Machine Learning Foundations
Linear Algebra II: Matrix Operations

Use Tensors in Python to
Solve Systems of Equations and
Identify Meaningful Patterns in Data

Jon Krohn, Ph.D.

jonkrohn.com/talks

github.com/jonkrohn/ML-foundations

Machine Learning Foundations
Linear Algebra II: Matrix Operations

Slides: jonkrohn.com/talks

Code: github.com/jonkrohn/ML-foundations

Stay in Touch:

jonkrohn.com to sign up for email newsletter

linkedin.com/in/jonkrohn

jonkrohn.com/youtube

twitter.com/JonKrohnLearns

The Pomodoro Technique

Rounds of:

- 25 minutes of work
- with 5 minute breaks

Questions best handled at breaks, so save questions until then.

When people ask questions that have already been answered, do me a favor and let them know, politely providing response if appropriate.

Except during breaks, I recommend attending to this lecture only as topics are not discrete: Later material builds on earlier material.

POLL

Where are you?

- The Americas
- Europe / Middle East / Africa
- Asia-Pacific
- Extra-Terrestrial Space

POLL

What are you?

- Developer / Engineer
- Scientist / Analyst / Statistician / Mathematician
- Combination of the Above
- Other

POLL

What is your level of familiarity with Linear Algebra?

- Little to no exposure
- Some understanding of the theory
- Deep understanding of the theory
- Deep understanding of the theory and experience applying linear algebra operations with code

POLL

What is your level of familiarity with Machine Learning?

- Little to no exposure, or exposure to theory only
- Experience applying machine learning with code
- Experience applying machine learning with code and some understanding of the underlying theory
- Experience applying machine learning with code and strong understanding of the underlying theory

ML Foundations Series

Linear Algebra II **builds upon** and is **foundational for**:

1. **Intro to Linear Algebra**
2. **Linear Algebra II: Matrix Operations**
3. Calculus I: Limits & Derivatives
4. Calculus II: Partial Derivatives & Integrals
5. Probability & Information Theory
6. Intro to Statistics
7. Algorithms & Data Structures
8. Optimization

Linear Algebra II: Matrix Operations

1. Review of Introductory Linear Algebra
2. Eigendecomposition
3. Matrix Operations for Machine Learning

Linear Algebra II: Matrix Operations

1. **Review of Introductory Linear Algebra**
2. Eigendecomposition
3. Matrix Operations for Machine Learning

Segment 1: Review of Matrix Properties

- Modern Linear Algebra Applications
- Tensors, Vectors, and Norms
- Matrix Multiplication
- Matrix Inversion
- Identity, Diagonal and Orthogonal Matrices

What Linear Algebra Is

"Solving for unknowns within system of linear equations"

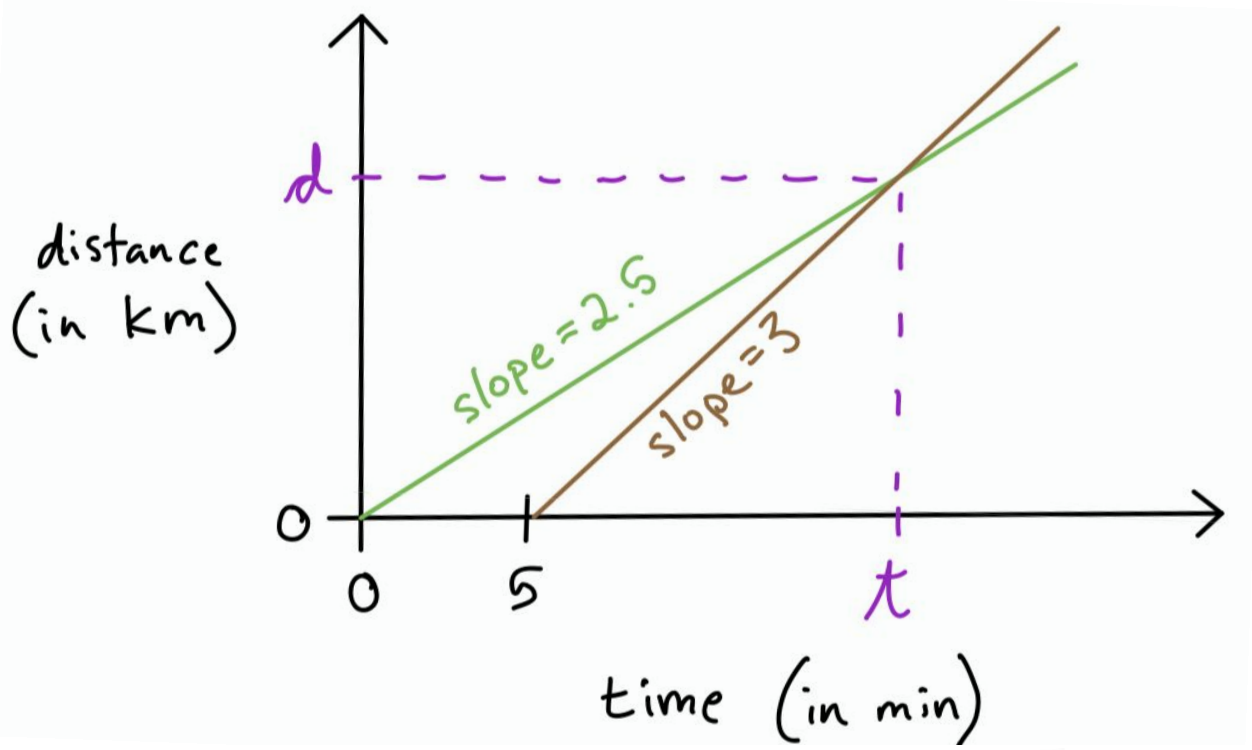
Consider the following example:

- Sheriff has 180 km/h car
- Bank robber has 150 km/h car and five-minute head start
- How long does it take the sheriff to catch the robber?
- What distance will they have traveled at that point?
- (For simplicity, let's ignore acceleration, traffic, etc.)

What Linear Algebra Is

Problem could be solved graphically with a plot:

(Note that: 150 km/h = 2.5 km/min 180 km/h = 3 km/min)



What Linear Algebra Is

Alternatively, problem can be solved *algebraically*:

