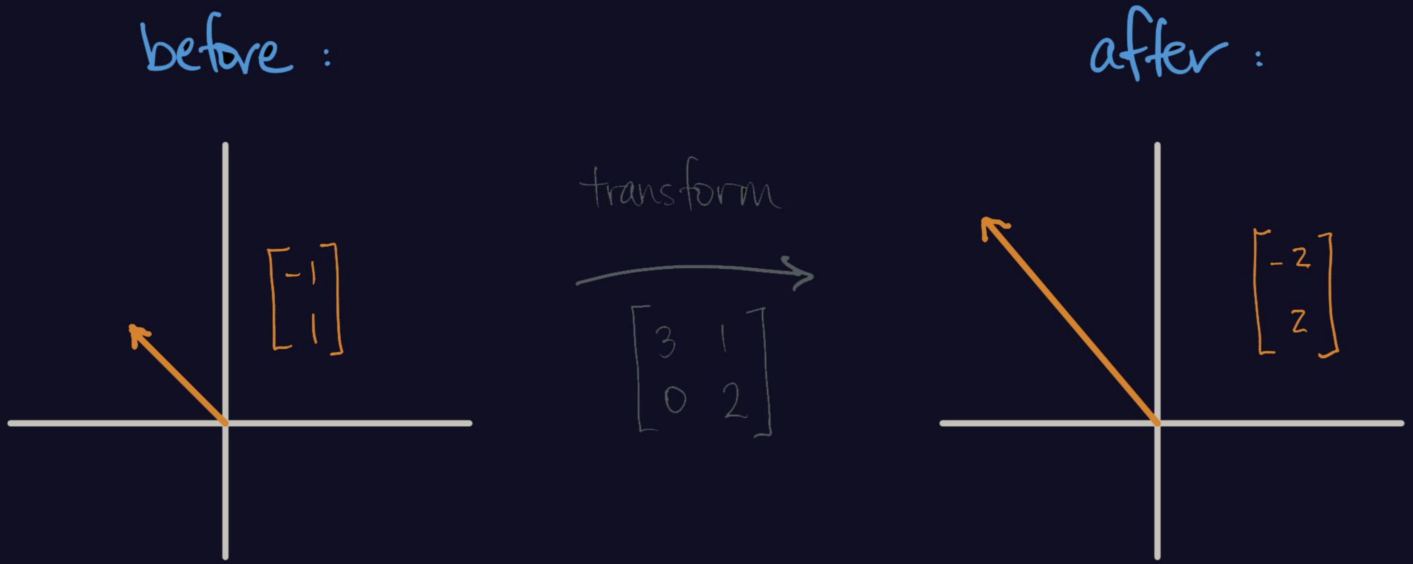
This leads us to **dimensionality reduction** techniques.

Dimensionality Reduction

via **eigenvectors** (a.k.a. “principal components”)

Idea: Find **eigenvectors**, a.k.a. "principal components."

Recall that a matrix represents a **transformation** between vector spaces. There are some transformations for which some vectors *never change direction*, but are only scaled.



These special vectors are called **eigenvectors**. The scaling factor is called an **eigenvalue**.

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$M \quad v \quad = \quad 2 \quad v$$

For visual intuition, see YouTube video [“Eigenvectors and Eigenvalues”] at 2:39 by Grant Sanderson (3Blue1Brown).

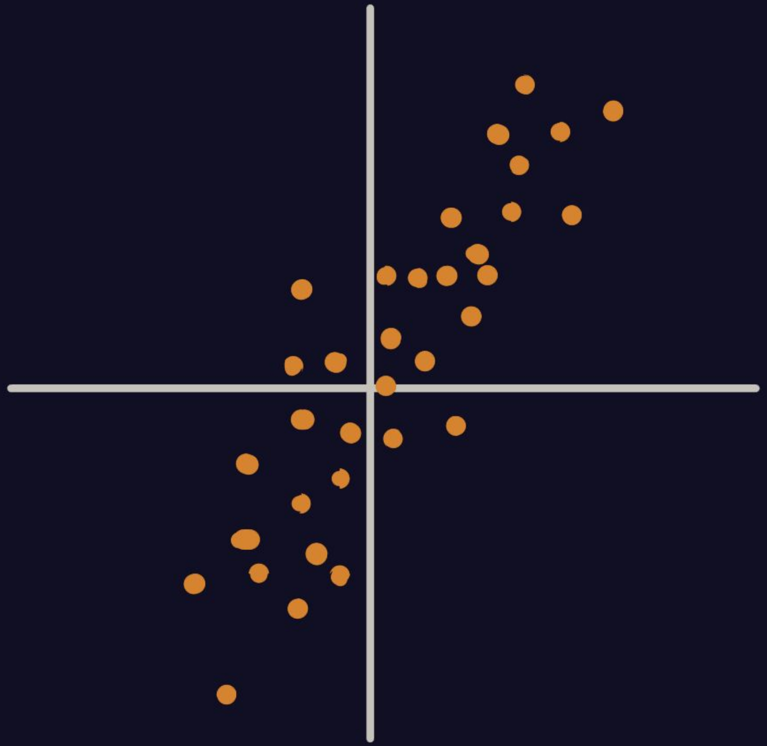
Eigenvectors play an important role in linear algebra.

