Principal Component Analysis

Dimensionality Reduction and the Maximum Variance Perspective

Based on Bishop PRML Chapter 12

November 9, 2025

The Dimensionality Challenge

Modern machine learning confronts us with high-dimensional data:

- Images: 28×28 grayscale $= 784$ dimensions
- Genomics: thousands of gene expressions per sample
- Text: vocabulary size in tens of thousands
- Sensor arrays: hundreds of simultaneous measurements

The Dimensionality Challenge

Modern machine learning confronts us with high-dimensional data:

- Images: 28×28 grayscale $= 784$ dimensions
- Genomics: thousands of gene expressions per sample
- Text: vocabulary size in tens of thousands
- Sensor arrays: hundreds of simultaneous measurements

Problems we face:

- Cannot visualize beyond 3D
- Computational cost scales poorly with dimension
- Many algorithms struggle in high dimensions (curse of dimensionality)

What We Want

An ideal dimensionality reduction method should:

- Find a low-dimensional representation of high-dimensional data

What We Want

An ideal dimensionality reduction method should:

- Find a low-dimensional representation of high-dimensional data
- Preserve the “meaningful” structure in the data

What We Want

An ideal dimensionality reduction method should:

- Find a low-dimensional representation of high-dimensional data
- Preserve the “meaningful” structure in the data
- Enable visualization and interpretation

What We Want

An ideal dimensionality reduction method should:

- Find a low-dimensional representation of high-dimensional data
- Preserve the "meaningful" structure in the data
- Enable visualization and interpretation
- Provide efficient computation

What We Want

An ideal dimensionality reduction method should:

- Find a low-dimensional representation of high-dimensional data
- Preserve the "meaningful" structure in the data
- Enable visualization and interpretation
- Provide efficient computation
- Serve as preprocessing for downstream tasks

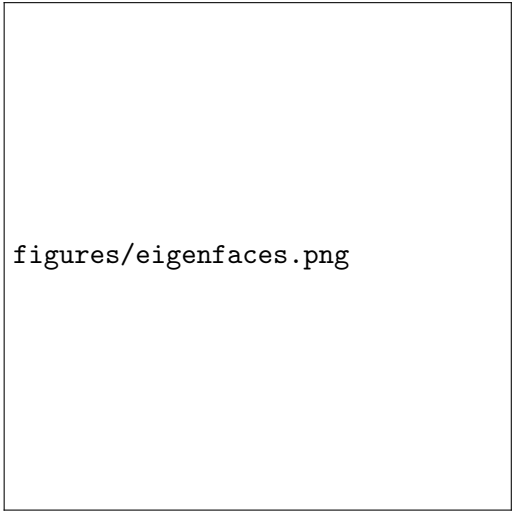
What We Want

An ideal dimensionality reduction method should:

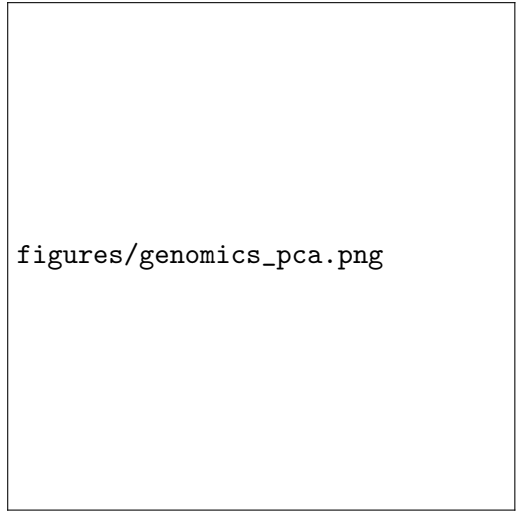
- Find a low-dimensional representation of high-dimensional data
- Preserve the "meaningful" structure in the data
- Enable visualization and interpretation
- Provide efficient computation
- Serve as preprocessing for downstream tasks

The central question: What does "meaningful" mean mathematically?

Real-World Applications



(a) Face recognition using eigenfaces



(b) Gene expression analysis

The Core Question

Given data in D dimensions, we want to project it to M dimensions where $M \ll D$.

The Core Question

Given data in D dimensions, we want to project it to M dimensions where $M \ll D$.

There are infinitely many ways to project high-dimensional data to lower dimensions.

The Core Question

Given data in D dimensions, we want to project it to M dimensions where $M \ll D$.

There are infinitely many ways to project high-dimensional data to lower dimensions.

Which projection is "best"?

The Core Question

Given data in D dimensions, we want to project it to M dimensions where $M \ll D$.

There are infinitely many ways to project high-dimensional data to lower dimensions.

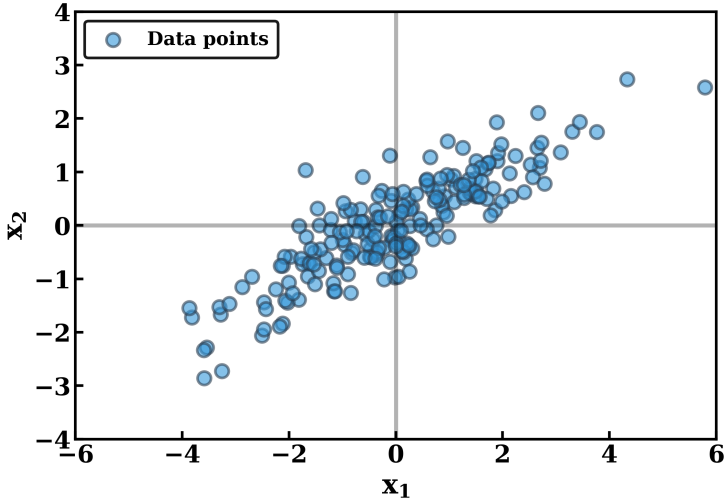
Which projection is "best"?

Principal Component Analysis provides an answer based on two equivalent perspectives:

1. **Maximum variance:** Keep directions with highest variance
2. **Minimum error:** Minimize information loss from projection

Both lead to the same solution.

A Simple 2D Example



2D data points forming an elliptical cloud

Consider data naturally lying along an elongated cloud. There's clearly more "spread"

Intuition: Direction of Spread

The data varies more in some directions than in others.

Intuition: Direction of Spread

The data varies more in some directions than in others.

Key insight: Directions with high variance contain more "information" about the structure of the data.

Intuition: Direction of Spread

The data varies more in some directions than in others.

Key insight: Directions with high variance contain more "information" about the structure of the data.

If we must reduce from 2D to 1D, we should keep the direction along which the data varies most.