Equation 1: $d = 2.5t$

Equation 2: $d = 3(t - 5)$

$2.5t = 3(t - 5)$

$2.5t = 3t - 15$

$2.5t - 3t = -15$

$-0.5t = -15$

$t = -15/-0.5 = 30$ min

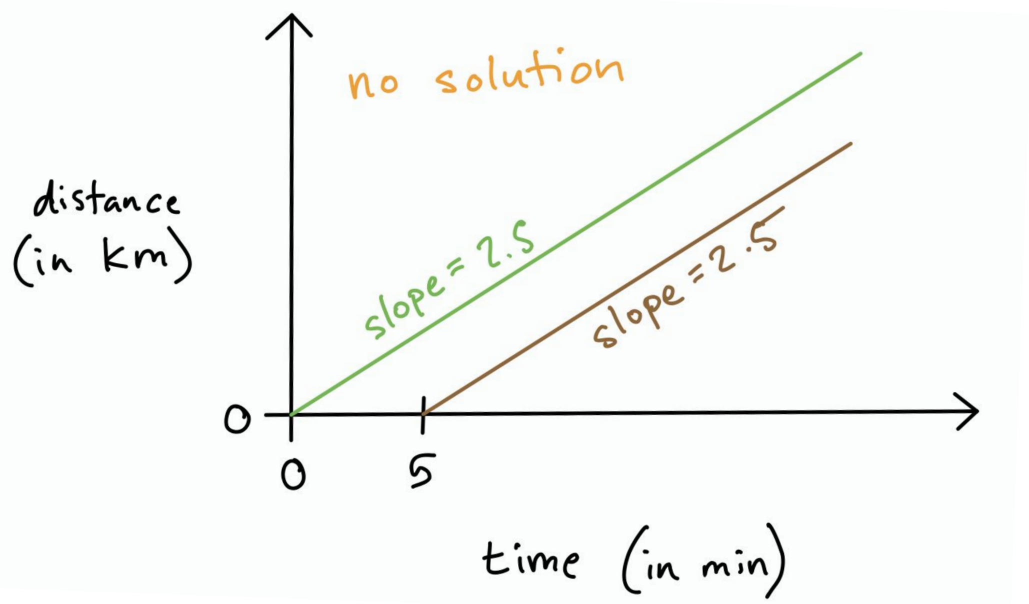
$d = 2.5t = 2.5(30) = 75$ km

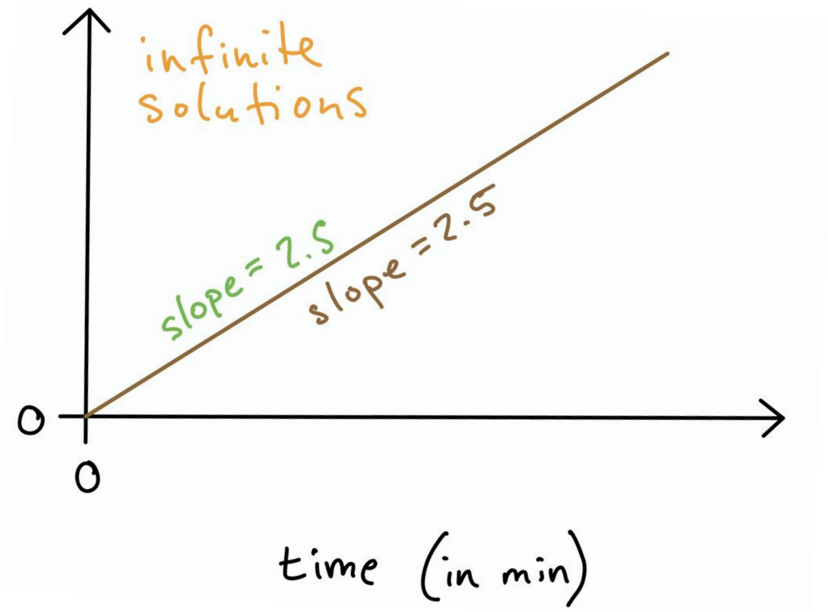
$d = 3(t - 5) = 3(30 - 5) = 3(25) = 75$ km

What Linear Algebra Is

No solution if sheriff's car is same speed as bank robber's.

Infinite solutions if same speed *and* same starting time.





These are the only three options in linear algebra: one, no, or infinite solutions.

It is impossible for lines to cross multiple times.

What Linear Algebra Is

In a given system of equations:

- Could be *many* equations
- Could be *many* unknowns in each equation

house price

distance to school

there could be m features (many!)

$$y = a + b x_1 + c x_2 + \ldots + m x_m$$

"y-intercept"

number of bedrooms

$$y = a + b x_1 + c x_2 + \dots + m x_m$$

$$\left[\begin{array}{c|ccccc}
y_1 & a + b x_{1,1} & + c x_{1,2} & + \dots & + m x_{1,m} \\
y_2 & a + b x_{2,1} & + c x_{2,2} & + \dots & + m x_{2,m} \\
\vdots & \vdots \quad \vdots & \vdots & & \vdots \\
y_n & a + b x_{n,1} & + c x_{n,2} & + \dots & + m x_{n,m}
\end{array}\right]$$

For any house i in the dataset, y_i = price and $x_{i,1}$ to $x_{i,m}$ are its features.

We solve for parameters a, b, c to m

Modern Linear Algebra Applications

- Solving for unknowns in ML algos, including deep learning
- Reducing dimensionality (e.g., **principal component analysis**)
- Ranking results (e.g., with **eigenvector**, including in Google PageRank algorithm; *see Saaty and Hu, 1998*)
- Recommenders (e.g., **singular value decomposition**, **SVD**)
- Natural language processing (e.g., **SVD**, matrix factorization)
    - Topic modeling
    - Semantic analysis

Tensors

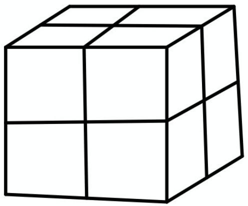
"ML generalization of vectors and matrices to any number of dimensions"

scalar x

vector $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$

matrix $\begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix}$

3-tensor



Dimensions	Mathematical Name	Description
0	scalar	magnitude only
1	vector	array
2	matrix	flat table, e.g., square
3	3-tensor	3D table, e.g., cube
n	*n*-tensor	higher dimensional

Vector Transposition

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

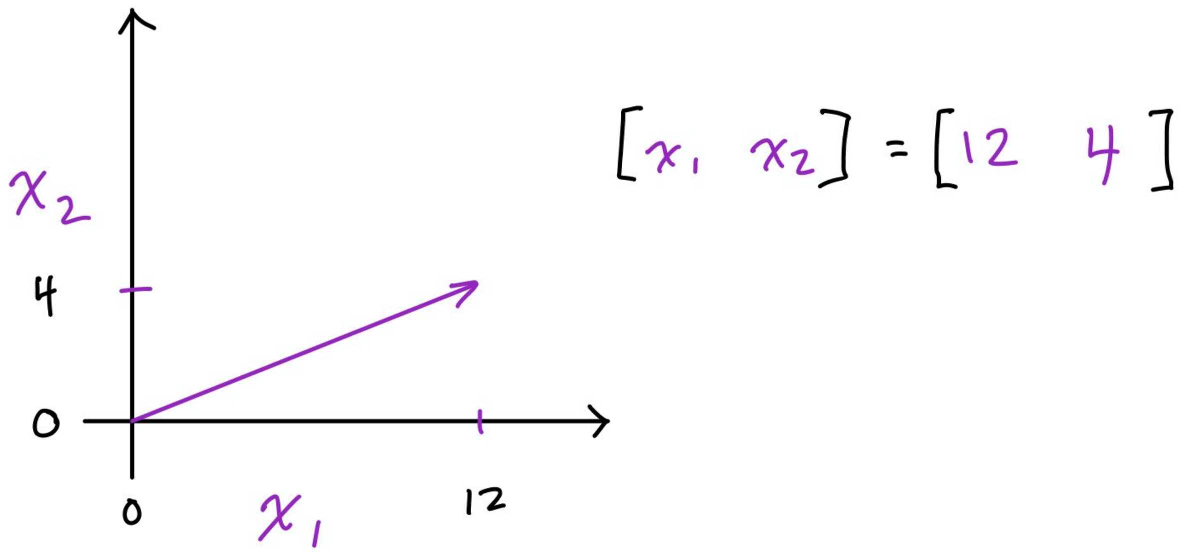
row vector — column vector

shape is $(1, 3)$ — $(3, 1)$

Hands-on code demo: `2-linear-algebra-ii.ipynb`

Norms

Vectors represent a magnitude and direction from origin:



Norms are functions that quantify vector magnitude:

- In ML, L^2 and L^1 norms are common, e.g., to avoid overfitting

L^2 Norm

- Described by:

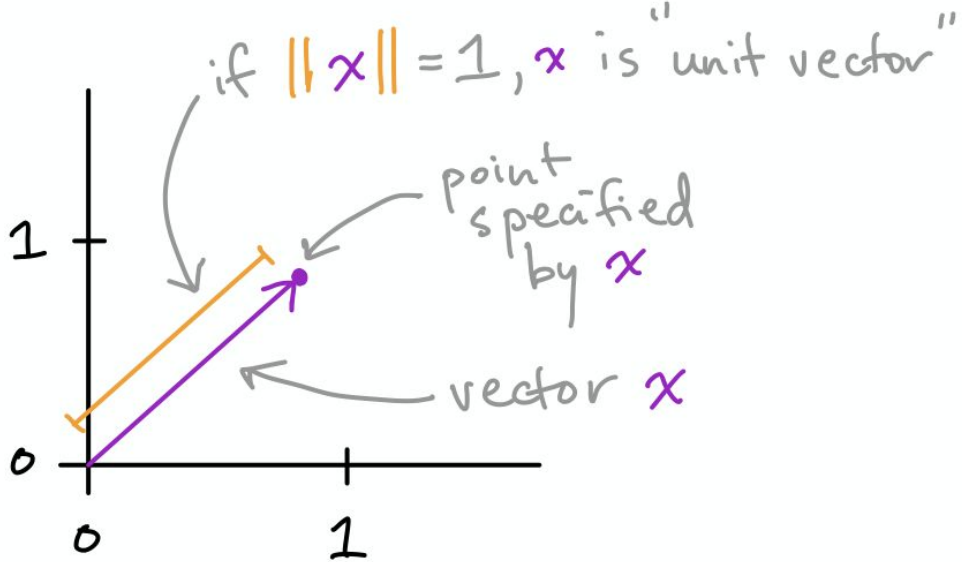
$$||\boldsymbol{x}||_2 = \sqrt{\sum_i x_i^2}$$

- Measures simple (Euclidean) distance from origin
- Most common norm in machine learning
    - Instead of $||\boldsymbol{x}||_2$, it can be denoted as $||\boldsymbol{x}||$

