رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Math Basics

Session 2: (Linear Algebra Fundamentals)

Task1: Research-Based Activity

Baraa Abu Sallout • 6:48 PM (Edited 6:54 PM)

10 points

Exploring Linear Algebra and its Applications in Machine Learning

Researching Basic Concepts and Their Use in AI

- Matrices and their use in data representation.
- Vectors and how they're used in image or text processing.
- Eigenvalues & Eigenvectors and their role in techniques like PCA.
- Linear Transformations and their impact on data.
- Each students should prepare a short word report explaining the concept they researched **with simple examples and AI applications**
- **Don't use GPTs, Search for references on Google and must attach the references at the end of the word file.**

Bonus:

- Researching Basic Concepts and Their Use in AI on Statistics Fundamentals (Descriptive & Inferential Statistics, Probability)

Class comments

Add class comment...

ليه ال Linear Alg, مهم في ML؟
مع ذكر على الاقل 3 أمثلة لكل منهما

Data -

Transformations in data

PCA

models -

neural network -

Embedding in LLM -

.Evel -

11:10 to 11:40 AM


رواد مصر الرقمية

Agenda:

1	Vectors and Matrices
2	Matrix Operations
3	Linear Transformations and Eigenvalues

Vectors and Matrices

Vector

A vector is an ordered list of numbers (scalars) that can represent a point in space. Vectors have both **magnitude** (length) and **direction**. In the context of AI, vectors are used to **represent data points**, **features**, or **weights**.

Vectors are typically represented as **columns** (column vectors) or **rows** (row vectors). For example, a 3-dimensional vector can be written as:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

Vectors

Examples

- **Position Vector**: In a 2D space, a vector $\mathbf{v} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ can represent a point 3 units to the right and 4 units up from the origin.

- **Feature Vector**: In machine learning, a vector can represent features of a data point, such as $\mathbf{x} = \begin{bmatrix} \text{age} \\ \text{income} \end{bmatrix} = \begin{bmatrix} 30 \\ 50000 \end{bmatrix}$.

Operations on Vectors

Addition

Vectors of the same dimension can be added by adding corresponding elements:

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \end{bmatrix}$$

Operations on Vectors

Subtraction

Similar to addition, but subtracting corresponding elements.

Scalar Multiplication

Multiplying a vector by a scalar (number) scales each element of the vector:

$$c \cdot \mathbf{v} = c \cdot \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} c \cdot v_1 \\ c \cdot v_2 \end{bmatrix}$$

Matrices

A matrix is a rectangular array of numbers arranged in rows and columns. Matrices can represent multiple vectors, data tables, or linear transformations.

A matrix A with m rows and n columns is denoted as $A_{m \times n}$. For example, a 2x3 matrix looks like this:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Matrices

Examples

- **Data Matrix:** In a dataset, each row could represent a different data point (case), and each column could represent a different feature (variable).

- **Transformation Matrix:** Matrices can be used to perform transformations like rotations, scaling, or translations in space

Why Vectors and Matrices are Important in AI and Machine Learning ?

- **Data Representation**: Vectors and matrices are used to represent datasets, where each row is a data point and each column is a feature.
- **Linear Transformations**: Matrices can represent transformations applied to data, such as scaling, rotating, and translating data points in space.
- **Matrix Operations:** Essential for performing calculations in machine learning algorithms, such as linear regression, neural networks, and principal component analysis (PCA).

Summary

- Vectors represent points or directions in space and can be used to model features of data points.
- Matrices represent more complex data structures and transformations and are fundamental tools for organizing and manipulating data in AI.
- Understanding vector and matrix operations is crucial for implementing and optimizing machine learning algorithms.

Matrix Operations

Matrix Operations

This segment covers essential matrix operations that are fundamental for understanding how data is manipulated and transformed in AI and machine learning. Matrix operations enable the efficient computation of complex transformations, solving systems of linear equations, and performing various data manipulations.

Matrix Multiplication

Matrix multiplication is a way of combining two matrices to produce a new matrix. The number of columns in the first matrix must match the number of rows in the second matrix for multiplication to be possible.

Notation:

If A is an m×n matrix and B is an n×p matrix, the product AB is an m×p matrix.