Intuition: Direction of Spread

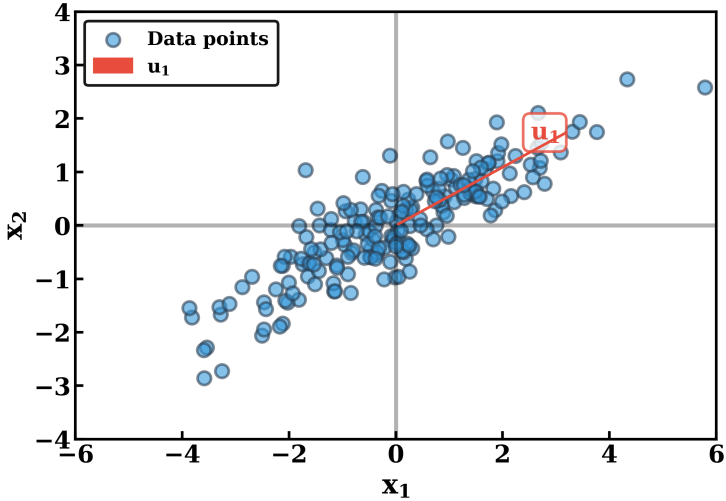
The data varies more in some directions than in others.

Key insight: Directions with high variance contain more "information" about the structure of the data.

If we must reduce from 2D to 1D, we should keep the direction along which the data varies most.

This direction of maximum variance is the **first principal component**.

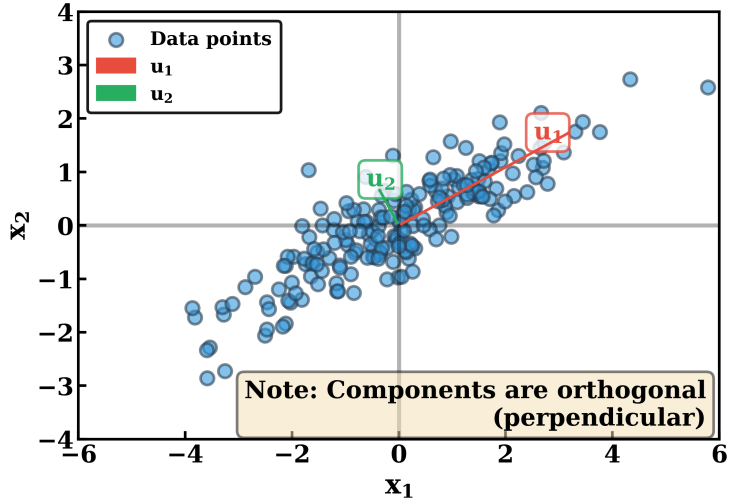
Visual: First Principal Component



First principal component $\mathbf{u}_1$ *captures maximum variance*

The first principal component $\mathbf{u}_1$ points in the direction of maximum variance

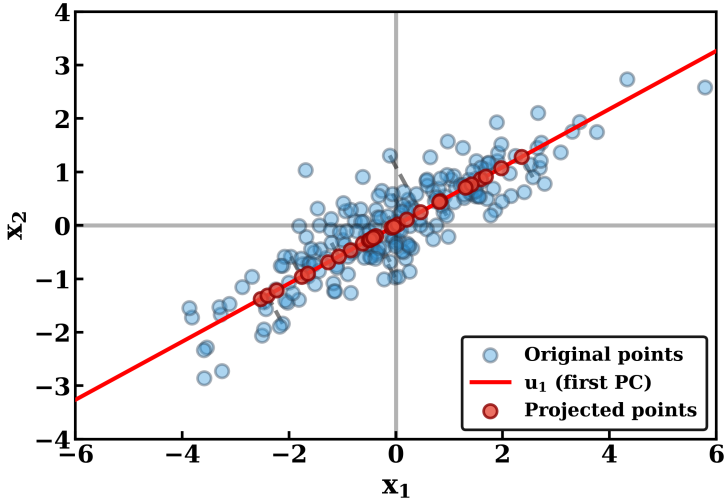
Visual: Second Principal Component



Both principal components shown as orthogonal vectors

The second principal component $\mathbf{u}_2$ is orthogonal to the first and captures the

Projection Intuition



Projection onto first principal component reduces dimensionality

Projecting onto $\mathbf{u}_1$ reduces dimensionality from 2D to 1D while preserving maximum

Information Preservation

Original data: 2 dimensions (x_1, x_2)
Reduced data: 1 dimension (coordinate along $\mathbf{u}_1$)

Information Preservation

Original data: 2 dimensions (x_1, x_2)
Reduced data: 1 dimension (coordinate along $\mathbf{u}_1$)

What did we lose?

- Variance in the perpendicular direction ($\mathbf{u}_2$ direction)
- This is the "small" variance—less important

Information Preservation

Original data: 2 dimensions (x_1, x_2)
Reduced data: 1 dimension (coordinate along $\mathbf{u}_1$)

What did we lose?

- Variance in the perpendicular direction ($\mathbf{u}_2$ direction)
- This is the "small" variance—less important

What did we keep?

- Variance in the principal direction
- This is the "large" variance—more important
- The overall structure and spread of the data

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)
2. Ordered by the variance of data along each direction

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)
2. Ordered by the variance of data along each direction
3. First component = maximum variance direction

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)
2. Ordered by the variance of data along each direction
3. First component = maximum variance direction
4. Second component = maximum variance among directions orthogonal to first

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)
2. Ordered by the variance of data along each direction
3. First component = maximum variance direction
4. Second component = maximum variance among directions orthogonal to first
5. And so on...

The Key Insight

Principal Component Analysis finds:

1. A set of orthogonal directions (principal components)
2. Ordered by the variance of data along each direction
3. First component = maximum variance direction
4. Second component = maximum variance among directions orthogonal to first
5. And so on...

For dimensionality reduction: project data onto the top M principal components.

This captures as much variance as possible in M dimensions.

Setup: The Data Matrix

Consider a dataset of N observations, each in D dimensions:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \quad \text{where } \mathbf{x}_n \in \mathbb{R}^D$$

Setup: The Data Matrix

Consider a dataset of N observations, each in D dimensions:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \quad \text{where } \mathbf{x}_n \in \mathbb{R}^D$$

The sample mean is:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

Setup: The Data Matrix

Consider a dataset of N observations, each in D dimensions:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \quad \text{where } \mathbf{x}_n \in \mathbb{R}^D$$

The sample mean is:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

Mean-centered data:

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$$

Setup: The Data Matrix

Consider a dataset of N observations, each in D dimensions:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \quad \text{where } \mathbf{x}_n \in \mathbb{R}^D$$

The sample mean is:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$$

Mean-centered data:

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$$

Why centering matters: PCA finds directions of maximum variance around the mean. Without centering, we'd be finding variance around the origin, which is not meaningful.

Finding the First Principal Component

We seek a direction $\mathbf{u}_1 \in \mathbb{R}^D$ such that the variance of the data projected onto this direction is maximized.

Finding the First Principal Component

We seek a direction $\mathbf{u}_1 \in \mathbb{R}^D$ such that the variance of the data projected onto this direction is maximized.

