Dimensionality Reduction with Principal Component Analysis (PCA)

GCCP

February 8, 2026

Lecture Notes: Dimensionality Reduction with Principal Component Analysis (PCA)

Learning Objectives

By the end of this lecture, you will be able to: 1. **Explain** the geometric relationship between variance, information, and coordinate rotation. 2. **Compute** (conceptually) and interpret eigenvectors and eigenvalues to identify principal components. 3. **Implement** a PCA pipeline in Python using `scikit-learn` to reduce feature space while retaining signal. 4. **Evaluate** the trade-off between dimensionality reduction and information loss using scree plots and explained variance ratios.

---

1. Variance Maximization: The Geometric Intuition

Why This Matters

In data science, we often equate "more data" with "better models." However, as you add features (dimensions), data becomes sparse, and distance metrics lose meaning—a phenomenon known as the **Curse of Dimensionality**. You need a way to compress your data without throwing away the signal. PCA allows you to reduce a dataset from 50 features to 5, not by simply deleting 45 columns, but by finding a new perspective that captures the essence of the original data.

The Core Concept: Variance is Information

To understand PCA, you must accept a fundamental premise of signal processing: **Variance equals Information**.

Imagine a dataset of students containing their "Grade Point Average" and "Number of Kidneys." The kidney variable likely has near-zero variance (almost ev-

eryone has two). It tells you nothing about the students. The GPA variable, however, varies significantly; it distinguishes one student from another. PCA seeks to find the directions (axes) in your data where the variance is highest.

PCA performs a coordinate transformation. It rotates the original axes (e.g., x and y) to align with the data's natural spread. The first new axis, called **Principal Component 1 (PC1)**, aligns with the direction of maximum variance. The second axis (PC2) aligns with the second most variance, subject to the constraint that it must be perpendicular (orthogonal) to PC1.

Analogy: The Photographer's Dilemma

Imagine you are holding a 3D teapot and you want to take a 2D photograph that best represents the object. * **Bad Angle:** If you take a photo from directly above, you just see a circle (the lid). You've lost the information about the handle, the spout, and the height. The "variance" of the teapot's shape is hidden in the dimension you collapsed. * **Good Angle:** If you take a side profile, you capture the width of the spout and the curve of the handle. You have maximized the "variance" (spread) of the object in your 2D projection.

PCA is the mathematical photographer. It rotates the object (the data) until it finds the angle that casts the widest shadow (preserves the most variance) and snaps the picture there.

Formal Definition: Projection

Mathematically, we are projecting data points onto a vector. If we have a vector $\mathbf{u}$ (with length 1) representing a direction, the projection of a data point $\mathbf{x}$ onto $\mathbf{u}$ is $\mathbf{x}^T\mathbf{u}$.

We want to find the specific vector $\mathbf{u}$ that maximizes the variance of these projections. Before doing this, we **must center the data** (subtract the mean) so the cloud of points is centered at the origin. If we denote the centered data matrix as $\mathbf{X}$, we are looking for $\mathbf{u}$ that maximizes:

$$\text{Var}(\mathbf{Xu}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^T\mathbf{u})^2$$

> **Pro Tip:** Always standardize your data (mean = 0, variance = 1) before applying PCA. If one variable is measured in millimeters (range 0–1000) and another in kilometers (range 0–1), the millimeter variable will dominate the variance simply due to scale, not informational value.

Synthesis Point

PCA does not select existing variables; it creates new ones (Principal Components) by finding linear combinations of the original features that maximize the

spread (variance) of the data points.

---

2. Eigenvectors and Eigenvalues: The Engine Room

Actionable Bridge

You now understand that we need to rotate axes to maximize variance. You will use **Eigen-decomposition** to calculate exactly *how* much to rotate and *which* resulting axes are worth keeping.

The Covariance Matrix

The search for the "best angle" begins with the **Covariance Matrix** (Σ). This matrix captures how every variable relates to every other variable.

If your dataset $\mathbf{X}$ is centered (mean subtracted), the covariance matrix is calculated as:

$$\Sigma = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$$

The diagonal elements of Σ represent the variance of each individual feature, while the off-diagonal elements represent the correlations between features.

Eigenvectors: The Directions

An **eigenvector** ($\mathbf{v}$) of a matrix is a non-zero vector that, when multiplied by that matrix, does not change direction—it only stretches or shrinks. The amount it stretches is the **eigenvalue** (λ).

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

In the context of PCA: 1. **Eigenvectors** point in the directions of the principal components (the new axes). 2. **Eigenvalues** represent the magnitude of variance captured by that component.

The eigenvector with the highest corresponding eigenvalue is PC1. The eigenvector with the second highest is PC2, and so on.

Orthogonality and Decorrelation

A crucial property of PCA is that the eigenvectors of a symmetric matrix (like the covariance matrix) are **orthogonal** (perpendicular) to each other.

This means that PCA automatically handles **multicollinearity**. If you have two highly correlated features (e.g., "Square Footage" and "Number of Rooms"), PCA will likely collapse them into a single prominent component (PC1) that

represents "Size," while a much smaller component (PC2) might capture the slight deviations between them. The resulting Principal Components are uncorrelated.