Hands-on code demo

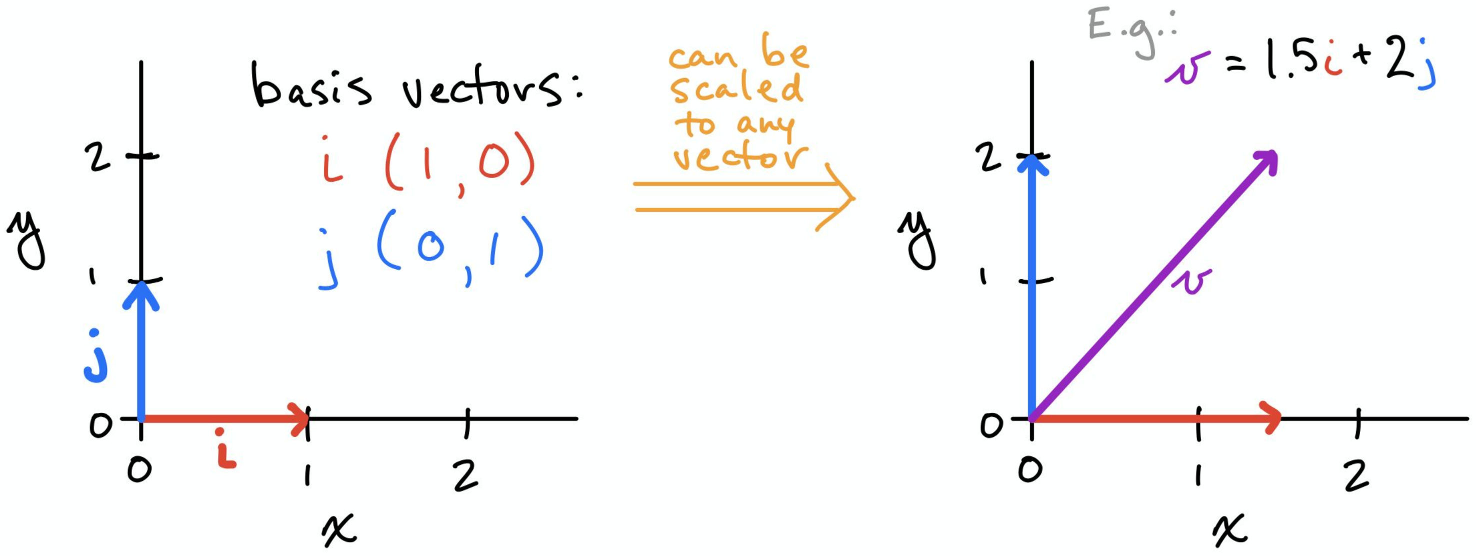
Unit Vectors

- Special case of vector where its length is equal to one
- Technically, $\boldsymbol{x}$ is a unit vector with "unit norm", i.e.: $||\boldsymbol{x}|| = 1$



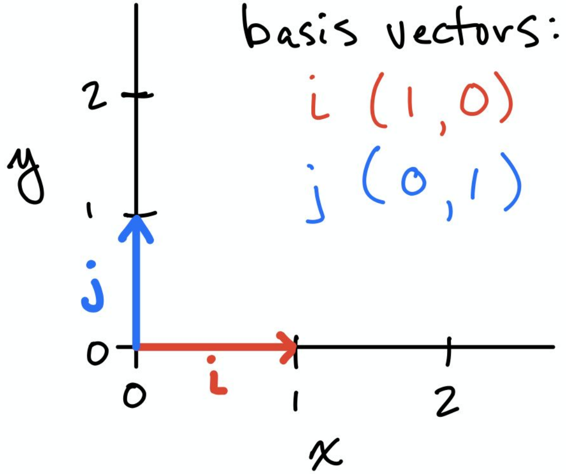
Basis Vectors

- Can be scaled to represent *any* vector in a given vector space
- Typically use unit vectors along axes of vector space (shown)



Orthogonal Vectors

- $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal vectors if $\boldsymbol{x}^\mathrm{T}\boldsymbol{y} = 0$
- Are at 90° angle to each other (assuming non-zero norms)
- n-dimensional space has max n mutually orthogonal vectors (again, assuming non-zero norms)
- **Orthonormal** vectors are orthogonal *and* all have unit norm
    - Basis vectors are an example



Matrix Transposition

Flip of axes over **main diagonal** such that:

$$(\boldsymbol{X}^{\mathrm{T}})_{i,j} = \boldsymbol{X}_{j,i}$$

$$\begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \\ X_{3,1} & X_{3,2} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} X_{1,1} & X_{2,1} & X_{3,1} \\ X_{1,2} & X_{2,2} & X_{3,2} \end{bmatrix}$$

Hands-on code demo

Symmetric Matrices

Special matrix case with following properties:

- Square
- $\boldsymbol{X}^T = \boldsymbol{X}$

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 7 & 8 \\ 2 & 8 & 9 \end{bmatrix}$$

Matrix Multiplication

$$\begin{array}{c c} m & \begin{bmatrix} C \end{bmatrix} \\ & p \end{array} = \begin{array}{c c} m & \begin{bmatrix} A \end{bmatrix} \\ & n \end{array} \begin{array}{c c} n & \begin{bmatrix} B \end{bmatrix} \\ & p \end{array}$$

$$C_{i,k} = \sum_{j} A_{i,j} B_{j,k}$$

Matrix Multiplication (with a Vector)

$$\begin{bmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 4 \cdot 2 \\ 5 \cdot 1 + 6 \cdot 2 \\ 7 \cdot 1 + 8 \cdot 2 \end{bmatrix} = \begin{bmatrix} 3 + 8 \\ 5 + 12 \\ 7 + 16 \end{bmatrix} = \begin{bmatrix} 11 \\ 17 \\ 23 \end{bmatrix}$$

Hands-on code demo

(Matrix-by-)Matrix Multiplication

$$\begin{bmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 9 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 3\cdot1+4\cdot2 & 3\cdot9+4\cdot0 \\ 5\cdot1+6\cdot2 & 5\cdot9+6\cdot0 \\ 7\cdot1+8\cdot2 & 7\cdot9+8\cdot0 \end{bmatrix} = \begin{bmatrix} 11 & 27 \\ 17 & 45 \\ 23 & 63 \end{bmatrix}$$

Hands-on code demo

$$y = a + b x_1 + c x_2 + \dots + m x_m$$

$$\left[\begin{array}{c|c} y_1 & a + b x_{1,1} + c x_{1,2} + \dots + m x_{1,m} \\ y_2 & a + b x_{2,1} + c x_{2,2} + \dots + m x_{2,m} \\ \vdots & \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\ y_n & a + b x_{n,1} + c x_{n,2} + \dots + m x_{n,m} \end{array}\right]$$

Strictly speaking, *x* extends rightward to *m*-1 not *m* because of the presence of *a* on the far left.

For any house i in the dataset, y_i = price and $x_{i,1}$ to $x_{i,m}$ are its features.

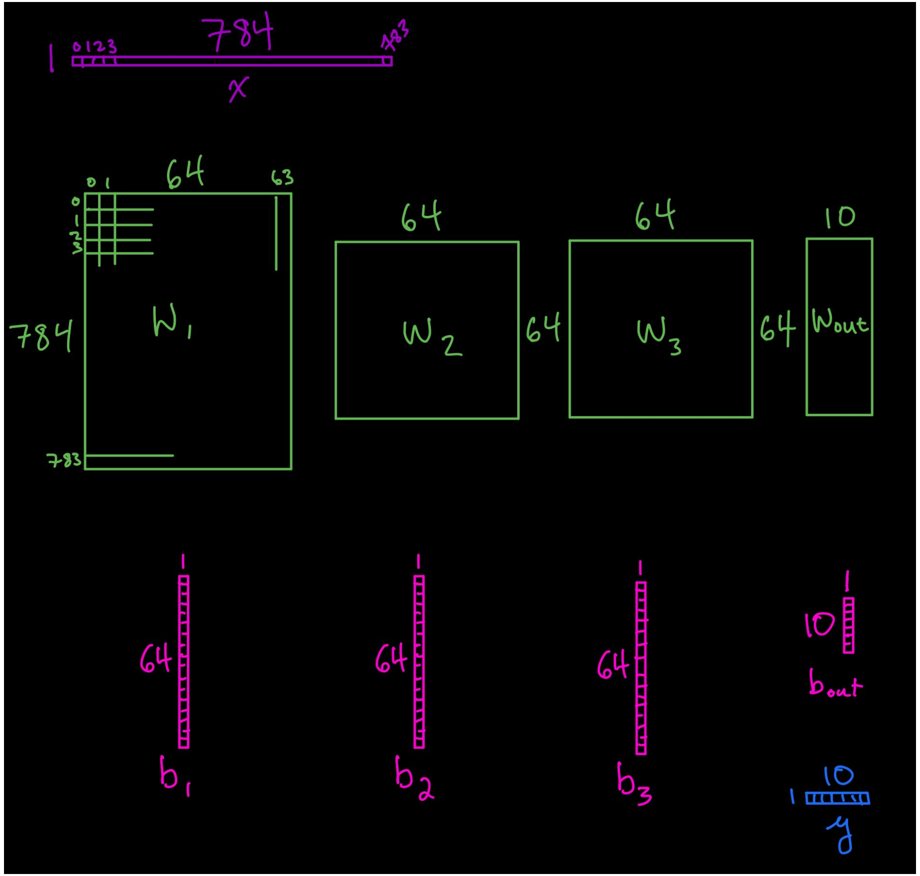
We solve for parameters a, b, c to m

Matrix Multiplication (in Regression)

$$\begin{array}{c} n \\ \text{cases} \\ \text{tall} \end{array} \left\{ \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right. = \underbrace{\begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix}}_{m \text{ features wide}} \begin{bmatrix} a \\ b \\ c \\ \vdots \\ m \end{bmatrix}$$

In other words, the matrix represents an *m*-dimensional space.