Constraint: $\mathbf{u}_1$ must be a unit vector:

$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

Finding the First Principal Component

We seek a direction $\mathbf{u}_1 \in \mathbb{R}^D$ such that the variance of the data projected onto this direction is maximized.

Constraint: $\mathbf{u}_1$ must be a unit vector:

$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

(Without this constraint, we could make variance arbitrarily large by scaling $\mathbf{u}_1$.)

Finding the First Principal Component

We seek a direction $\mathbf{u}_1 \in \mathbb{R}^D$ such that the variance of the data projected onto this direction is maximized.

Constraint: $\mathbf{u}_1$ must be a unit vector:

$$\mathbf{u}_1^T \mathbf{u}_1 = 1$$

(Without this constraint, we could make variance arbitrarily large by scaling $\mathbf{u}_1$.)

The projection of each data point $\tilde{\mathbf{x}}_n$ onto $\mathbf{u}_1$ is:

$$\mathbf{u}_1^T \tilde{\mathbf{x}}_n$$

This is a scalar—the coordinate along $\mathbf{u}_1$.

Variance of Projections

The projected data are the scalars: $\mathbf{u}_1^T \tilde{\mathbf{x}}_1, \mathbf{u}_1^T \tilde{\mathbf{x}}_2, \ldots, \mathbf{u}_1^T \tilde{\mathbf{x}}_N$

Variance of Projections

The projected data are the scalars: $\mathbf{u}_1^T\tilde{\mathbf{x}}_1, \mathbf{u}_1^T\tilde{\mathbf{x}}_2, \ldots, \mathbf{u}_1^T\tilde{\mathbf{x}}_N$

Since the data are mean-centered, the mean of projections is zero:

$$\frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n = \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\right) = \mathbf{u}_1^T\mathbf{0} = 0$$

Variance of Projections

The projected data are the scalars: $\mathbf{u}_1^T\tilde{\mathbf{x}}_1, \mathbf{u}_1^T\tilde{\mathbf{x}}_2, \ldots, \mathbf{u}_1^T\tilde{\mathbf{x}}_N$

Since the data are mean-centered, the mean of projections is zero:

$$\frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n = \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\right) = \mathbf{u}_1^T\mathbf{0} = 0$$

The sample variance of projections is:

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_n)^2$$

Goal: Maximize this quantity over choice of $\mathbf{u}_1$.

The Covariance Matrix

We can rewrite the variance more compactly using the data covariance matrix.

The Covariance Matrix

We can rewrite the variance more compactly using the data covariance matrix.

Define the **sample covariance matrix**:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$$

The Covariance Matrix

We can rewrite the variance more compactly using the data covariance matrix.

Define the **sample covariance matrix**:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$$

This is a $D \times D$ matrix capturing the variance and covariance structure of the data.

The Covariance Matrix

We can rewrite the variance more compactly using the data covariance matrix.

Define the **sample covariance matrix**:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$$

This is a $D \times D$ matrix capturing the variance and covariance structure of the data.

Note: $\mathbf{S}$ is symmetric and positive semi-definite.

Variance in Matrix Form

The variance of projections can now be written as:

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_n)^2 &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\mathbf{u}_1 \\ &= \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\right)\mathbf{u}_1 \\ &= \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1\end{aligned}$$

Our optimization problem:

$$\max_{\mathbf{u}_1}\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 \quad \text{subject to} \quad \mathbf{u}_1^T\mathbf{u}_1 = 1$$

Variance in Matrix Form

The variance of projections can now be written as:

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_n)^2 = \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\mathbf{u}_1$$

$$= \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$$

Our optimization problem:

$$\max_{\mathbf{u}_1}\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 \quad \text{subject to} \quad \mathbf{u}_1^T\mathbf{u}_1 = 1$$

Variance in Matrix Form

The variance of projections can now be written as:

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_n)^2 &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\mathbf{u}_1 \\ &= \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\right)\mathbf{u}_1 \\ &= \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1\end{aligned}$$

Variance in Matrix Form

The variance of projections can now be written as:

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\tilde{\mathbf{x}}_n)^2 &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\mathbf{u}_1 \\ &= \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\tilde{\mathbf{x}}_n^T\right)\mathbf{u}_1 \\ &= \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1\end{aligned}$$

Our optimization problem:

$$\max_{\mathbf{u}_1}\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 \quad \text{subject to} \quad \mathbf{u}_1^T\mathbf{u}_1 = 1$$

Constrained Optimization

We use the method of Lagrange multipliers.

Constrained Optimization

We use the method of Lagrange multipliers.

Form the Lagrangian:

$$\mathcal{L}(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

Constrained Optimization

We use the method of Lagrange multipliers.

Form the Lagrangian:

$$\mathcal{L}(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

where λ_1 is the Lagrange multiplier.

Constrained Optimization

We use the method of Lagrange multipliers.

Form the Lagrangian:

$$\mathcal{L}(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

where λ_1 is the Lagrange multiplier.

Taking the derivative with respect to $\mathbf{u}_1$ and setting to zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_1} = 2\mathbf{S}\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = \mathbf{0}$$

The Solution Emerges

From the stationarity condition:

$$2\mathbf{S}\mathbf{u}_1 - 2\lambda_1\mathbf{u}_1 = \mathbf{0}$$

The Solution Emerges

From the stationarity condition:

$$2\mathbf{S}\mathbf{u}_1 - 2\lambda_1\mathbf{u}_1 = \mathbf{0}$$

Simplifying:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

The Solution Emerges

From the stationarity condition:

$$2\mathbf{S}\mathbf{u}_1 - 2\lambda_1\mathbf{u}_1 = \mathbf{0}$$

Simplifying:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

This is an eigenvalue equation!

- $\mathbf{u}_1$ is an eigenvector of the covariance matrix $\mathbf{S}$
- λ_1 is the corresponding eigenvalue

See Bishop eq. 12.6

Which Eigenvector?

The covariance matrix $\mathbf{S}$ has D eigenvectors (assuming $N \geq D$).
Which one do we choose?

Which Eigenvector?

The covariance matrix $\mathbf{S}$ has D eigenvectors (assuming $N \geq D$).
Which one do we choose?

Multiply the eigenvalue equation by $\mathbf{u}_1^T$:

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1^T \mathbf{u}_1 = \lambda_1$$

Which Eigenvector?

The covariance matrix $\mathbf{S}$ has D eigenvectors (assuming $N \geq D$).
Which one do we choose?

Multiply the eigenvalue equation by $\mathbf{u}_1^T$:

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1^T \mathbf{u}_1 = \lambda_1$$

But $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$ is exactly the variance we're trying to maximize!

Which Eigenvector?

The covariance matrix $\mathbf{S}$ has D eigenvectors (assuming $N \geq D$).
Which one do we choose?

Multiply the eigenvalue equation by $\mathbf{u}_1^T$:

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1^T \mathbf{u}_1 = \lambda_1$$

But $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$ is exactly the variance we're trying to maximize!