Formula:

Each element of the resulting matrix C=AB is calculated as the dot product of a row in A and a column in B:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matrix Multiplication

Example

- Let A be a 2×3 matrix and B be a 3×2 matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

- The resulting matrix $C = AB$ is:

$$C = \begin{bmatrix} (1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11) & (1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12) \\ (4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11) & (4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12) \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

Square Matrix

A matrix with the same number of rows and columns.

```
import numpy as np

square_matrix = np.array([[1, 2, 3],
                          [4, 5, 6],
                          [7, 8, 9]])
print("Square Matrix:\n", square_matrix)
```

```
Square Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

This is a 3x3 matrix where the number of rows equals the number of columns.

Practical Uses

- **Covariance Matrix** → always square.
- **Adjacency Matrix** (Social Networks / Graphs).
- **Image Filters** (3×3, 5×5 kernels in Computer Vision).

Diagonal Matrix

A matrix in which all off-diagonal elements are zero

```
[40] diagonal_matrix = np.diag([1, 2, 3])
     print("Diagonal Matrix:\n", diagonal_matrix)
```

```
Diagonal Matrix:
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
```

The np.diag() function creates a diagonal matrix where only the elements along the main diagonal are non-zero.

Practical Uses

- **Covariance matrices** sometimes simplified to diagonal (no correlation between features).
- **Feature scaling** and **whitening** often use diagonal matrices.
- **Image processing filters** can be diagonal for certain effects.
- Simplifies many linear algebra operations.

Identity Matrix

A square matrix with ones on the main diagonal and zeros elsewhere.

```
identity_matrix = np.eye(3)
print("Identity Matrix:\n", identity_matrix)
```

```
Identity Matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

- A special type of diagonal matrix.
- Diagonal elements = 1.
- All off-diagonal elements = 0.
- The np.eye(3) function creates a 3x3 identity matrix. Identity matrices are used as the multiplicative identity in matrix algebra.

Practical Uses

- **Linear Algebra:** neutral element in multiplication.
- **Machine Learning:** used in regularization (adding λI to covariance matrix in Ridge Regression).
- **Computer Vision:** identity filter keeps the image unchanged.
- **Optimization:** often appears in matrix factorizations.

Transpose of a Matrix

Flipping a matrix over its diagonal, switching the row and column indices.

```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])
transpose_matrix = np.transpose(matrix)
print("Original Matrix:\n", matrix)
print("Transpose of Matrix:\n", transpose_matrix)
```

```
Original Matrix:
 [[1 2 3]
 [4 5 6]]
Transpose of Matrix:
 [[1 4]
 [2 5]
 [3 6]]
```

- The np.transpose() function is used to find the transpose of a matrix, which switches the rows and columns.
- Rows become columns, columns become rows.

Practical Uses

- **Dot Product: can be written as** $x^T y$
- **Covariance Matrix:** built using transposed data vectors.
- **Machine Learning:** in linear regression $X^T X$
- **Deep Learning:** weight matrices often transposed in backpropagation.

Multiple Linear Regression

Estimating Parameters Using Least Squares

- **Normal Equation:**

$$\beta = (X^T X)^{-1} X^T Y$$

  - X: Input matrix (including bias term)
  - Y: Output vector
  - β: Coefficient vector

Trace of a Matrix

The sum of all the elements on the main diagonal of a square matrix.

```
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
trace = np.trace(matrix)
print("Trace of Matrix:", trace)
```

```
Trace of Matrix: 15
```

The np.trace() function calculates the trace of a matrix, which is often used in machine learning algorithms and various mathematical computations.

Practical Uses

- **Machine Learning:** cost functions sometimes include trace terms (e.g., matrix norms).
- **Linear Algebra: trace =** sum of eigenvalues of a matrix.
- **PCA / Dimensionality Reduction:** trace of covariance matrix = total variance.
- **Optimization:** used in matrix calculus for gradients.

Determinant of a Matrix

A scalar value that is a function of a square matrix, often used in solving linear equations, calculating matrix inverses, and more.

```
matrix = np.array([[4, 6],
                   [3, 8]])
determinant = np.linalg.det(matrix)
print("Determinant of Matrix:", determinant)
```

```
Determinant of Matrix: 14.000000000000004
```

The np.linalg.det() function computes the determinant, which provides important information about the matrix properties, such as invertibility.

Practical Uses

- **Unique scalar value calculated from a square matrix.**
- **Tells us whether the matrix is invertible or not.**
- **In Data Science → used in solving linear equations.**

Determinant of a Matrix

The determinant is a scalar value that can be computed from the elements of a square matrix. It provides important properties of the matrix, such as whether it is invertible (non-singular).

Notation:

For a square matrix A, the determinant is denoted as det(A) or $|A|$.