Worked Example: 2D Data Trace

Imagine a tiny dataset with 2 variables (x_1, x_2) centered at the origin:

$$\mathbf{X} = \begin{bmatrix} 1 & 0.8 \\ -1 & -0.8 \\ 2 & 1.6 \end{bmatrix}$$

1. **Calculate Covariance Matrix**: You would find that x_1 and x_2 vary together positively.
2. **Find Eigenvectors**:
   - $\mathbf{v}_1 \approx [0.78, 0.62]$: This vector points diagonally up-right. This is the direction of the "main line" of the data.
   - $\mathbf{v}_2 \approx [-0.62, 0.78]$: This vector points diagonally up-left, perpendicular to $\mathbf{v}_1$.
3. **Find Eigenvalues**:
   - $\lambda_1 \approx 5.0$: This is large, meaning $\mathbf{v}_1$ explains a lot of variance.
   - $\lambda_2 \approx 0.1$: This is tiny, meaning $\mathbf{v}_2$ explains very little (noise).

You could drop PC2 and project all points onto PC1, reducing the data to 1D while keeping nearly all the information.

Synthesis Point

The covariance matrix describes the shape of the data cloud; the eigenvectors describe the orientation of that shape (the axes); and the eigenvalues describe the length of the axes (the amount of information).

---

3. PCA Application and Interpretation

Actionable Bridge

Theory is useless without implementation. You will use this workflow when preprocessing high-dimensional data (like genomic sequences or pixel data) before feeding it into a classifier like Logistic Regression or a Support Vector Machine.

Implementation with Scikit-Learn

The standard workflow involves three steps: Standardization, Fitting, and Transformation.

```
import numpy as np
import pandas as pd
```

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1. Create dummy data: 5 features, 100 samples
# Assume 'raw_data' is a DataFrame representing customer metrics
np.random.seed(42)
raw_data = np.random.rand(100, 5)

# 2. Standardization (CRITICAL STEP)
# Scales data to mean=0, variance=1
scaler = StandardScaler()
scaled_data = scaler.fit_transform(raw_data)

# 3. Apply PCA
# Let's keep all components first to analyze variance
pca = PCA()
pca.fit(scaled_data)

# 4. Analyze Explained Variance
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
# Output might look like: [0.45, 0.30, 0.15, 0.08, 0.02]
# This means PC1 holds 45% of the dataset's information.

# 5. Transform the data
# If we decide to keep top 2 components (75% of info)
pca_final = PCA(n_components=2)
reduced_data = pca_final.fit_transform(scaled_data)

print(f"Original shape: {scaled_data.shape}") # (100, 5)
print(f"Reduced shape: {reduced_data.shape}")   # (100, 2)
```

Deciding "k": The Scree Plot

How many components (k) should you keep? You rarely keep them all. * **Explained Variance Ratio**: Look at the cumulative sum. A common threshold is to keep enough components to explain 80-95% of the variance. * **The Elbow Method (Scree Plot)**: Plot the eigenvalues (y-axis) against the component number (x-axis). The graph usually drops sharply and then levels off. The "elbow" is the point where adding more components yields diminishing returns.

Interpretation: What do the Components Mean?

A common criticism of PCA is that the new features are "black boxes." You can't just say "PC1 increased." However, you can interpret them by inspecting the **Loadings** (the components_ attribute in sklearn).

Loadings show the correlation between the original features and the principal

component.

Example Scenario: Housing Data Imagine you run PCA on housing data with features: Price, SqFt, Rooms, CrimeRate, DistanceToCity.

- **PC1 Loadings**:
  - Price: +0.90
  - SqFt: +0.85
  - Rooms: +0.80
  - CrimeRate: -0.10
  - *Interpretation*: PC1 likely represents **"Luxury/Size"**. High values mean a big, expensive house.
- **PC2 Loadings**:
  - Price: -0.10
  - CrimeRate: +0.80
  - DistanceToCity: -0.70
  - *Interpretation*: PC2 might represent **"Neighborhood Quality"** or **"Urban Density"**.

Common Pitfalls

1. **Forgetting to Scale**: If you skip `StandardScaler`, the variable with the largest raw numbers determines the principal component.
2. **Information Loss**: PCA is "lossy" compression. If the 5% of variance you discarded contained the specific anomaly you were trying to detect (e.g., fraud detection), your model will fail.
3. **Linearity Assumption**: PCA assumes data structure is linear. If your data lies on a curved manifold (like a Swiss Roll shape), PCA will flatten it incorrectly. In those cases, you need techniques like t-SNE or UMAP.

Synthesis Point

PCA transforms your data from a set of possibly correlated, noisy features into a smaller set of ranked, uncorrelated "super-features." Success depends on interpreting these new features correctly and ensuring you haven't discarded critical low-variance signal.

Key Takeaways

1. **Variance is the currency of information** in PCA; we seek axes that maximize the spread of data points to distinguish them from one another.
2. **Dimensionality reduction is a projection**, akin to finding the best shadow of a 3D object, not simply deleting columns from a spreadsheet.
3. **Standardization is mandatory** before PCA; without it, variables with large units (e.g., salary) will dominate variables with small units (e.g., age), distorting the components.

4. **Eigenvectors define the new axes** (orientation), while **Eigenvalues define the importance** (magnitude/variance explained) of those axes.
5. Principal Components are mathematically guaranteed to be **orthogonal (uncorrelated)**, which solves multicollinearity issues in downstream models.
6. Use the **Explained Variance Ratio** and **Scree Plots** to determine the optimal number of components (k) to retain, typically aiming for 80-95% cumulative variance.
7. **Interpretability requires inspecting loadings**; analyze how original features contribute to each component to understand what "PC1" actually represents in the real world.