Conclusion: The variance captured equals the eigenvalue λ_1.

To maximize variance, choose the eigenvector with the **largest eigenvalue**.

Subsequent Principal Components

For the second principal component $\mathbf{u}_2$:

- Maximize variance of projections onto $\mathbf{u}_2$

Subsequent Principal Components

For the second principal component $\mathbf{u}_2$:

- Maximize variance of projections onto $\mathbf{u}_2$
- Subject to: $\mathbf{u}_2^T \mathbf{u}_2 = 1$ (unit vector)

Subsequent Principal Components

For the second principal component $\mathbf{u}_2$:

- Maximize variance of projections onto $\mathbf{u}_2$
- Subject to: $\mathbf{u}_2^T \mathbf{u}_2 = 1$ (unit vector)
- **And**: $\mathbf{u}_2^T \mathbf{u}_1 = 0$ (orthogonal to first PC)

Subsequent Principal Components

For the second principal component $\mathbf{u}_2$:

- Maximize variance of projections onto $\mathbf{u}_2$
- Subject to: $\mathbf{u}_2^T \mathbf{u}_2 = 1$ (unit vector)
- **And**: $\mathbf{u}_2^T \mathbf{u}_1 = 0$ (orthogonal to first PC)

Following similar derivation, we find:

$$\mathbf{S}\mathbf{u}_2 = \lambda_2 \mathbf{u}_2$$

where $\mathbf{u}_2$ is the eigenvector with the second-largest eigenvalue λ_2.

Subsequent Principal Components

For the second principal component $\mathbf{u}_2$:

- Maximize variance of projections onto $\mathbf{u}_2$
- Subject to: $\mathbf{u}_2^T \mathbf{u}_2 = 1$ (unit vector)
- **And**: $\mathbf{u}_2^T \mathbf{u}_1 = 0$ (orthogonal to first PC)

Following similar derivation, we find:

$$\mathbf{S}\mathbf{u}_2 = \lambda_2 \mathbf{u}_2$$

where $\mathbf{u}_2$ is the eigenvector with the second-largest eigenvalue λ_2.

General pattern: The i-th principal component is the eigenvector corresponding to the i-th largest eigenvalue.

Complete Solution

The covariance matrix has eigendecomposition:

$$\mathbf{S} = \sum_{i=1}^{D} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$.

Complete Solution

The covariance matrix has eigendecomposition:

$$\mathbf{S} = \sum_{i=1}^{D} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$.

The principal components are:

- $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_D$ (eigenvectors)
- Ordered by their corresponding eigenvalues
- Form an orthonormal basis of $\mathbb{R}^D$

Complete Solution

The covariance matrix has eigendecomposition:

$$\mathbf{S} = \sum_{i=1}^{D} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$.

The principal components are:

- $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_D$ (eigenvectors)
- Ordered by their corresponding eigenvalues
- Form an orthonormal basis of $\mathbb{R}^D$

The variance captured by the i-th component is λ_i.

Summary of Maximum Variance Formulation

Problem: Find low-dimensional projection preserving maximum variance.

Solution:

1. Compute the sample covariance matrix $\mathbf{S}$
2. Find eigenvalues and eigenvectors of $\mathbf{S}$
3. Sort eigenvalues: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$
4. The first M principal components are the eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_M$
5. Project data: $\mathbf{z}_n = \mathbf{U}_M^T \tilde{\mathbf{x}}_n$ where $\mathbf{U}_M = [\mathbf{u}_1 \cdots \mathbf{u}_M]$

Total variance captured: $\sum_{i=1}^{M} \lambda_i$
Total variance in data: $\sum_{i=1}^{D} \lambda_i = \text{tr}(\mathbf{S})$

A Different Perspective

Instead of *maximizing variance captured*...

A Different Perspective

Instead of *maximizing variance captured*...

...we can equivalently *minimize information loss* from projection.

A Different Perspective

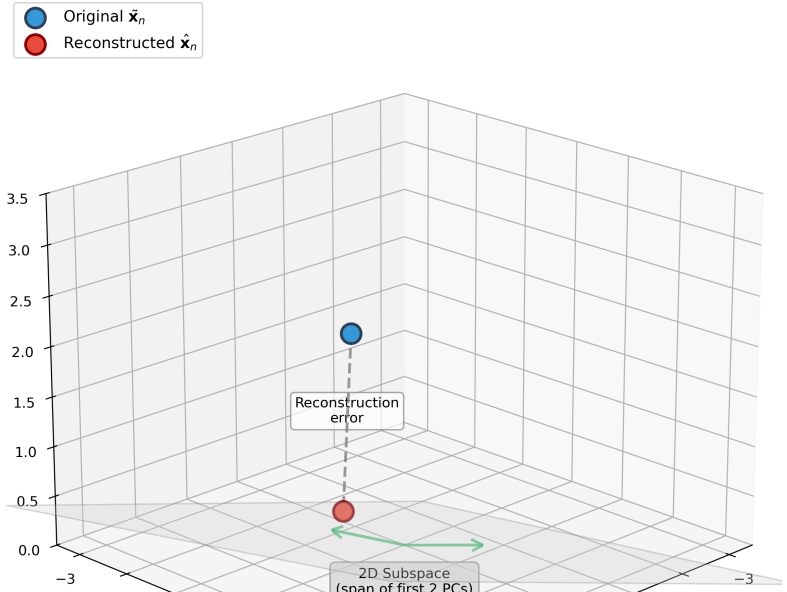
Instead of *maximizing variance captured*...

...we can equivalently *minimize information loss* from projection.

Idea:

- Project data onto M-dimensional subspace
- Reconstruct back to original D dimensions
- Measure error between original and reconstruction
- Choose subspace that minimizes this error

The Projection Operation



Mathematical Formulation

Let $\mathbf{z}_n \in \mathbb{R}^M$ be the projection of $\tilde{\mathbf{x}}_n$ onto the M-dimensional subspace.

Mathematical Formulation

Let $\mathbf{z}_n \in \mathbb{R}^M$ be the projection of $\tilde{\mathbf{x}}_n$ onto the M-dimensional subspace.

Representation in subspace:

$$\mathbf{z}_n = \mathbf{U}_M^T \tilde{\mathbf{x}}_n$$

where $\mathbf{U}_M = [\mathbf{u}_1, \ldots, \mathbf{u}_M]$ spans the subspace.

Mathematical Formulation

Let $\mathbf{z}_n \in \mathbb{R}^M$ be the projection of $\tilde{\mathbf{x}}_n$ onto the M-dimensional subspace.

Representation in subspace:

$$\mathbf{z}_n = \mathbf{U}_M^T \tilde{\mathbf{x}}_n$$

where $\mathbf{U}_M = [\mathbf{u}_1, \ldots, \mathbf{u}_M]$ spans the subspace.