Matrix Multiplication (in Deep Learning)



See:

- artificial-neurons.ipynb
- jonkrohn.com/deepTF1
- jonkrohn.com/convTF1
- jonkrohn.com/convTF2
- jonkrohn.com/deepPT

Identity Matrices

Symmetric matrix where:

- Every element along main diagonal is 1
- All other elements are 0
- Notation: $\boldsymbol{I}_n$ where n = height (or width)
- n-length vector unchanged if multiplied by $\boldsymbol{I}_n$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

I_4

Matrix Inversion

- Clever, convenient approach for solving linear equations
- An alternative to manually solving with elimination or addition

- **Matrix inverse** of $\boldsymbol{X}$ is denoted as $\boldsymbol{X}^{-1}$
  - Satisfies: $\boldsymbol{X}^{-1}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{X}^{-1} = \boldsymbol{I}_n$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

I_4

Matrix Inversion

$$\begin{matrix} n \\ \text{cases} \\ \text{tall} \end{matrix} \left\{ \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right. = \underbrace{\begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix}}_{m \text{ features wide}} \begin{bmatrix} a \\ b \\ c \\ \vdots \\ m \end{bmatrix}$$

The regression formula can be represented as:

$y = Xw$ (w is the vector of weights b through m)

Matrix Inversion

In the equation $y = Xw$:

- We know the outcomes y, which could be house prices
- We know the features X, which are predictors like bedroom count
- Vector w contains the unknowns, the model's learnable parameters

Assuming X^{-1} exists, matrix inversion can solve for w:

$$Xw = y$$

$$X^{-1}Xw = X^{-1}y$$

$$I_n w = X^{-1}y$$

$$w = X^{-1}y$$

Matrix Inversion

$$\begin{cases} 4b + 2c = 4 \\ -5b - 3c = -7 \end{cases}$$

$$X = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ -5 & -3 \end{bmatrix} \qquad y = \begin{bmatrix} 4 \\ -7 \end{bmatrix}$$

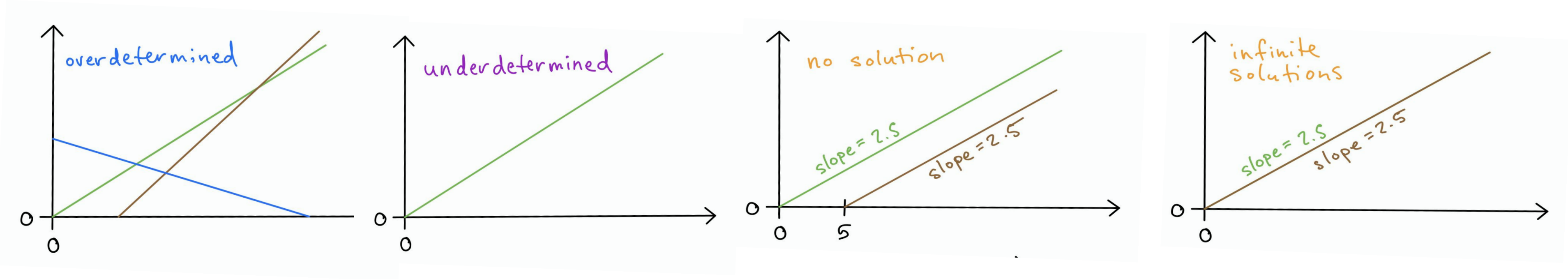
$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix} = X^{-1} y$$

Hands-on code demo

Matrix Inversion

Nifty trick, but can only be calculated if:

- Matrix is square: $n_{row} = n_{col}$ (i.e., "vector span" = "matrix range")
  - Avoids **overdetermination**: n_{row} (# of equations) > n_{col} (# of dims)
  - Avoids **underdetermination**: $n_{row} < n_{col}$
- Matrix isn't "singular", i.e.: all columns are linearly independent
  - **E.g., if a column is [1, 2], another can't be [2, 4] or also be [1, 2]**



Note that solving for unknowns may still be possible by other means if matrix can't be inverted...

Diagonal Matrices

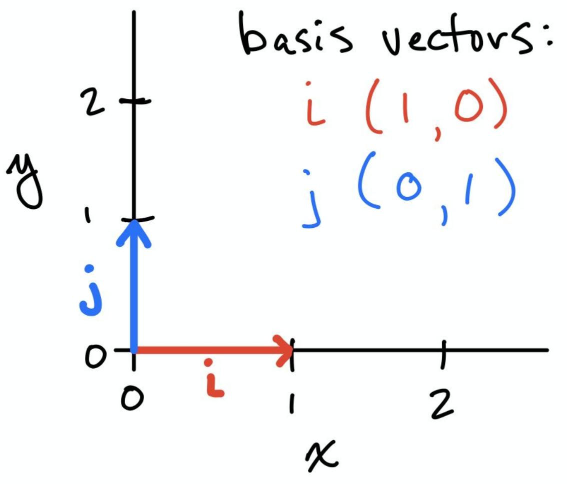
- Nonzero elements along main diagonal; zeros everywhere else
- Identity matrix is an example
- If square, denoted as diag($\boldsymbol{x}$) where $\boldsymbol{x}$ is vector of main-diagonal elements
- Computationally efficient:
  - Multiplication: $\text{diag}(\boldsymbol{x})\boldsymbol{y} = \boldsymbol{x} \odot \boldsymbol{y}$
  - Inversion: $\text{diag}(\boldsymbol{x})^{-1} = \text{diag}[1/\boldsymbol{x}_1, \ldots, 1/\boldsymbol{x}_n]^T$
    - Can't divide by zero so $\boldsymbol{x}$ can't include zero
- Can be non-square and computation still efficient:
  - If $h > w$, simply add zeros to product
  - If $w > h$, remove elements from product

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

I_4

Orthogonal Matrices

Recall orthonormal vectors from earlier:



In orthogonal matrices, orthonormal vectors:

- Make up all rows
- Make up all columns

This means: $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{I}$

Which also means: $\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{A}^{-1}\boldsymbol{I} = \boldsymbol{A}^{-1}$

Calculating $\boldsymbol{A}^{\mathrm{T}}$ is cheap, therefore so is calculating $\boldsymbol{A}^{-1}$

Linear Algebra II: Matrix Operations

1. Review of Introductory Linear Algebra
2. **Eigendecomposition**
3. Matrix Operations for Machine Learning

Segment 2: Eigendecomposition

- Applying Matrices
- Affine Transformations
- Eigenvectors
- Eigenvalues
- Matrix Determinants
- Matrix Decomposition
- Applications of Eigendecomposition

Matrix-Application Exercises

Using pen(cil) and paper:

1. Apply the identity matrix $\boldsymbol{I}_3$ to the vector $\boldsymbol{u}$.
2. Apply the matrix $\boldsymbol{B}$ to the vector $\boldsymbol{u}$.
3. Concatenate vector $\boldsymbol{u}$ with vector $\boldsymbol{u}_2$ to form a matrix $\boldsymbol{U}$, then apply the matrix $\boldsymbol{B}$ to the matrix $\boldsymbol{U}$.

$$u = \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix} \qquad u_2 = \begin{bmatrix} 0 \\ -4 \\ 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 & 0 & -1 \\ -2 & 3 & 1 \\ 0 & 4 & -1 \end{bmatrix}$$