Formula for a 2x2 Matrix

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Determinant of a Matrix

Example

For a matrix $A = \begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix}$, the determinant is:

$$\det(A) = (3 \cdot 6) - (8 \cdot 4) = 18 - 32 = -14$$

Inverse of a Matrix

A matrix that, when multiplied with the original matrix, yields the identity matrix.

```
matrix = np.array([[1, 2],
                   [3, 4]])
inverse_matrix = np.linalg.inv(matrix)
print("Inverse of Matrix:\n", inverse_matrix)
```

```
Inverse of Matrix:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

The np.linalg.inv() function finds the inverse of a matrix, which is useful for solving systems of linear equations.

Practical Uses

- **Works only for square matrices with non-zero determinant.**
- **Like an "undo" operation in mathematics.**
- **Important in regression and optimization problems.**

Inverse of a Matrix

The inverse of a matrix A is a matr A^{-1} such that when it is multiplied by A, it results in the identity I: $A \times A^{-1} = I$

Conditions

:

- A matrix must be square (same number of rows and columns) and have a non-zero determinant to have an inverse.

Formula for a 2x2 Matrix

For a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the inverse is:

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Inverse of a Matrix

Example

For $A = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$:

$$\det(A) = (4 \cdot 6) - (7 \cdot 2) = 24 - 14 = 10$$

$$A^{-1} = \frac{1}{10} \begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix} = \begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix}$$

Transpose of a Matrix

The transpose of a matrix is obtained by flipping the matrix over its diagonal, converting rows into columns.

Notation:

If A is a matrix, the transpose is denoted by A^T

Example

For $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$:

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Summary