Reconstruction back to D dimensions:

$$\hat{\mathbf{x}}_n = \mathbf{U}_M \mathbf{z}_n = \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n$$

Mathematical Formulation

Let $\mathbf{z}_n \in \mathbb{R}^M$ be the projection of $\tilde{\mathbf{x}}_n$ onto the M-dimensional subspace.

Representation in subspace:

$$\mathbf{z}_n = \mathbf{U}_M^T \tilde{\mathbf{x}}_n$$

where $\mathbf{U}_M = [\mathbf{u}_1, \ldots, \mathbf{u}_M]$ spans the subspace.

Reconstruction back to D dimensions:

$$\hat{\mathbf{x}}_n = \mathbf{U}_M \mathbf{z}_n = \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n$$

(Note: $\mathbf{U}_M \mathbf{U}_M^T$ is the projection matrix onto the subspace.)

Reconstruction Error

The reconstruction error for point n is:

$$\|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 = \|\tilde{\mathbf{x}}_n - \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n\|^2$$

Reconstruction Error

The reconstruction error for point n is:

$$\|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 = \|\tilde{\mathbf{x}}_n - \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n\|^2$$

The total reconstruction error across all data points:

$$J = \frac{1}{N} \sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n - \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n\|^2$$

Reconstruction Error

The reconstruction error for point n is:

$$\|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 = \|\tilde{\mathbf{x}}_n - \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n\|^2$$

The total reconstruction error across all data points:

$$J = \frac{1}{N} \sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n - \mathbf{U}_M \mathbf{U}_M^T \tilde{\mathbf{x}}_n\|^2$$

Goal: Choose the M-dimensional subspace (i.e., choose $\mathbf{U}_M$) to minimize J.

Equivalence to Maximum Variance

It can be shown that:

$$\sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 + \sum_{n=1}^{N} \|\hat{\mathbf{x}}_n\|^2 = \sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n\|^2 = \text{constant}$$

Equivalence to Maximum Variance

It can be shown that:

$$\sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 + \sum_{n=1}^{N} \|\hat{\mathbf{x}}_n\|^2 = \sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n\|^2 = \text{constant}$$

The first term is variance *lost* (reconstruction error).
The second term is variance *captured* (in the subspace).
The third term is total variance (fixed).

Equivalence to Maximum Variance

It can be shown that:

$$\sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n - \hat{\mathbf{x}}_n\|^2 + \sum_{n=1}^{N} \|\hat{\mathbf{x}}_n\|^2 = \sum_{n=1}^{N} \|\tilde{\mathbf{x}}_n\|^2 = \text{constant}$$

The first term is variance *lost* (reconstruction error).
The second term is variance *captured* (in the subspace).
The third term is total variance (fixed).

Therefore:

$$\text{Minimizing reconstruction error} \iff \text{Maximizing captured variance}$$

Both formulations yield the same solution! See Bishop §12.1.1

Intuitive Connection

Variance decomposition

Total variance = Captured variance + Lost variance

(constant) = (in subspace) + (reconstruction error)

Intuitive Connection

Variance decomposition

Total variance = Captured variance + Lost variance

(constant) = (in subspace) + (reconstruction error)

Since total variance is fixed:

- Maximizing captured variance $\Leftrightarrow$ Minimizing lost variance
- Maximum variance formulation $\Leftrightarrow$ Minimum error formulation

Intuitive Connection

Variance decomposition

Total variance = Captured variance + Lost variance

(constant) = (in subspace) + (reconstruction error)

Since total variance is fixed:

- Maximizing captured variance $\Leftrightarrow$ Minimizing lost variance
- Maximum variance formulation $\Leftrightarrow$ Minimum error formulation

Both perspectives lead to the same principal components.

Geometric Interpretation

[IMAGE PLACEHOLDER]

Description: Side-by-side comparison showing orthogonal decomposition of a data point in 2D:

- Left panel: Show original centered data point $\tilde{\mathbf{x}}_n$ as a blue arrow from origin. Show the 1D subspace (line through origin representing span of $\mathbf{u}_1$). Decompose the point into two orthogonal components: parallel component (projection onto line, in green) and perpendicular component (reconstruction error, in red). Use right angle symbol to show orthogonality.
- Right panel: Same decomposition but label green component as "Captured (large variance)" and red component as "Lost (small variance)." Show that minimizing red component is equivalent to maximizing green component.

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n
3. Compute covariance: $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n
3. Compute covariance: $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$
4. Eigendecomposition: Find eigenvalues λ_i and eigenvectors $\mathbf{u}_i$ of $\mathbf{S}$

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n
3. Compute covariance: $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$
4. Eigendecomposition: Find eigenvalues λ_i and eigenvectors $\mathbf{u}_i$ of $\mathbf{S}$
5. Sort: Order eigenvalues (and eigenvectors) in descending order

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n
3. Compute covariance: $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$
4. Eigendecomposition: Find eigenvalues λ_i and eigenvectors $\mathbf{u}_i$ of $\mathbf{S}$
5. Sort: Order eigenvalues (and eigenvectors) in descending order
6. Select: Take first M eigenvectors to form $\mathbf{U}_M = [\mathbf{u}_1 \cdots \mathbf{u}_M]$

The PCA Algorithm

Input: Data matrix $\mathbf{X}$ of size $N \times D$, desired dimension M
Algorithm:

1. Compute mean: $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$
2. Center data: $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$ for all n
3. Compute covariance: $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T$
4. Eigendecomposition: Find eigenvalues λ_i and eigenvectors $\mathbf{u}_i$ of $\mathbf{S}$
5. Sort: Order eigenvalues (and eigenvectors) in descending order
6. Select: Take first M eigenvectors to form $\mathbf{U}_M = [\mathbf{u}_1 \cdots \mathbf{u}_M]$
7. Project: $\mathbf{z}_n = \mathbf{U}_M^T \tilde{\mathbf{x}}_n$ for all n

Output: Low-dimensional representations $\mathbf{z}_1, \ldots, \mathbf{z}_N$

Computational Considerations

Covariance matrix: $D \times D$ (can be very large!)

- For $D = 10{,}000$ features, $\mathbf{S}$ requires $\sim$800 MB (double precision)
- Eigendecomposition: $O(D^3)$ complexity

Computational Considerations

Covariance matrix: $D \times D$ (can be very large!)

- For $D = 10{,}000$ features, $\mathbf{S}$ requires $\sim$800 MB (double precision)
- Eigendecomposition: $O(D^3)$ complexity

More efficient alternative: Singular Value Decomposition (SVD)

- Apply SVD directly to centered data matrix $\tilde{\mathbf{X}}$ (size $N \times D$)
- $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
- Right singular vectors (columns of $\mathbf{V}$) are the principal components
- Eigenvalues: $\lambda_i = \sigma_i^2 / N$
- More numerically stable and often faster when $N < D$

Computational Considerations

Covariance matrix: $D \times D$ (can be very large!)