Solutions

$$I_3 u = \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix}$$

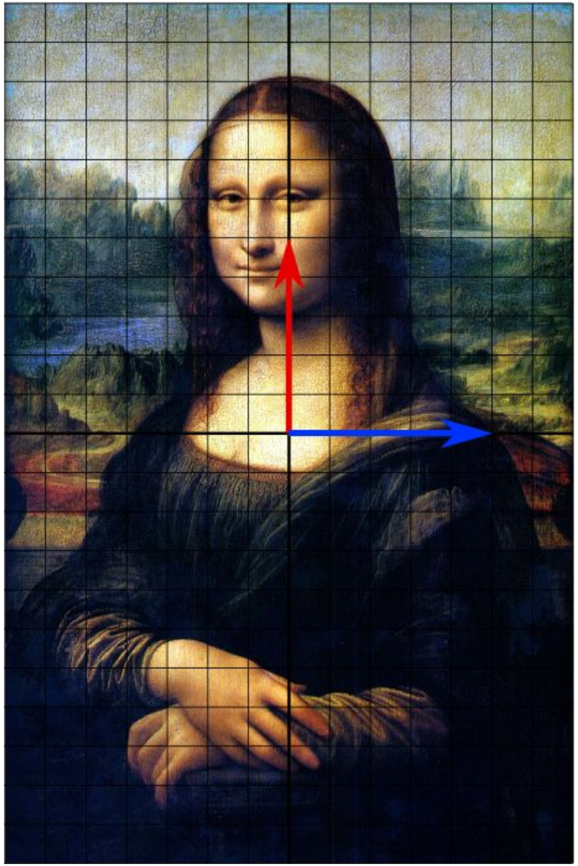
$$B u = \begin{bmatrix} 4 & +0 & +3 \\ -4 & +15 & -3 \\ 0 & 20 & 3 \end{bmatrix} = \begin{bmatrix} 7 \\ 8 \\ 23 \end{bmatrix}$$

$$B u_2 = \begin{bmatrix} 0 & +0 & -6 \\ 0 & -12 & 6 \\ 0 & -16 & -6 \end{bmatrix} = \begin{bmatrix} -6 \\ -6 \\ -22 \end{bmatrix}$$

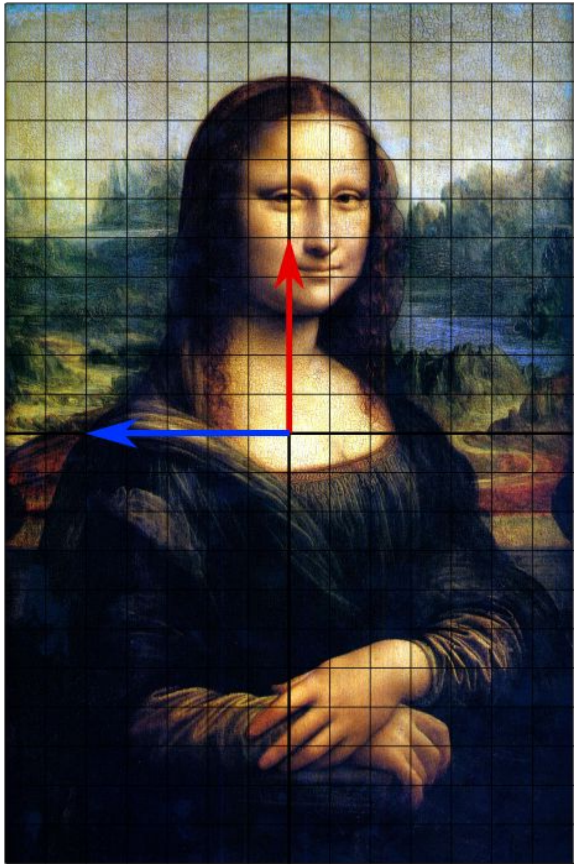
$$BU = \begin{bmatrix} 7 & -6 \\ 8 & -6 \\ 23 & -22 \end{bmatrix}$$

Hands-on code demo

Eigenvectors



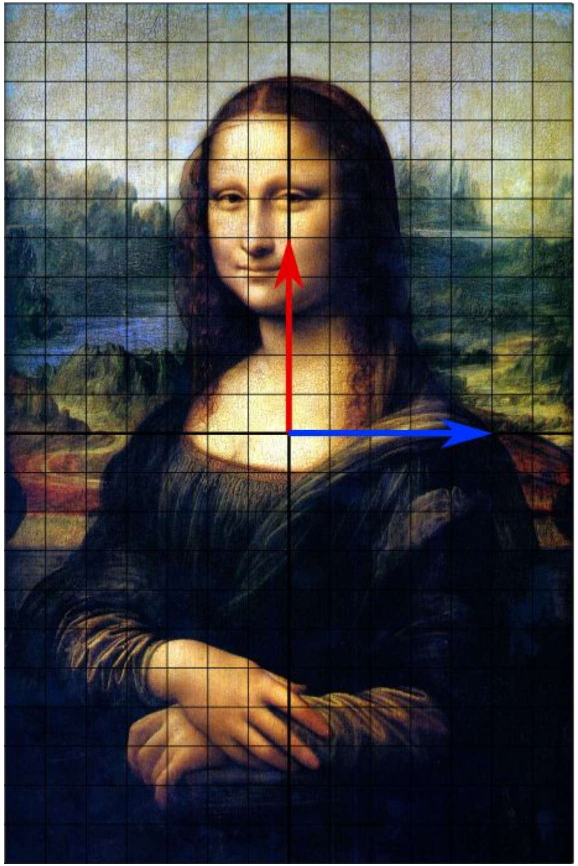
Flipping matrix applied



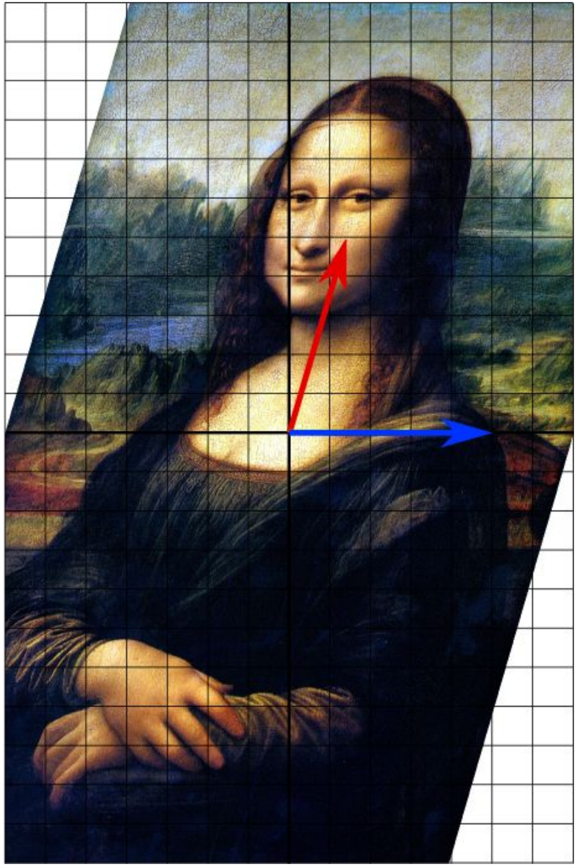
Red vector and **blue vector** are **eigenvectors** for the flipping matrix.

commons.wikimedia.org/w/index.php?curid=12768508

Eigenvectors



Shearing matrix applied

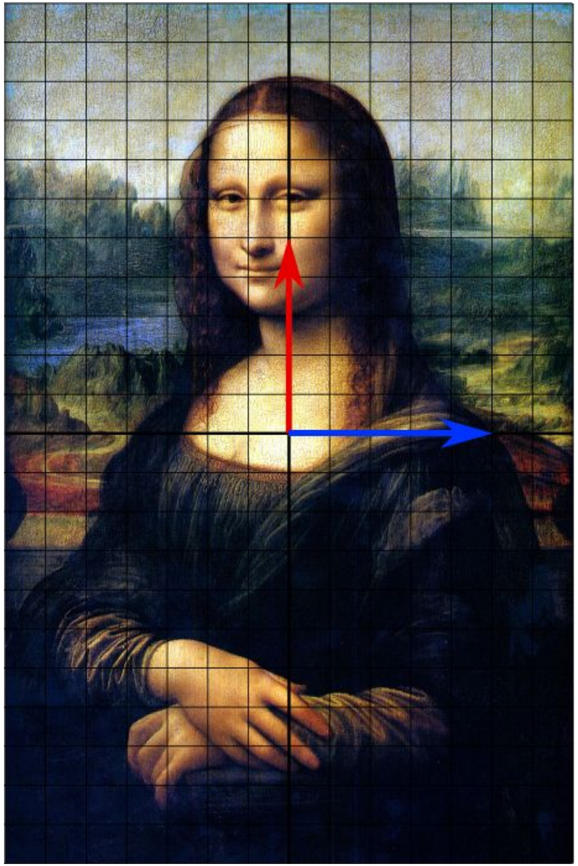


Red vector knocked off span

Blue vector isn't -- it maintains its direction -- so it is an **eigenvector** for the shearing matrix.