In the context of data science and machine learning, they encode **valuable information**. Here’s one example.

Suppose we have n data points in m-dimensional space.

Oftentimes, these points will be **clustered** along a line or other low-dimensional subspace.

What is that line or subspace?

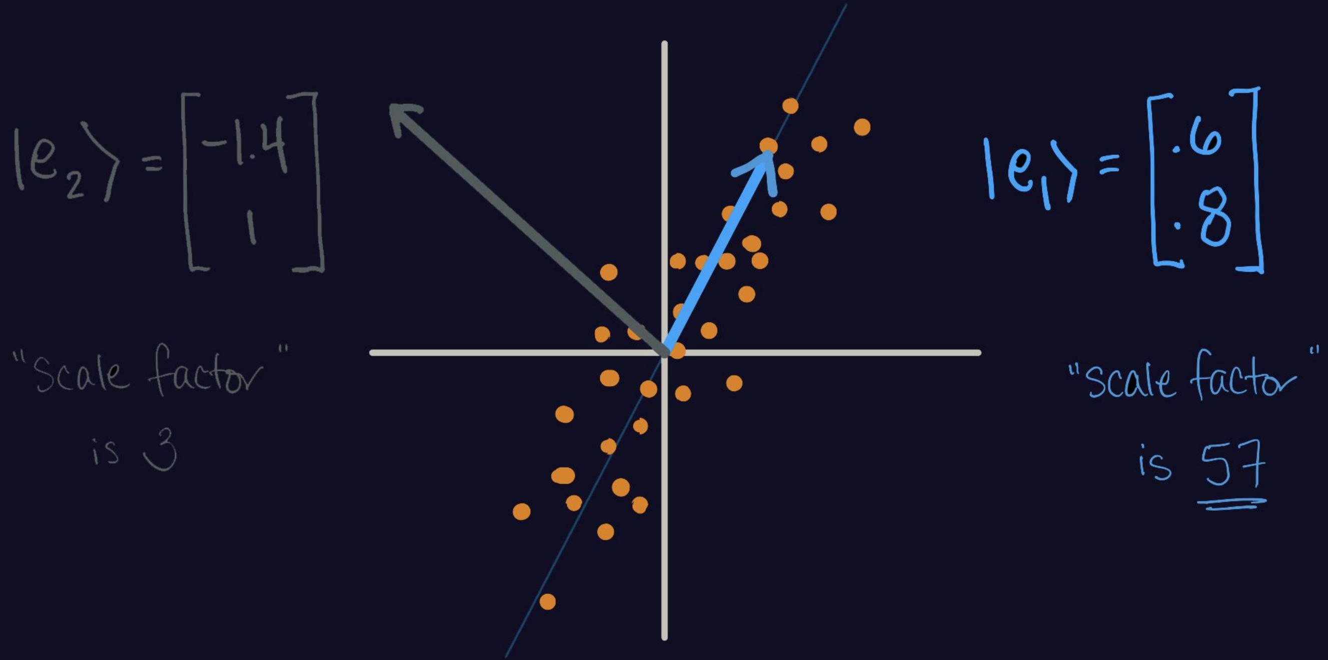


To find this line, first organize the data points into an $m \times n$ matrix A, and then compute the eigenvectors of $AA^T/n\text{-}1$.