- For $D = 10{,}000$ features, $\mathbf{S}$ requires $\sim$800 MB (double precision)
- Eigendecomposition: $O(D^3)$ complexity

More efficient alternative: Singular Value Decomposition (SVD)

- Apply SVD directly to centered data matrix $\tilde{\mathbf{X}}$ (size $N \times D$)
- $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
- Right singular vectors (columns of $\mathbf{V}$) are the principal components
- Eigenvalues: $\lambda_i = \sigma_i^2 / N$
- More numerically stable and often faster when $N < D$

Standard implementations (scikit-learn, R, MATLAB) use SVD internally.

How Many Components?

The fundamental question: What should M be?

How Many Components?

The fundamental question: What should M be?

No single correct answer—depends on application:

- Visualization: $M = 2$ or $M = 3$
- Dimensionality reduction for ML: determined by validation
- Compression: trade-off between size and quality

How Many Components?

The fundamental question: What should M be?

No single correct answer—depends on application:

- Visualization: $M = 2$ or $M = 3$
- Dimensionality reduction for ML: determined by validation
- Compression: trade-off between size and quality

Several diagnostic tools help us decide:

1. Scree plot (eigenvalue spectrum)
2. Cumulative variance explained
3. Cross-validation on downstream task
4. Domain knowledge

Scree Plot

[IMAGE PLACEHOLDER]

Description: A scree plot showing eigenvalue magnitude vs. component number. X-axis: "Principal Component" numbered 1 to 20. Y-axis: "Eigenvalue λ_i" (log scale preferred). The plot should show a characteristic "elbow" pattern: first few eigenvalues are large and drop rapidly, then there's an elbow around component 4-5, after which eigenvalues decrease slowly. Mark the elbow point with a dashed vertical line and annotation "Elbow suggests $M \approx 4$". Use blue circles connected by lines. Include a horizontal red dashed line showing "noise floor" for the smallest eigenvalues.

Look for the "elbow" where eigenvalues drop off sharply, then plateau.

Cumulative Variance Explained

[IMAGE PLACEHOLDER]

Description: Plot of cumulative proportion of variance explained vs. number of components. X-axis: "Number of Components M" from 1 to 20. Y-axis: "Cumulative Variance Explained" from 0 to 1 (or 0% to 100%). Show a monotonically increasing curve that starts steep and gradually flattens. Add horizontal dashed lines at 0.90, 0.95, and 0.99 with labels. Add vertical dashed lines showing how many components needed to reach each threshold (e.g., $M = 5$ for 90%, $M = 8$ for 95%, $M = 12$ for 99%). Use a smooth blue curve.

Proportion of variance explained by first M components:

$$\frac{\sum_{i=1}^{M} \lambda_i}{\sum_{i=1}^{D} \lambda_i}$$

Common thresholds: 90%, 95%, or 99% variance explained.

Data Preprocessing Matters

Centering: Always required for PCA

- Ensures we measure variance around the data mean, not the origin
- Already discussed

Data Preprocessing Matters

Centering: Always required for PCA

- Ensures we measure variance around the data mean, not the origin
- Already discussed

Scaling: Often necessary but not automatic

- PCA is sensitive to the scale of features
- Feature with large variance (e.g., income in dollars) will dominate
- Feature with small variance (e.g., age in years) will be ignored

Data Preprocessing Matters

Centering: Always required for PCA

- Ensures we measure variance around the data mean, not the origin
- Already discussed

Scaling: Often necessary but not automatic

- PCA is sensitive to the scale of features
- Feature with large variance (e.g., income in dollars) will dominate
- Feature with small variance (e.g., age in years) will be ignored

Standard practice: Standardize features to unit variance

$$x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{\sigma_d}$$

where σ_d is the standard deviation of feature d.

This ensures all features contribute on equal footing.

Practical Example Setup

We'll work through PCA on a classic dataset.

Iris Dataset:

- 150 samples of iris flowers
- 4 features: sepal length, sepal width, petal length, petal width
- 3 species: setosa, versicolor, virginica

Practical Example Setup

We'll work through PCA on a classic dataset.

Iris Dataset:

- 150 samples of iris flowers
- 4 features: sepal length, sepal width, petal length, petal width
- 3 species: setosa, versicolor, virginica

Goal:

- Reduce from 4D to 2D for visualization
- See if species structure is preserved
- Interpret what principal components represent

Practical Example Setup

We'll work through PCA on a classic dataset.

Iris Dataset:

- 150 samples of iris flowers
- 4 features: sepal length, sepal width, petal length, petal width
- 3 species: setosa, versicolor, virginica

Goal:

- Reduce from 4D to 2D for visualization
- See if species structure is preserved
- Interpret what principal components represent

This is a perfect example because:

- Low enough dimension to understand fully
- Real biological data with known structure
- Widely recognized in ML community

Example: Original High-Dimensional Data

[IMAGE PLACEHOLDER]

Description: A 2×2 grid of scatter plots showing pairwise relationships between the 4 iris features. Each subplot shows two features against each other (sepal length vs sepal width, sepal length vs petal length, etc.). Points should be colored by species: red for setosa, green for versicolor, blue for virginica. This creates a "pairs plot" or "scatter plot matrix." Label axes clearly with feature names. Show that in the original feature space, species have some separation but overlap exists in certain views.

The original 4D space shown as pairwise projections. Some separation between species is visible, but overlapping.

Covariance Structure

[IMAGE PLACEHOLDER]

Description: Heatmap of the 4×4 covariance matrix. Rows and columns labeled: sepal length, sepal width, petal length, petal width. Use a diverging colormap (e.g., blue for negative, white for zero, red for positive correlations). Strong positive correlations should appear between petal length/width and sepal length. Sepal width might show weaker or slightly negative correlation with others. Add a colorbar. Annotate cells with numerical correlation values.

The covariance matrix reveals strong correlations between certain features (e.g., petal dimensions).

Eigenvalue Spectrum

[IMAGE PLACEHOLDER]

Description: Scree plot for the Iris dataset showing 4 eigenvalues. X-axis: Component number (1-4). Y-axis: Eigenvalue magnitude. Show bars or connected points. First eigenvalue should be dominant (much larger than others), second eigenvalue moderate, third and fourth small. Include numerical labels on each point showing the actual eigenvalue. Below or beside, show the variance explained by each component as percentages (e.g., PC1: 72.9%, PC2: 22.8%, PC3: 3.7%, PC4: 0.5% — these are approximate typical values for scaled Iris data).

First two components capture $\sim$96% of variance—excellent for 2D visualization!

First Two Principal Components

[IMAGE PLACEHOLDER]

Description: 2D scatter plot of the Iris data projected onto the first two principal components. X-axis: "First Principal Component (PC1)" ranging approximately -3 to 3. Y-axis: "Second Principal Component (PC2)" ranging approximately -2 to 2. Color points by species (same colors as before: red setosa, green versicolor, blue virginica). Show that setosa is clearly separated from the other two species, while versicolor and virginica have slight overlap. This demonstrates that PCA preserves the species structure well. Add a legend showing species colors.