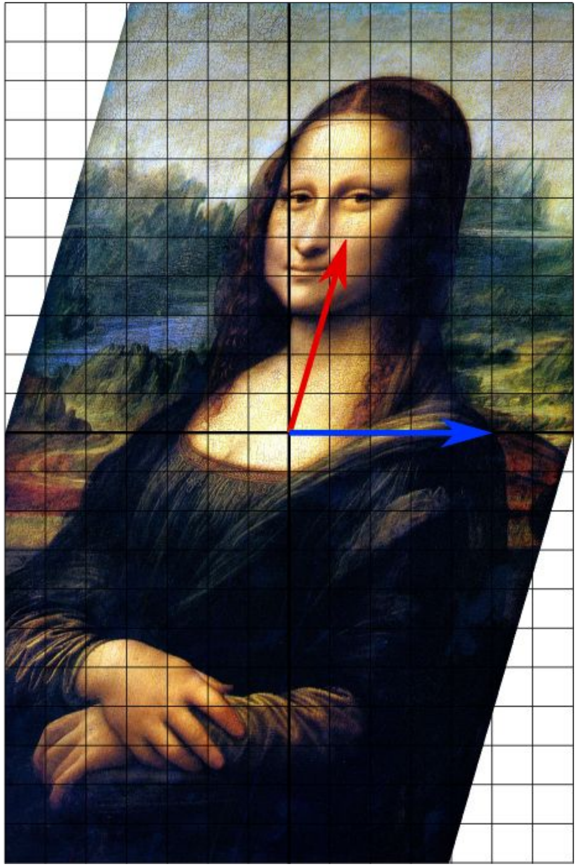
commons.wikimedia.org/w/index.php?curid=12768508

Eigen*values*



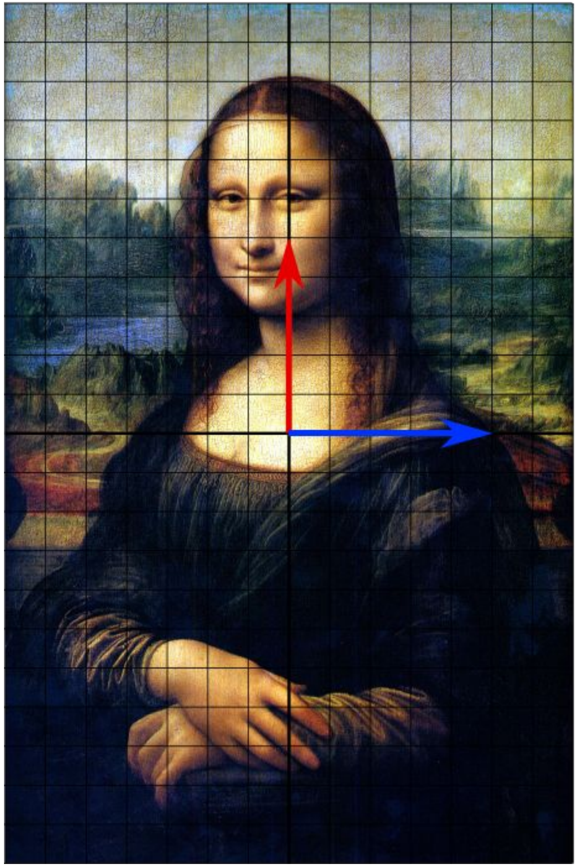
In this case, **eigenvector** retains exact length, so it's **eigenvalue** = 1.

If **eigenvector** were to double in length, its **eigenvalue** = 2; if it halves, **eigenvalue** = 0.5.



commons.wikimedia.org/w/index.php?curid=12768508

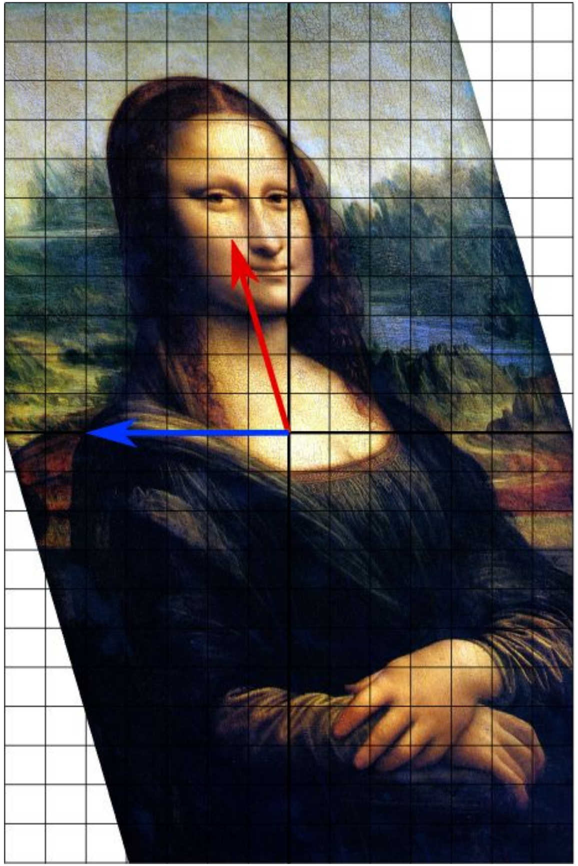
Eigenvalues



Eigenvalues can also have a negative sign, e.g., a new **shearing-and-flipping matrix** has the same **eigenvector** as shearing-only matrix but its **eigenvalue** = -1.

If **eigenvector** were to double in length while exactly reversing direction, **eigenvalue** would be -2.

Hands-on code demo

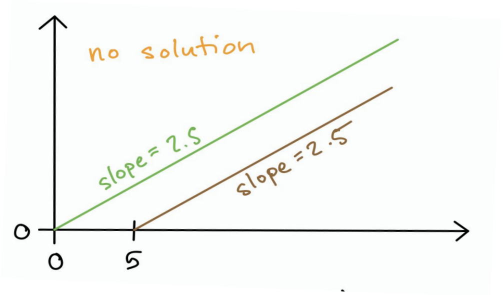


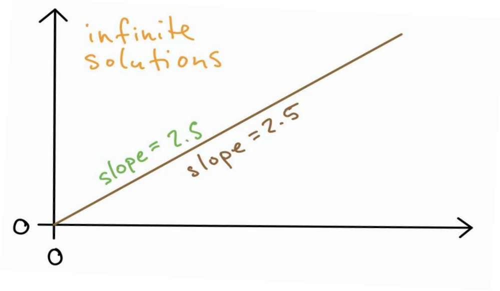
commons.wikimedia.org/w/index.php?curid=12768508

Matrix Determinants

- Map <u>square</u> matrix to scalar
- Enable us to determine whether matrix can be inverted

- For matrix $\boldsymbol{X}$, denoted as det($\boldsymbol{X}$)
- If det($\boldsymbol{X}$) = 0:
  - Matrix $\boldsymbol{X}^{-1}$ can't be computed because: $\boldsymbol{X}^{-1}$ has 1/det($\boldsymbol{X}$) = 1/**0**
  - Matrix $\boldsymbol{X}$ is singular: It contains linearly-dependent columns
- det($\boldsymbol{X}$) easiest to calculate for 2x2 matrix...





Determinant of 2x2 Matrix

$$X = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad X = \begin{bmatrix} 4 & 2 \\ -5 & -3 \end{bmatrix}$$

$$|X| = ad - bc$$

$$|X| = 4(-3) - 2(-5)$$
$$= -12 + 10$$
$$= -2$$

Hands-on code demo

Determinant of 2x2 Matrix

$$X = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$N = \begin{bmatrix} -4 & 1 \\ -8 & 2 \end{bmatrix}$$

$$|X| = ad - bc$$

$$|N| = -4 \cdot 2 - 1(-8)$$

$$= -8 + 8$$

$$= 0$$

Hands-on code demo

Generalizing Determinants: Recursion

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} \\ x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} \end{bmatrix}$$

5 rows means 4 rounds of recursion

$$|X| = x_{1,1}\det(X_{1,1}) - x_{1,2}\det(X_{1,2}) + x_{1,3}\det(X_{1,3}) - x_{1,4}\det(X_{1,4}) + x_{1,5}\det(X_{1,5})$$

alternating +/-

Generalizing Determinants: Recursion

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & -1 & 3 \\ 0 & 5 & 1 \end{bmatrix}$$

3 rows means
2 rounds of recursion

$$|X| = x_{1,1}\det(X_{1,1}) - x_{1,2}\det(X_{1,2}) + x_{1,3}\det(X_{1,3})$$

$$= 1\begin{vmatrix} -1 & 3 \\ 5 & 1 \end{vmatrix} - 2\begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix} + 4\begin{vmatrix} 2 & -1 \\ 0 & 5 \end{vmatrix}$$

$$= 1(-1\cdot 1 - 3\cdot 5) - 2(2\cdot 1 - 3\cdot 0) + 4(2\cdot 5 - (-1)(0))$$

$$= 1(-1 - 15) - 2(2 - 0) + 4(10 - 0)$$

$$= -16 - 4 + 40$$

$$= 20$$

Hands-on code demo

Exercises

Using pencil and paper, calculate the determinant of the matrices below. Indicate which have an inverse and which don't:

1. $$\begin{bmatrix} 25 & 2 \\ 3 & 4 \end{bmatrix}$$

2. $$\begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix}$$

3. $$\begin{bmatrix} 2 & 1 & -3 \\ 4 & -5 & 2 \\ 0 & -1 & 3 \end{bmatrix}$$

Solutions

1. 94; has inverse
2. 4; has inverse
3. -26; has inverse

Determinants & Eigenvalues

$\det(\boldsymbol{X})$ = product of all eigenvalues of $\boldsymbol{X}$

Hands-on code demo

$|\det(\boldsymbol{X})|$ quantifies volume change as a result of applying $\boldsymbol{X}$:

- If $\det(\boldsymbol{X}) = 0$, then $\boldsymbol{X}$ collapses space completely in at least one dimension, thereby eliminating all volume
- If $0 < |\det(\boldsymbol{X})| < 1$, then $\boldsymbol{X}$ contracts volume to some extent
- If $|\det(\boldsymbol{X})| = 1$, then $\boldsymbol{X}$ preserves volume exactly
- If $|\det(\boldsymbol{X})| > 1$, then $\boldsymbol{X}$ expands volume

Hands-on code demo

Eigendecomposition

$$A = V \Lambda V^{-1}$$

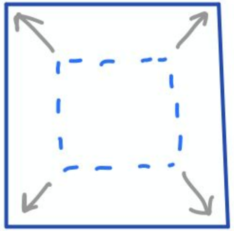
The decomposition of a matrix into eigenvectors and eigenvalues reveals characteristics of the matrix, e.g.:

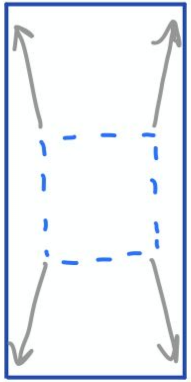
- Matrix is singular if and only if any of its eigenvalues are zero
- Under specific conditions (see §2.7 of Goodfellow et al., 2016), can optimize quadratic expressions:
    - Maximum of $f(\boldsymbol{x})$ = largest eigenvalue
    - Minimum of $f(\boldsymbol{x})$ = smallest eigenvalue

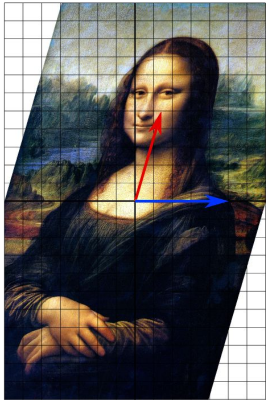
Hands-on code demo

Eigendecomposition Examples

2D geometric transformation	**Scaling (equal)**	**Scaling (unequal)**	**Horizontal shear**	**Vertical shear**
2x2 Matrix	$[[k, 0], [0, k]]$	$[[k_1, 0], [0, k_2]]$	$[[1, k], [0, 1]]$	$[[1, 0], [k, 1]]$
Eigenvalues	$\lambda_1 = \lambda_2 = k$	$\lambda_1 = k_1$ and $\lambda_2 = k_2$	$\lambda_1 = \lambda_2 = 1$	$\lambda_1 = \lambda_2 = 2$
Example eigenvectors	non-zero	$\boldsymbol{v}_1 = [1,0]$ and $\boldsymbol{v}_2 = [0,1]$	$\boldsymbol{v}_1 = [1,0]$	$\boldsymbol{v}_1 = [0,1]$







Eigendecomposition Applications

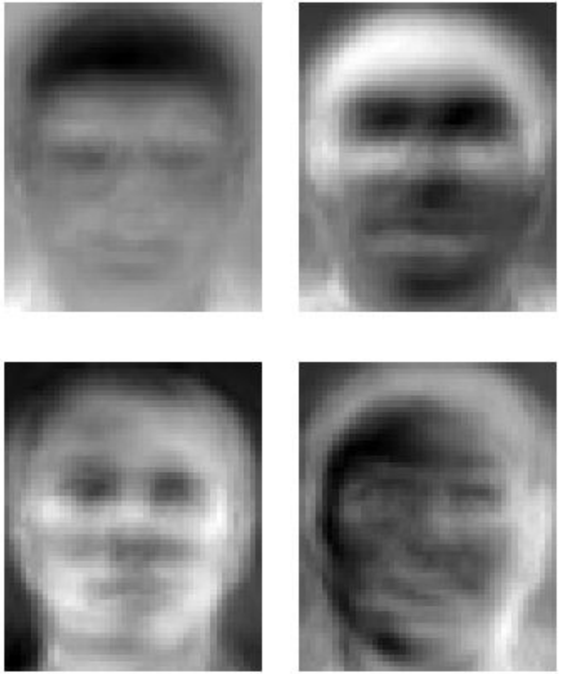
Matrix is of type:	**If all its eigenvalues are:**
Positive definite	>0
Positive semidefinite	≥ 0
Negative definite	<0
Negative semidefinite	≤ 0

Applying a matrix of a particular type to some vector $\boldsymbol{x}$ can have a characteristic impact (again, see §2.7 of Goodfellow et al., 2016):

- E.g., semidefinite matrices collapse tensors along 1+ dimensions

Eigendecomposition Applications

- Eigenvectors, as underlying characteristics of a dataset, can be recombined into any members of the dataset, e.g.:
  - Eigenfaces **(shown)**
  - Eigenvoices
  - Eigenfrequencies (of vibrations)
- Quantum mechanics:
  - Molecular orbitals
  - Schrödinger wave equation
- Reproduction number R_0 in epidemiology
- Calculating determinants *(already covered)*
- SVD & Moore-Penrose pseudoinverse (*next*)
- Principal component analysis *(coming up)*



commons.wikimedia.org/w/index.php?curid=442472

Linear Algebra II: Matrix Operations

1. Review of Introductory Linear Algebra
2. Eigendecomposition
3. **Matrix Operations for Machine Learning**

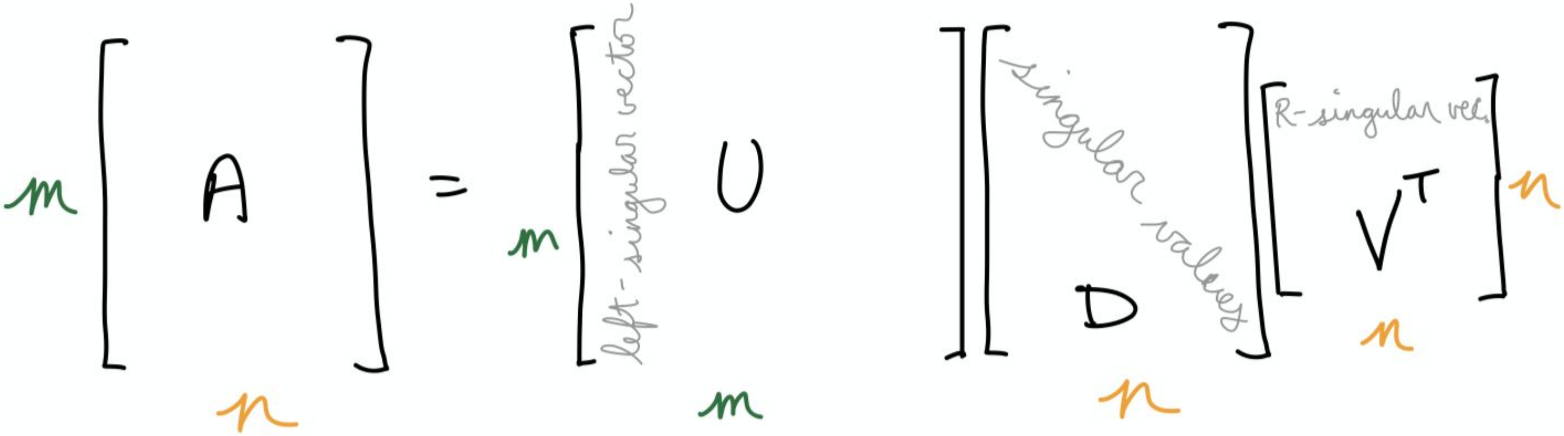
Segment 3: Matrix Operations for ML

- Singular Value Decomposition (SVD)
- The Moore-Penrose Pseudoinverse
- The Trace Operator
- Principal Component Analysis (PCA)
- Resources for Further Study of Linear Algebra

Singular Value Decomposition

- Unlike eigendecomposition, which is applicable to square matrices only, SVD is applicable to *any* real-valued matrix
- Decomposes matrix into:
  - **Singular vectors** (analogous to eigenvectors)
  - **Singular values** (analogous to eigenvalues)
- For some matrix $\boldsymbol{A}$, its SVD is $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}$