- **Matrix multiplication** is crucial for data transformations and neural network operations.
- **Determinants** help determine the properties of matrices, such as invertibility.
- **Matrix inverses** are used to solve linear equations and optimize models.
- **Matrix transposition** is essential for aligning data dimensions and preparing matrices for multiplication.

Types of Matrices

Square Matrix

A matrix with the same number of rows and columns (e.g., n×n). It’s used often in linear transformations and finding determinants.

Example: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Types of Matrices

Diagonal Matrix

A square matrix where all non-diagonal elements are zero. Only the diagonal elements may be non-zero.

Example: $\begin{bmatrix} 5 & 0 \\ 0 & 3 \end{bmatrix}$

Types of Matrices

Identity Matrix

A special type of diagonal matrix where all the diagonal elements are 1. It acts as the multiplicative identity in matrix operations.

Example: $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Types of Matrices

Zero Matrix

A matrix where all elements are zero. It's the additive identity in matrix operations.

Example: $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Types of Matrices

Row Matrix

A matrix with only one row.

Example: $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

Types of Matrices

Symmetric Matrix

A matrix with only one row.

A square matrix that is equal to its transpose $(A = A^T)$

Example: $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$

Linear Transformations and Eigenvalues

Linear Transformations and Eigenvalues

This segment covers the concepts of linear transformations and eigenvalues, which are fundamental in understanding how data can be manipulated and analyzed in various dimensions. These concepts play a crucial role in areas such as dimensionality reduction, stability analysis, and optimization in AI and machine learning.

Linear Transformations

- **Rotation**
- **Scaling**
- **Reflection**