The 2D projection preserves species separation well. Setosa is distinctly separated; some overlap between versicolor and virginica.

Reconstruction Quality

[IMAGE PLACEHOLDER]

Description: Show reconstruction quality comparison. Create a 2×3 grid:

- Row 1: Three iris flower samples (can be stylized representations or actual iris photos)
- Row 2: Reconstructions of the same samples using 2 PCs

Add text below each column showing "Original", "2 PCs (96% var.)". If using actual measurements, show the 4 feature values as bar charts for each sample, comparing original (blue bars) vs. reconstructed (orange bars). Include reconstruction error values.

With 2 components (96% variance), reconstruction is excellent. Original and reconstructed feature values are very close.

Interpretation of Components

What do the principal components actually represent?

The first principal component $\mathbf{u}_1$ has approximate loadings:

$$\mathbf{u}_1 \approx [0.52, -0.27, 0.58, 0.56]$$

Interpretation of Components

What do the principal components actually represent?

The first principal component $\mathbf{u}_1$ has approximate loadings:

$$\mathbf{u}_1 \approx [0.52, -0.27, 0.58, 0.56]$$

Interpretation: PC1 is roughly "overall flower size"

- Large positive weights on petal length/width and sepal length
- Negative weight on sepal width
- Flowers with high PC1 scores are generally larger

Interpretation of Components

What do the principal components actually represent?

The first principal component $\mathbf{u}_1$ has approximate loadings:

$$\mathbf{u}_1 \approx [0.52, -0.27, 0.58, 0.56]$$

Interpretation: PC1 is roughly "overall flower size"

- Large positive weights on petal length/width and sepal length
- Negative weight on sepal width
- Flowers with high PC1 scores are generally larger

The second principal component focuses on contrasts between sepal and petal dimensions.

Note: Component interpretation is data-dependent and requires domain knowledge!

Assumptions of PCA

PCA makes several implicit assumptions:

1. **Linearity**
   - Principal components are linear combinations of original features
   - Projections are onto linear subspaces
   - Cannot capture nonlinear structure

Assumptions of PCA

PCA makes several implicit assumptions:

1. **Linearity**
   - Principal components are linear combinations of original features
   - Projections are onto linear subspaces
   - Cannot capture nonlinear structure

2. **Variance = Importance**
   - Assumes high-variance directions are "meaningful"
   - Works well when data is roughly Gaussian
   - Can fail when discriminative information is in low-variance directions

Assumptions of PCA

PCA makes several implicit assumptions:

1. **Linearity**
   - Principal components are linear combinations of original features
   - Projections are onto linear subspaces
   - Cannot capture nonlinear structure

2. **Variance = Importance**
   - Assumes high-variance directions are "meaningful"
   - Works well when data is roughly Gaussian
   - Can fail when discriminative information is in low-variance directions

3. **Orthogonality**
   - Components must be uncorrelated (orthogonal)
   - Sometimes natural structure isn't orthogonal

When PCA Struggles

[IMAGE PLACEHOLDER]

Description: Two examples where PCA fails:

- Left panel: "Swiss roll" - a 2D manifold embedded in 3D that looks like a rolled-up sheet. Show the 3D structure with points colored by their position along the roll. The structure is nonlinear—unrolling it requires nonlinear methods. Show that PCA would capture the major linear trends but miss the rolled structure.
- Right panel: Two classes in 2D that are separable but where the separation is in the low-variance direction. For example, two elongated parallel clusters with large variance along their length but small variance between them. PCA's first component would go along the clusters (useless for classification), while the discriminative direction is the second component (low variance).

Outlier Sensitivity

PCA is based on variance (second-moment statistic), making it sensitive to outliers.

Outlier Sensitivity

PCA is based on variance (second-moment statistic), making it sensitive to outliers.

Problem:

- Single outlier can heavily influence covariance matrix
- Principal components may be "pulled" toward outliers
- Resulting projection may not represent typical data structure

Outlier Sensitivity

PCA is based on variance (second-moment statistic), making it sensitive to outliers.

Problem:

- Single outlier can heavily influence covariance matrix
- Principal components may be "pulled" toward outliers
- Resulting projection may not represent typical data structure

Solutions:

- Robust PCA variants (using robust covariance estimators)
- Outlier detection and removal before PCA
- Regularization methods

Connections to Other Methods

PCA is related to many other techniques:

Probabilistic PCA (Bishop §12.2)

- Probabilistic latent variable model
- Provides principled handling of missing data
- Enables model comparison via likelihood

Connections to Other Methods

PCA is related to many other techniques:

Probabilistic PCA (Bishop §12.2)

- Probabilistic latent variable model
- Provides principled handling of missing data
- Enables model comparison via likelihood

Independent Component Analysis (ICA)

- Finds statistically independent (not just uncorrelated) components
- Useful for blind source separation

Connections to Other Methods

PCA is related to many other techniques:

Probabilistic PCA (Bishop §12.2)

- Probabilistic latent variable model
- Provides principled handling of missing data
- Enables model comparison via likelihood

Independent Component Analysis (ICA)

- Finds statistically independent (not just uncorrelated) components
- Useful for blind source separation

Kernel PCA

- Nonlinear generalization using kernel trick
- Can capture curved manifold structure

More Extensions

Autoencoders

- Neural network approach to dimensionality reduction
- Can learn nonlinear encodings
- Linear autoencoders recover PCA subspace

More Extensions

Autoencoders

- Neural network approach to dimensionality reduction
- Can learn nonlinear encodings
- Linear autoencoders recover PCA subspace

Sparse PCA

- Enforces sparsity in component loadings
- Improves interpretability (only few features have non-zero weights)
- Trades off variance explained for simplicity

More Extensions

Autoencoders

- Neural network approach to dimensionality reduction
- Can learn nonlinear encodings
- Linear autoencoders recover PCA subspace

Sparse PCA

- Enforces sparsity in component loadings
- Improves interpretability (only few features have non-zero weights)
- Trades off variance explained for simplicity

Supervised alternatives

- Linear Discriminant Analysis (LDA): uses class labels
- Partial Least Squares: considers both features and targets
- Better when goal is prediction, not just description

Practical Pitfalls

Common mistakes when applying PCA:

1. **Forgetting to center or scale**
   - Always center data (subtract mean)
   - Usually standardize features (divide by std. dev.)

Practical Pitfalls

Common mistakes when applying PCA:

1. **Forgetting to center or scale**
   - Always center data (subtract mean)
   - Usually standardize features (divide by std. dev.)

2. **Over-interpreting components**
   - Components are mathematical constructs, not always meaningful
   - Interpretation requires domain knowledge
   - Components can be unstable with small sample sizes

Practical Pitfalls

Common mistakes when applying PCA:

1. **Forgetting to center or scale**
   - Always center data (subtract mean)
   - Usually standardize features (divide by std. dev.)