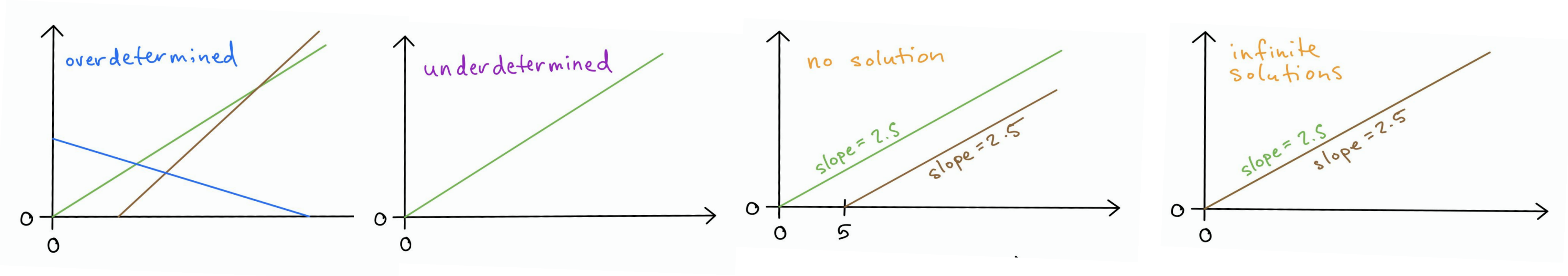
Hands-on code demo



Matrix Inversion *Revisited*

Nifty trick, but can only be calculated if:

- Matrix is square: $n_{row} = n_{col}$ (i.e., "vector span" = "matrix range")
  - Avoids **overdetermination**: n_{row} (# of equations) > n_{col} (# of dims)
  - Avoids **underdetermination**: $n_{row} < n_{col}$
- Matrix isn't "singular", i.e.: all columns are linearly independent
  - **E.g., if a column is [1, 2], another can't be [2, 4] or also be [1, 2]**



Note that solving for unknowns may still be possible by other means if matrix can't be inverted.

Moore-Penrose Pseudoinverse

- Such an "other mean" when matrix itself can't be inverted
- Instead, invert the singular values of the matrix

For some matrix $\boldsymbol{A}$, its pseudoinverse $\boldsymbol{A}^+$ can be calculated by:

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{D}^+\boldsymbol{U}^\mathrm{T}$$

Where:

- $\boldsymbol{U}$, $\boldsymbol{D}$, and $\boldsymbol{V}$ are SVD of $\boldsymbol{A}$
- $\boldsymbol{D}^+ = (\boldsymbol{D}$ with reciprocal of all-non zero elements$)^\mathrm{T}$

Hands-on code demo

Moore-Penrose Pseudoinverse

$\boldsymbol{A}^{+}$ is mega useful because non-square matrices are common in ML:

$$y = a + b x_1 + c x_2 + \ldots + m x_m$$

$$\begin{bmatrix} y_1 & \Big| & a + b x_{1,1} + c x_{1,2} + \ldots + m x_{1,m} \\ y_2 & \Big| & a + b x_{2,1} + c x_{2,2} + \ldots + m x_{2,m} \\ \vdots & \Big| & \vdots \\ y_n & \Big| & a + b x_{n,1} + c x_{n,2} + \ldots + m x_{n,m} \end{bmatrix}$$

For any house i in the dataset, y_i = price and $x_{i,1}$ to $x_{i,m}$ are its features.

We solve for parameters a, b, c to m

Matrix Inversion *Revisited*

$$\begin{matrix} n \\ \text{cases} \\ \text{tall} \end{matrix} \left\{ \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right. = \underbrace{\begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix}}_{m \text{ features wide}} \begin{bmatrix} a \\ b \\ c \\ \vdots \\ m \end{bmatrix}$$

The regression formula can be represented as:

$\boldsymbol{y} = \boldsymbol{Xw}$ (***w*** is the vector of weights *a* through *m*)

Matrix Inversion *Revisited*

In the equation $y = Xw$:

- We know the outcomes y, which could be house prices
- We know the features X, which are predictors like bedroom count
- Vector w contains the unknowns, the model's learnable parameters

Assuming X^{-1} exists, matrix inversion can solve for w:

$$Xw = y$$

$$X^{-1}Xw = X^{-1}y$$

$$I_n w = X^{-1}y$$

$$w = X^{-1}y$$

Matrix Inversion *Revisited*

$$\begin{cases} 4b + 2c = 4 \\ -5b - 3c = -7 \end{cases}$$

$$X = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ -5 & -3 \end{bmatrix} \qquad y = \begin{bmatrix} 4 \\ -7 \end{bmatrix}$$

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix} = X^{-1} y$$

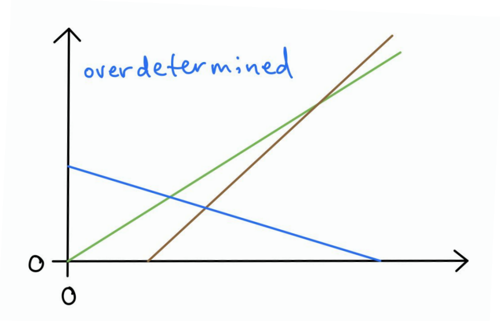
Moore-Penrose Pseudoinverse

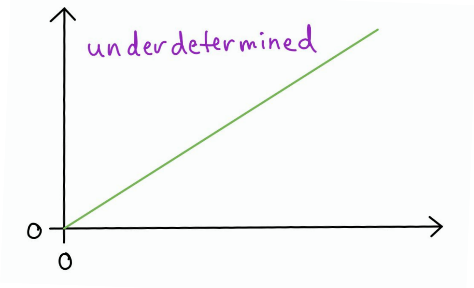
- Would be unusual to have exactly as many cases (n) as features (m)
- With pseudoinverse $\boldsymbol{X}^+$, we can now estimate model weights $\boldsymbol{w}$ if $n \neq m$:

$$\boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{y}$$

- If $\boldsymbol{X}$ is overdetermined ($n > m$), $\boldsymbol{X}^+$ provides $\boldsymbol{X}\boldsymbol{y}$ as close to $\boldsymbol{w}$ as possible (in terms of Euclidean distance, specifically $||\boldsymbol{X}\boldsymbol{y} - \boldsymbol{w}||_2$)
- If $\boldsymbol{X}$ is underdetermined ($n < m$), $\boldsymbol{X}^+$ provides the $\boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{y}$ solution that has the smallest Euclidean norm $||\boldsymbol{x}||_2$ from all the possible solutions

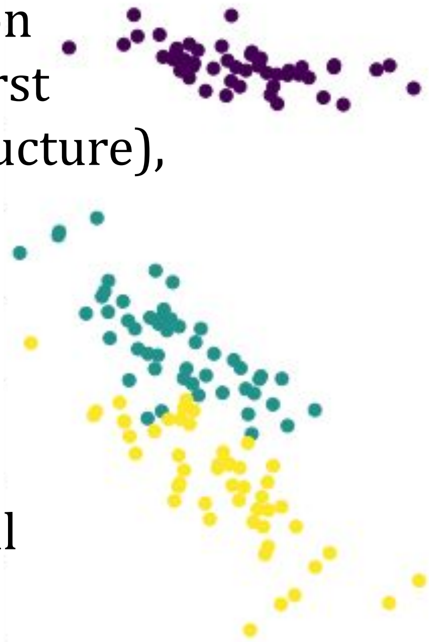
Hands-on code demo





Principal Component Analysis

- Simple machine learning algorithm
- **Unsupervised**: enables identification of structure in unlabeled data
- Like eigendecomposition and SVD, enables lossy compression
  - To minimize both loss of precision and data footprint, first **principal component** contains most variance (data structure), second PC contains next most, and so on
- Involves many linear algebra concepts already covered, e.g.:
  - Norms
  - Orthogonal and identity matrices
  - Trace operator
  - See Goodfellow et al. (2016) §2.12 for five pages of detail



Hands-on code demo

Resources for Further Study

- **Basic algebra**:
  - Khan Academy
  - 3Blue1Brown on YouTube
- **Linear algebra**:
  - 3Blue1Brown again
  - Ch. 2 of Goodfellow et al. (2016) *Deep Learning* (free)
  - Ch. 2 of Deisenroth et al. (2020) *Mathematics for ML*
  - Sheldon Axler's (2015) *Linear Algebra Done Right*
  - Gilbert Strang: Linear Algebra course via MIT Open Courseware
    - *Linear Algebra and Its Applications* book
- **Next steps in the *ML Foundations* series**:
  - Calculus I: Limits & Derivatives
  - Calculus II: Partial Derivatives & Integrals

Stay in Touch

`jonkrohn.com` to sign up for email newsletter

`linkedin.com/in/jonkrohn`

YouTube `jonkrohn.com/youtube`

`twitter.com/JonKrohnLearns`

POLL *with Multiple Answers Possible*

What follow-up topics interest you most?

- More Linear Algebra
- Calculus
- Probability / Statistics
- Computer Science (e.g., algorithms, data structures)
- Machine Learning Basics
- Advanced Machine Learning, incl. Deep Learning
- Something Else

NEBULA

PLACEHOLDER
FOR:

5-Minute Timer

PLACEHOLDER
FOR:

10-Minute Timer

PLACEHOLDER
FOR:

15-Minute Timer