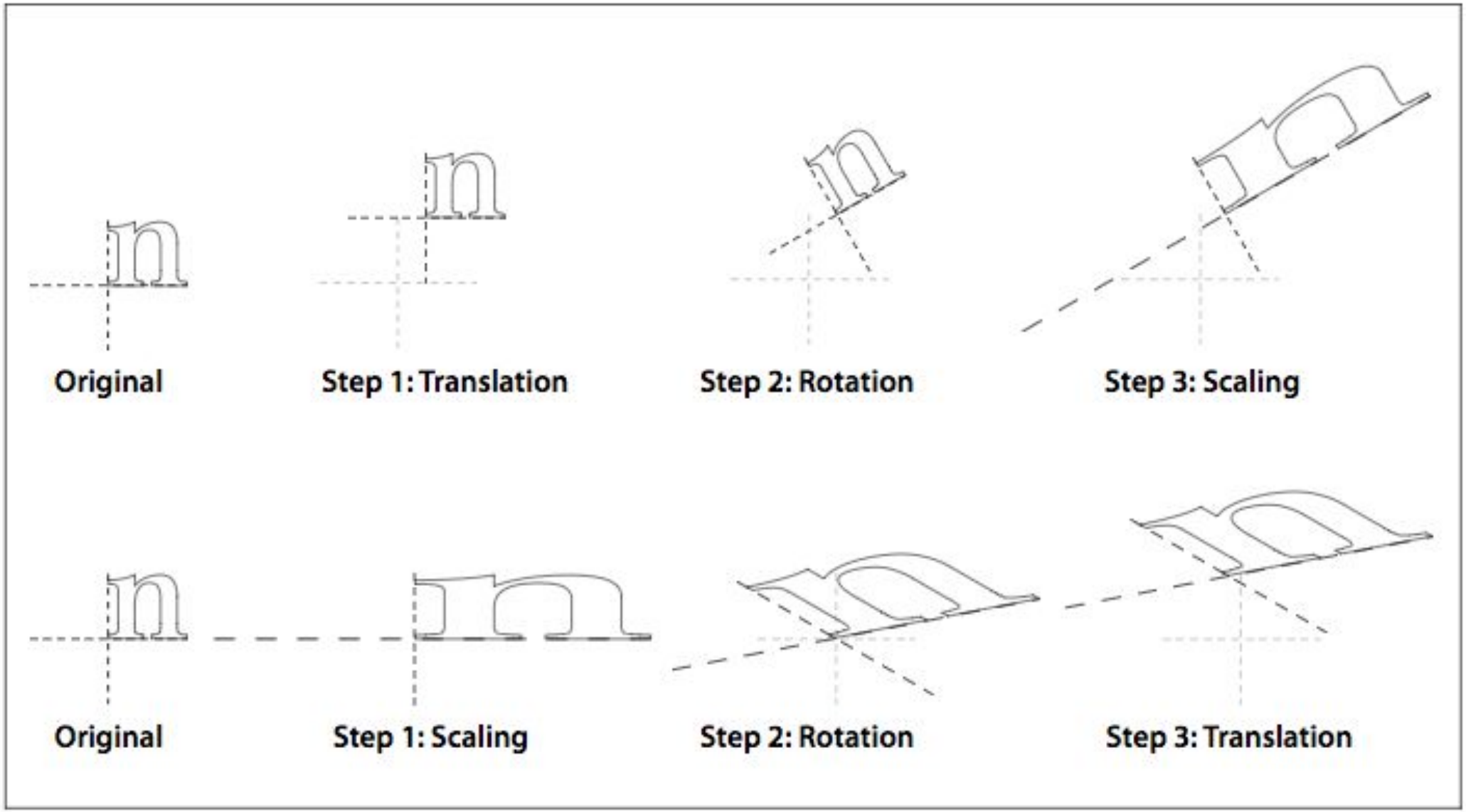


FIGURE 4.6 *Effect of transformation order*

Linear Transformations

A linear transformation is a function that maps vectors from one space to another, preserving vector addition and scalar multiplication. Essentially, it transforms vectors while maintaining the structure of the space.

Matrix Representation

Matrix Representation Linear transformations can be represented by matrices. If x is a vector and A is a matrix, then Ax is a linear transformation of x.

Example: If $A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, then:

$$A\mathbf{x} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$$

Common Types of Linear Transformations

Scaling

Changes the size of vectors. Represented by a diagonal matrix where each diagonal element scales the corresponding dimension.

Example: Scaling by a factor of 2, $A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, scales vectors by 2.

Common Types of Linear Transformations

Rotation

Rotates vectors around the origin. Represented by a rotation matrix.

Example in 2D: $A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$ rotates vectors by an angle θ.

Common Types of Linear Transformations

Reflection

Flips vectors over a specified line (in 2D) or plane (in 3D).

Example in 2D: Reflection over the x-axis, $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

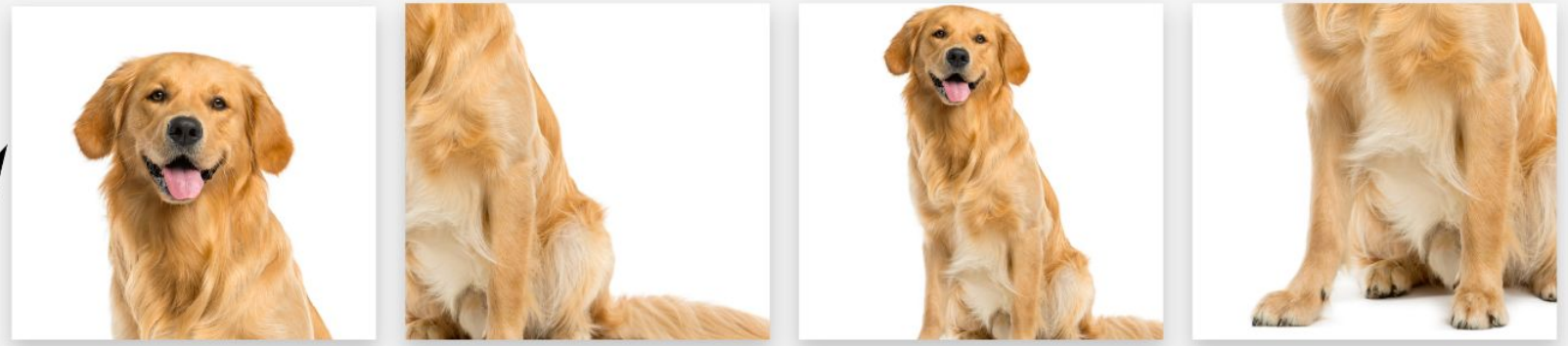
Common Types of Linear Transformations

Shearing

Distorts the shape of vectors in one direction while keeping the other direction unchanged.

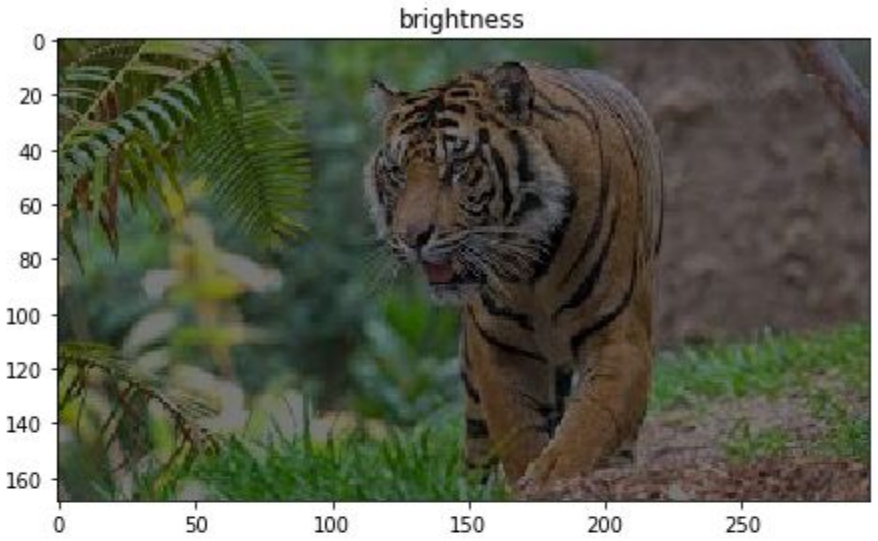
Example in 2D Horizontal shear in 2D, $A = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$, where k is the shear factor.

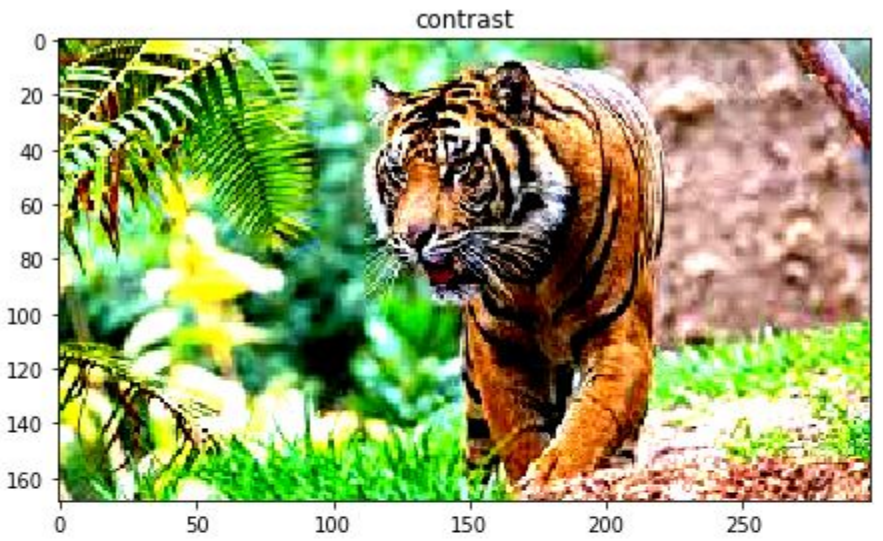
Original image



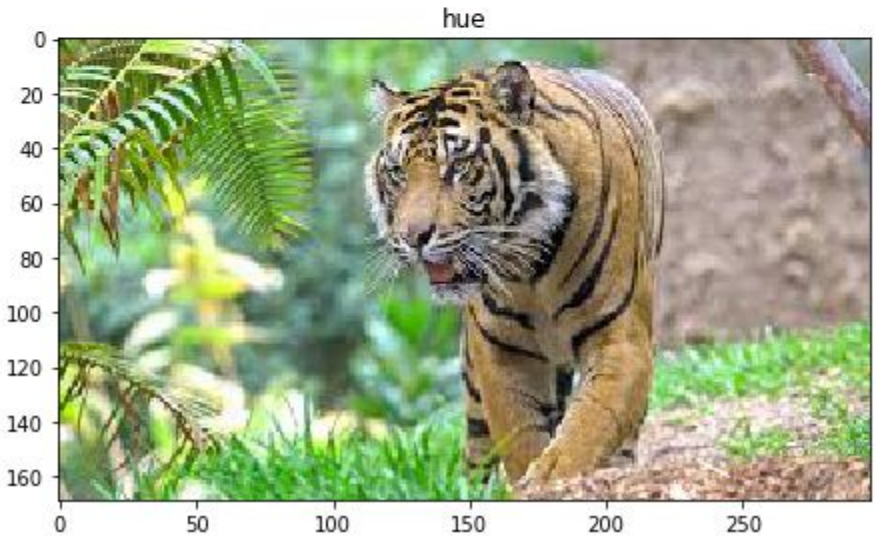
Random cropping

Random erasing


brightness
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


contrast
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