2. **Over-interpreting components**
   - Components are mathematical constructs, not always meaningful
   - Interpretation requires domain knowledge
   - Components can be unstable with small sample sizes

3. **Wrong number of components**
   - Too few: lose important information
   - Too many: keep noise, computational waste
   - Always validate on downstream task when possible

Implementation Options

Three main approaches in Python:

1. **NumPy (manual implementation)**
   - Full control, educational value
   - Use `np.cov()` and `np.linalg.eig()`
   - Good for understanding, tedious for production

Implementation Options

Three main approaches in Python:

1. **NumPy (manual implementation)**
   - Full control, educational value
   - Use `np.cov()` and `np.linalg.eig()`
   - Good for understanding, tedious for production

2. **Scikit-learn (recommended)**
   - `sklearn.decomposition.PCA`
   - Industry standard, well-tested, efficient
   - Consistent API with other sklearn methods

Implementation Options

Three main approaches in Python:

1. **NumPy (manual implementation)**
   - Full control, educational value
   - Use `np.cov()` and `np.linalg.eig()`
   - Good for understanding, tedious for production

2. **Scikit-learn (recommended)**
   - `sklearn.decomposition.PCA`
   - Industry standard, well-tested, efficient
   - Consistent API with other sklearn methods

3. **SciPy**
   - Lower-level linear algebra routines
   - `scipy.linalg` for SVD-based approach
   - More flexible but requires more code

Scikit-learn Example: Basic Usage

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load your data (N samples, D features)
X = np.loadtxt('data.csv', delimiter=',')

# IMPORTANT: Standardize features to unit variance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit PCA - reduce to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X_scaled)

# X_reduced now has shape (N, 2)
print(f"Reduced data shape: {X_reduced.shape}")
```

Key points: **always** standardize first, then fit PCA

Analyzing Results

```python
# Variance explained by each component
print("Variance explained ratio:")
print(pca.explained_variance_ratio_)
# Output: [0.729, 0.229]  (example values)

# Cumulative variance
print("Cumulative variance:")
print(pca.explained_variance_ratio_.cumsum())
# Output: [0.729, 0.958]

# Access principal components (shape: n_components x n_features)
components = pca.components_
print(f"PC1 loadings: {components[0]}")

# Reconstruct original data (with information loss)
X_reconstructed = pca.inverse_transform(X_reduced)

# Compute reconstruction error
```

Choosing Number of Components

```python
# Strategy 1: Fit with all components first, then decide
pca_full = PCA()
pca_full.fit(X_scaled)

# Plot scree plot
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(pca_full.explained_variance_) + 1),
         pca_full.explained_variance_, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.title('Scree Plot')
plt.show()

# Strategy 2: Specify variance threshold automatically
pca_auto = PCA(n_components=0.95)  # Keep 95% variance
pca_auto.fit(X_scaled)
print(f"Number of components selected: {pca_auto.n_components_}
    ")
```

Key Concepts Recap

What is PCA?

- Unsupervised linear dimensionality reduction
- Projects data onto lower-dimensional subspace
- Preserves maximum variance (equivalently: minimizes reconstruction error)

Key Concepts Recap

What is PCA?

- Unsupervised linear dimensionality reduction
- Projects data onto lower-dimensional subspace
- Preserves maximum variance (equivalently: minimizes reconstruction error)

How does it work?

- Eigendecomposition of data covariance matrix
- Principal components = eigenvectors
- Components ordered by eigenvalues (= variance captured)

Key Concepts Recap

What is PCA?

- Unsupervised linear dimensionality reduction
- Projects data onto lower-dimensional subspace
- Preserves maximum variance (equivalently: minimizes reconstruction error)

How does it work?

- Eigendecomposition of data covariance matrix
- Principal components = eigenvectors
- Components ordered by eigenvalues (= variance captured)

Key assumptions:

- Linearity of projections
- Variance indicates importance
- Orthogonality of components

When to Use PCA

Excellent for:

- Exploratory data analysis
- Visualization of high-dimensional data (reduce to 2D/3D)
- Preprocessing for supervised learning (reduce features, speed up training)
- Noise reduction (keep top components, discard noisy ones)
- Compression (store low-dimensional representation)
- When data is approximately Gaussian with linear structure

When to Use PCA

Excellent for:

- Exploratory data analysis
- Visualization of high-dimensional data (reduce to 2D/3D)
- Preprocessing for supervised learning (reduce features, speed up training)
- Noise reduction (keep top components, discard noisy ones)
- Compression (store low-dimensional representation)
- When data is approximately Gaussian with linear structure

Consider alternatives when:

- Data lies on nonlinear manifold $\rightarrow$ Kernel PCA, t-SNE, UMAP
- Discriminative task with labels $\rightarrow$ LDA, supervised methods
- Need interpretable/sparse components $\rightarrow$ Sparse PCA, Factor Analysis
- Outliers present $\rightarrow$ Robust PCA

Looking Ahead

Topics we haven't covered:

Probabilistic PCA (Bishop §12.2)

- Latent variable model: $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$
- Principled treatment of noise and missing data
- Enables Bayesian inference, model selection

Looking Ahead

Topics we haven't covered:

Probabilistic PCA (Bishop §12.2)

- Latent variable model: $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$
- Principled treatment of noise and missing data
- Enables Bayesian inference, model selection

Advanced topics:

- Kernel PCA for nonlinear dimensionality reduction
- Factor Analysis and its relationship to PCA
- Modern deep learning approaches (variational autoencoders)
- Manifold learning methods (Isomap, LLE, t-SNE, UMAP)

Looking Ahead

Topics we haven't covered:

Probabilistic PCA (Bishop §12.2)

- Latent variable model: $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$
- Principled treatment of noise and missing data
- Enables Bayesian inference, model selection

Advanced topics:

- Kernel PCA for nonlinear dimensionality reduction
- Factor Analysis and its relationship to PCA
- Modern deep learning approaches (variational autoencoders)
- Manifold learning methods (Isomap, LLE, t-SNE, UMAP)

For supervised learning:

- Linear Discriminant Analysis (uses class labels)
- Canonical Correlation Analysis (two sets of variables)

References and Further Reading

Primary source:

- Bishop, C.M. (2006). *Pattern Recognition and Machine Learning.* Chapter 12 (§12.1)

Additional resources:

- Jolliffe, I.T. (2002). *Principal Component Analysis.* Springer. [Comprehensive treatment]
- Shlens, J. (2014). "A Tutorial on Principal Component Analysis." arXiv:1404.1100 [Accessible tutorial]
- James et al. (2013). *An Introduction to Statistical Learning.* Chapter 10.2 [Practical perspective]

Software documentation:

- Scikit-learn PCA guide: `scikit-learn.org/stable/modules/decomposition.html`

Questions?