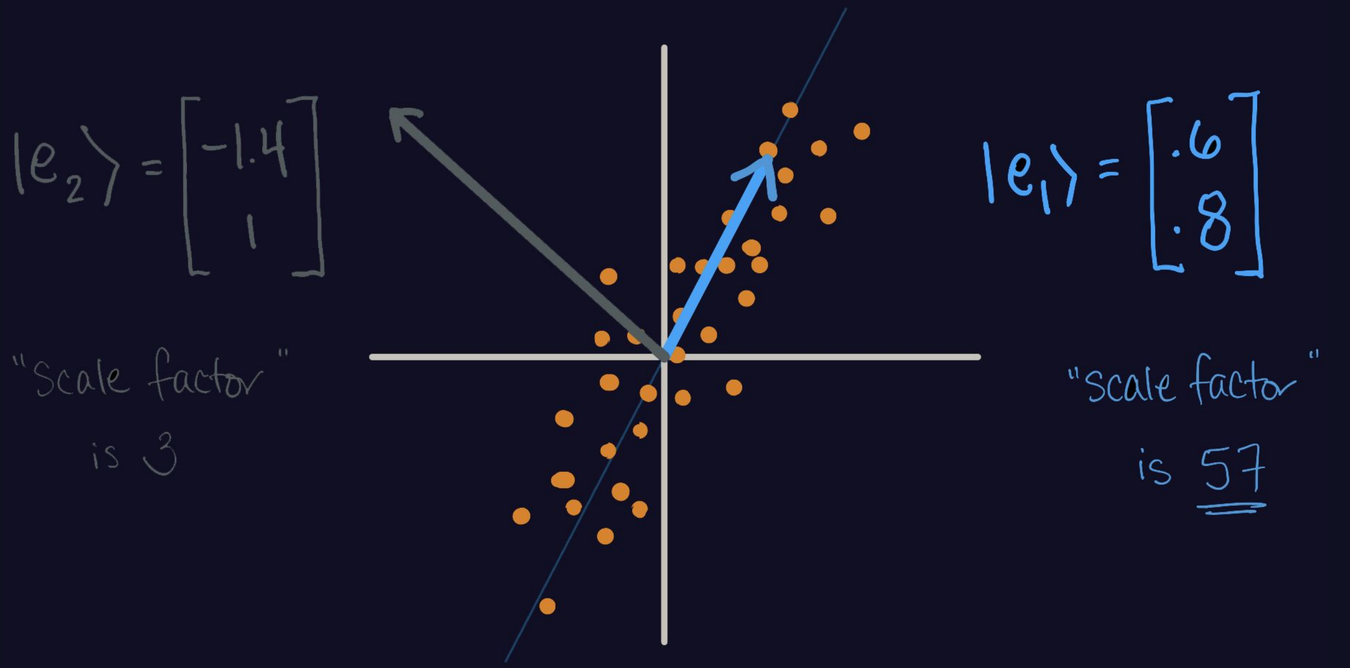
$$A = \begin{array}{cc} \begin{array}{cccccc} \text{Student 1} & \text{Student 2} & \text{Student 3} & \text{Student 4} & \text{Student 5} & \text{Student 6} \end{array} & \\ \begin{bmatrix} 3 & -4 & 7 & 1 & -4 & -3 \\ 7 & -6 & 8 & -1 & -1 & -7 \end{bmatrix} & \begin{array}{l} \text{math} \\ \text{history} \end{array} \end{array}$$

There's a little more to this. We first center the data points by subtracting the *mean* of each row. The matrix $AA^T/n\text{-}1$ is called a sample covariance matrix.

In this example, the matrix $AA^T/5$ has two eigenvectors, shown below. The eigenvector with the largest eigenvalue picks out the **principal direction** of the data.



This is the idea behind **principal component analysis.** For more on this example, see ch. 7 of Gilbert Strang's *Introduction to Linear Algebra* (5th ed.).



Here’s the takeaway.

Oftentimes, relevant data occupies only a small portion of the ambient large-dimensional vector space.

Working within that smaller space increases efficiency, saves resources, and reveals relevant structure in the data.

Linear algebra, and other dimensionality reduction techniques, can help locate that small space.

In summary...

Linear algebra is key for machine learning.

Data Representations

Use basic ideas of linear algebra to represent data in a way that computers can understand: **vectors.**

Vector Embeddings

Learn ways to choose these representations wisely via **matrix factorizations**.

Dimensionality Reduction

Deal with large-dimensional data using linear maps and their **eigenvectors**.

Thanks