saturation
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


hue
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

Original image

Flip

Rotation

Blur

Exposure

Contrast

Grayscale

Augmented images

Dimension reduction (Feature extraction)

	1	2	3	4	5	6	7		200
	Height	Weight	Average blood pressure	Average heart rate	BMI	Cholesterol levels	Average cigarettes/day	...	Sugar levels
Person 1	150	80	140/90	63	36	5.0	0		99
Person 2	174	90	90/60	100	32	4.1	0		95
Person 3	183	109	120/80	95	29	3.6	1		92
Person 4	186	95	122/75	84	28	4.8	5		89
Person 5	170	67	95/60	76	23	2.7	10		100
Person 6	180	82	92/60	78	25	3.7	10		112
Person 7	165	71	124/80	81	26	3.8	0		113
Person 8	172	78	97/70	90	24	3.4	0		100
...									
Person 20	190	75	90/60	78	21	4.2	0		82

200 FACTORS (VARIABLES)

Principal Component Analysis (PCA)

PC1	PC2	PC3	PC4	PC5
-1	3	-1	4	4
2	4	2	5	5
3	2	4	2	2
4	4	5	-4	-4
5	5	2	2	5
2	5	-4	3	2
-4	-6	5	5	-4
-3	-6	-6	2	5
8	-3	-6	-3	-6

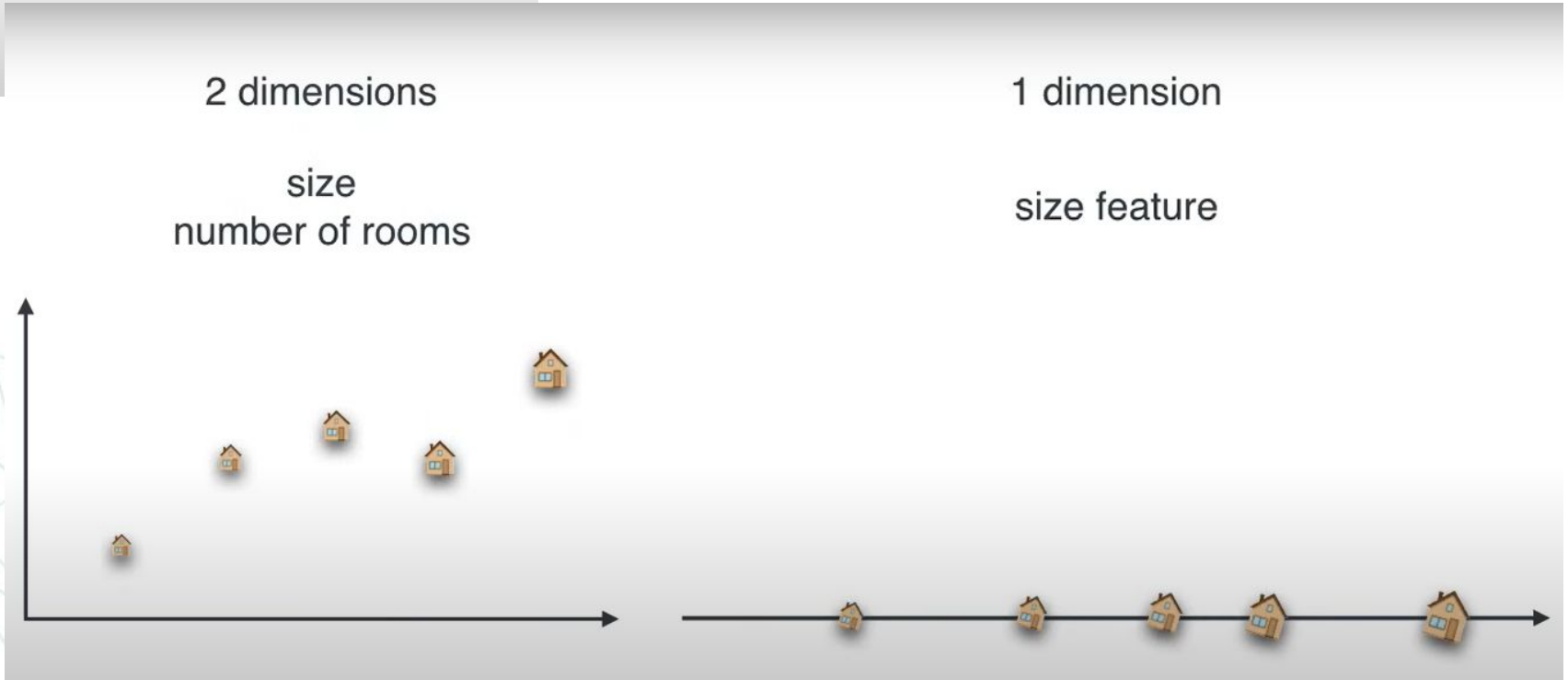
5 PRINCIPAL COMPONENTS

Housing Data

Dimension reduction (Feature extraction)

Size
Number of rooms
Number of bathrooms
→ Size feature

Schools around
Crime rate
→ Location feature



برتب ال Eigenvalues من الاكبر للاصغر
بحتفظ بالاكبر اكبر Eigenvalues اكبر variance
Eigenvectors (PC) <<

Eigenvalues and **Eigenvectors**.

Now that we've seen how NumPy helps us create and manipulate matrices, let's move one step deeper into *Linear Algebra concepts* that are very important in Data Science and Machine Learning.

Two of the most fundamental ideas are **Eigenvalues** and **Eigenvectors**.

- They may sound a bit abstract at first, but they are actually everywhere in Data Science.
- From **PCA for dimensionality reduction**, to **understanding covariance in datasets**, to **optimizing machine learning models** – these concepts play a key role.

Think of them as the tools that tell us the *main directions of information* in our data, and how strong each direction is.

الـ Eigenvectors بتعطينا أهم الاتجاهات في بياناتنا،

والـ Eigenvalues بتحكيلنا كم هذا الاتجاه مهم.

1. Eigenvalues & Eigenvectors

Imagine you have a **matrix (Matrix)** that represents a transformation (like rotation, stretching, or compression).

- **Eigenvector:** a direction (vector) that does not change its direction after the transformation, it may only get stretched or shrunk.

- **Eigenvalue:** the number that tells you how much the eigenvector is stretched or shrunk.

- eigenvector = fixed direction, eigenvalue = scaling factor.

2. Eigenvalues & Covariance Matrix

- The **Covariance Matrix** describes how features (variables) are correlated with each other.
- If we perform **Eigen Decomposition** on it:
  - **Eigenvectors =** the directions that explain the largest amount of variance in the data.
  - **Eigenvalues =** how much variance is explained along each direction.

This is exactly what happens in **PCA (Principal Component Analysis):**

1. Sort eigenvalues from largest to smallest.
2. Keep the top eigenvectors → reduce dimensions while preserving most of the variance.

Eigenvalues and Eigenvectors

In the context of a linear transformation, an eigenvector of a matrix A is a non-zero vector that only changes by a scalar factor when A is applied to it. The scalar is known as the eigenvalue.

Eigenvalue (λ): A scalar indicating how much the eigenvector is stretched or compressed.

Eigenvector (v): A vector that does not change its direction during the transformation.

Eigenvalues and Eigenvectors

Mathematical Formulation

$$A\mathbf{v} = \lambda\mathbf{v}$$

Where:

- **A** is the transformation matrix.
- **v** is the eigenvector.
- **λ** is the eigenvalue.

Eigenvalues and Eigenvectors

Finding Eigenvalues and Eigenvectors

- To find eigenvalues, solve the characteristic equation:

$$\det(A - \lambda I) = 0$$

- Once eigenvalues (λ) are known, solve $(A-\lambda I)v=0$ to find the corresponding eigenvectors.

Eigenvalues and Eigenvectors

Example: Let $A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$:

1. Calculate the characteristic equation:

$$\det(A - \lambda I) = \det \begin{bmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{bmatrix} = (4-\lambda)(3-\lambda) - 2 \cdot 1 = 0$$

2. Solve for λ\lambdaλ:

$$\lambda^2 - 7\lambda + 10 = 0 \Rightarrow (\lambda - 5)(\lambda - 2) = 0$$

So, the eigenvalues are λ=5 and λ=2.

3. Substitute each λ back into (A−λI)v=0 to find the eigenvectors.

For $\lambda = 5$:

Solve the equation $(\mathbf{A} - 5\mathbf{I})\mathbf{v} = 0$:

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

This gives the eigenvector:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

For $\lambda = 2$:

Now solve the equation $(\mathbf{A} - 2\mathbf{I})\mathbf{v} = 0$:

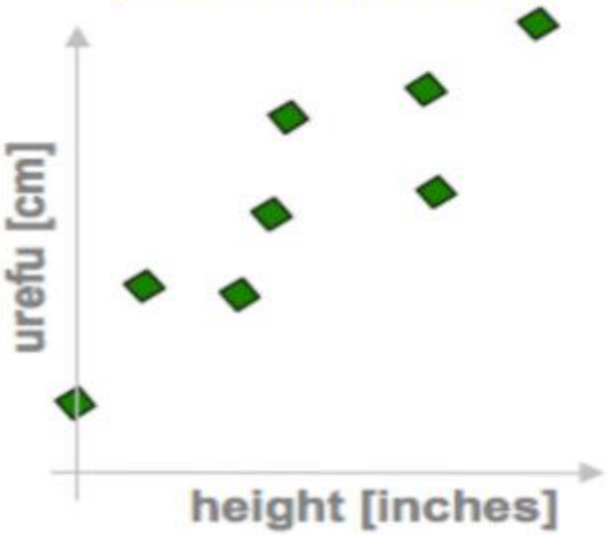
$$\begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This gives the eigenvector:

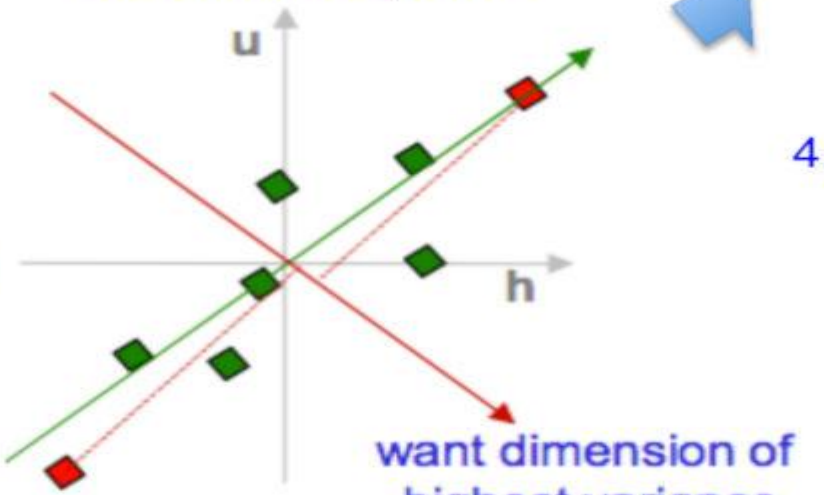
$$\mathbf{v}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

PCA in a nutshell

1. correlated hi-d data
("urefu" means "height" in Swahili)



2. center the points



3. compute covariance matrix

$$\begin{array}{c} \\ h \\ u \end{array}\begin{array}{c} \begin{array}{cc} h & u \end{array} \\ \begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \end{array} \rightarrow \text{cov}(h,u) = \frac{1}{n}\sum_{i=1}^{n} h_i u_i$$

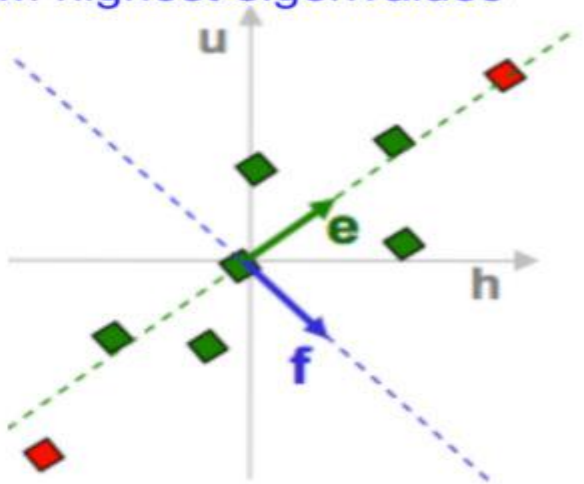
4. eigenvectors + eigenvalues

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} e_h \\ e_u \end{bmatrix} = \lambda_e \begin{bmatrix} e_h \\ e_u \end{bmatrix}$$

$$\begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} f_h \\ f_u \end{bmatrix} = \lambda_f \begin{bmatrix} f_h \\ f_u \end{bmatrix}$$

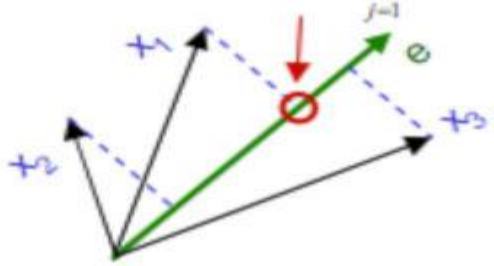
```
eig(cov(data))
```

5. pick m<d eigenvectors
w. highest eigenvalues



6. project data points to those eigenvectors

$$x_e^{'} = x^T e = \sum_{j=1}^{a} x_{ij} e_j$$



7. uncorrelated low-d data