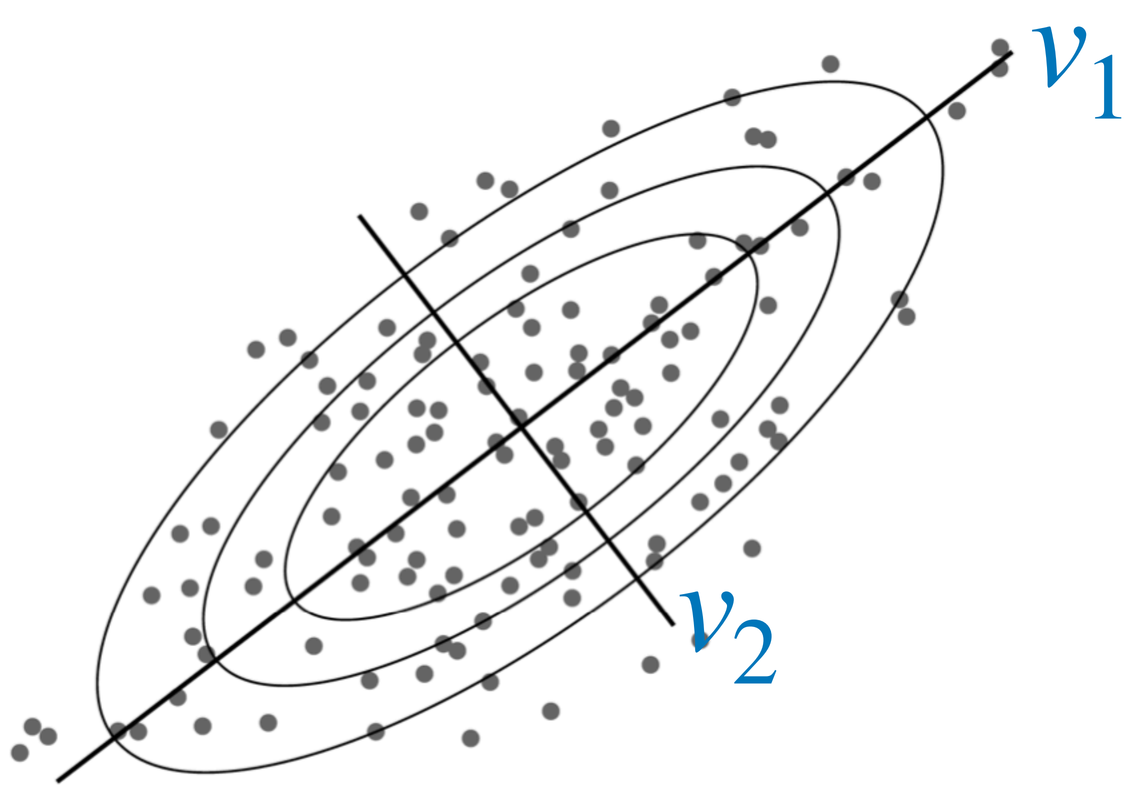
Principal Component Analysis (PCA)

Foundations of Data Analysis

March 27, 2023

Principal Component Analysis



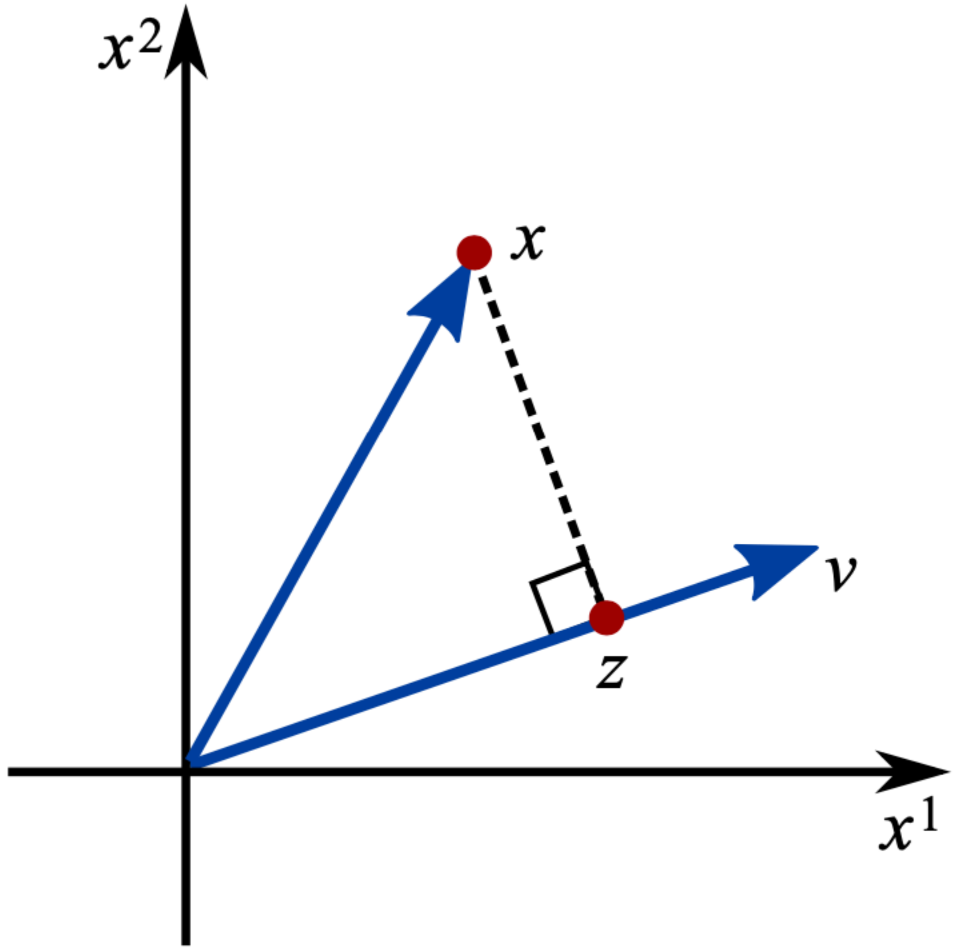
PCA is an eigen analysis of the covariance matrix:

$$\Sigma = V \Lambda V^T$$

- Eigenvectors: $v_k = V_{\bullet k}$ are **principal components**
- Eigenvalues: λ_k are the **variance** of the data in the v_k direction

Maximizing Variance of Projected Data

Fact: PCA finds dimensions that maximize variance

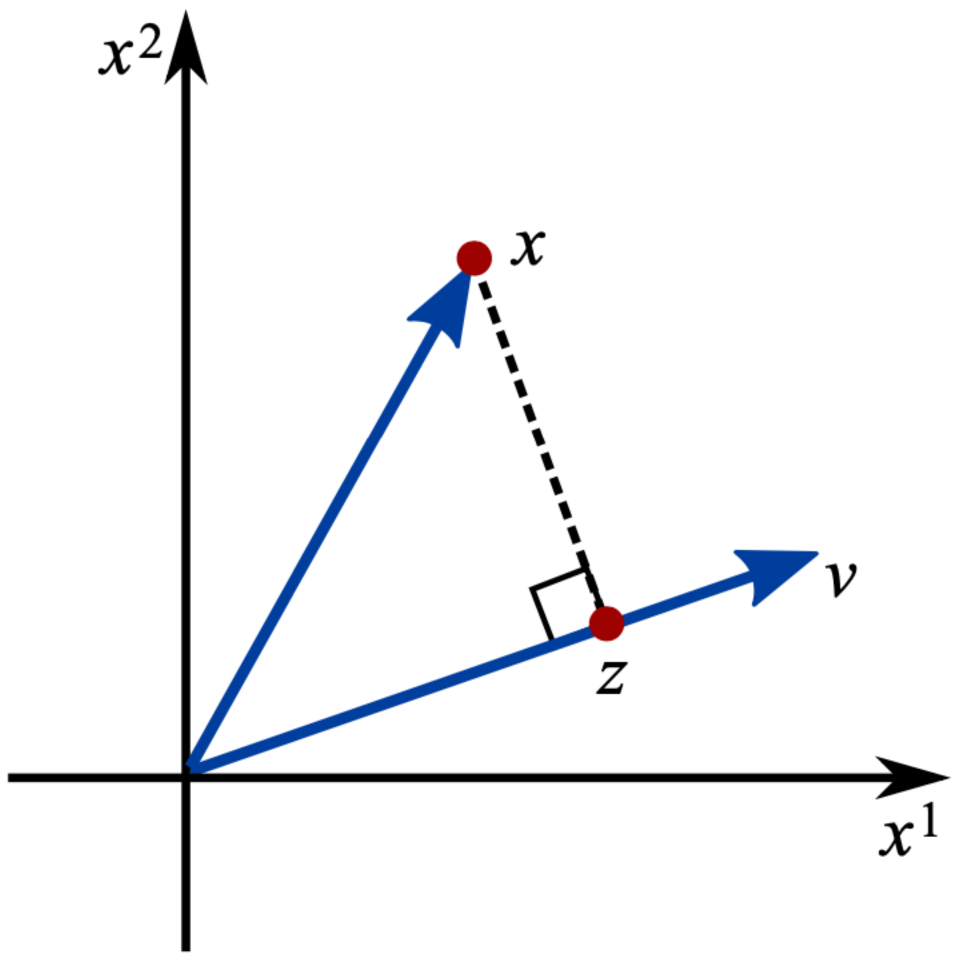


Given direction $v \in \mathbb{R}^d$, with $\|v\| = 1$,

project data point $x \in \mathbb{R}^d$ onto v:

$$z = \langle v, x \rangle$$

Maximizing Variance of Projected Data



Given mean-centered data, x_i,

first principal component, v_1 maximizes variance:

$$v_1 = \arg\max_{\|v\|=1} \sum_{i=1}^{n} \langle v, x_i \rangle^2$$

Dimensionality Reduction

Goal: Find a k-dimensional subspace, v_k, that best fits our data

Least-squares fit:

$$\arg\min_{V_k} \sum_{i=1}^{n} \text{distance}(V_k, x_i)^2$$

Solution: Use first k principal components:

$$V_k = \text{span}(v_1, v_2, \cdots, v_k)$$

PCA Algorithm Summary

Input: Data matrix $X : n \times d$

1. Compute centered data $\tilde{X}$
2. Compute covariance matrix:

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

3. Eigen analysis of covariance:

$$\Sigma = V\Lambda V^T$$

- *numpy.linalg.eigh* computes an eigen analysis of a symmetric matrix
- *numpy.linalg.SVD* for singular value decomposition.

PCA Algorithm Summary

Input: Data matrix $X : n \times d$

1. Compute centered data $\tilde{X}$
2. Compute covariance matrix:

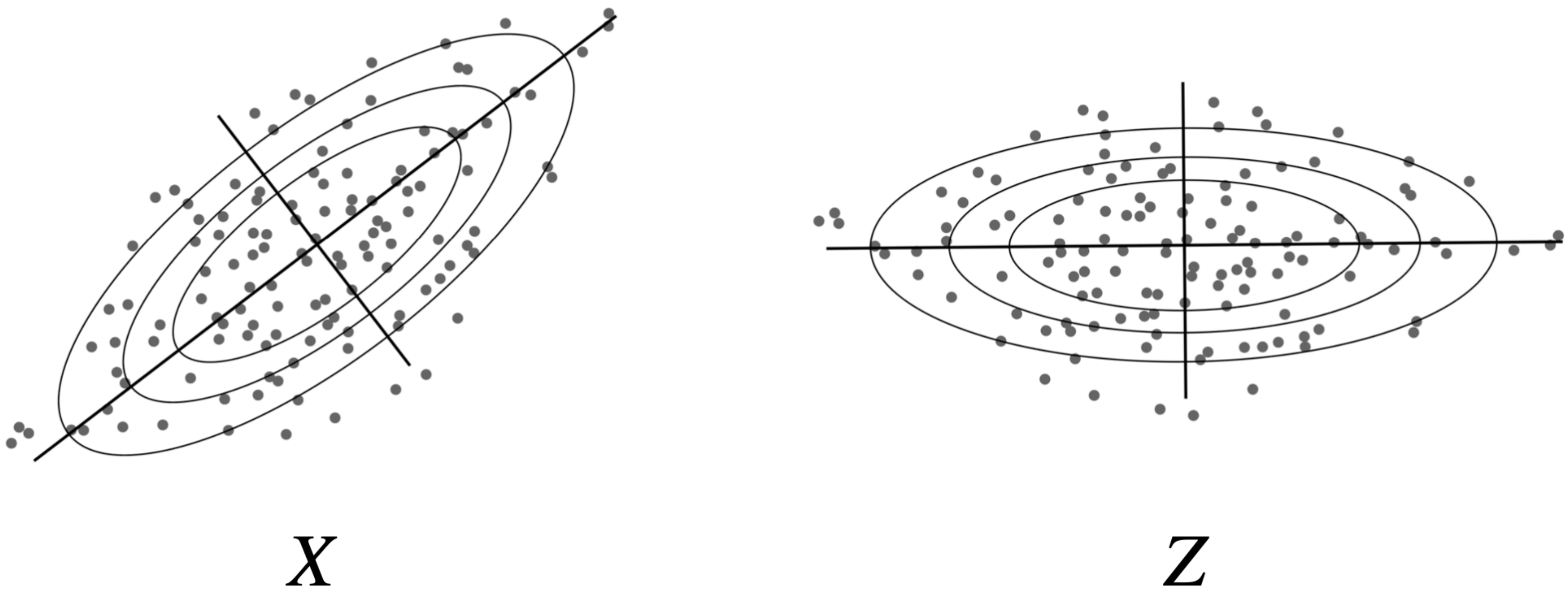
$$\Sigma = \frac{1}{n-1} \sum_{i=1}^{n} \tilde{X}^T \tilde{X}$$

3. Eigen analysis of covariance:

$$\Sigma = V \Lambda V^T$$

- Transpose trick: when n<<d. Compute $\tilde{X}\tilde{X}^T$ instead!

PC's as Rotation

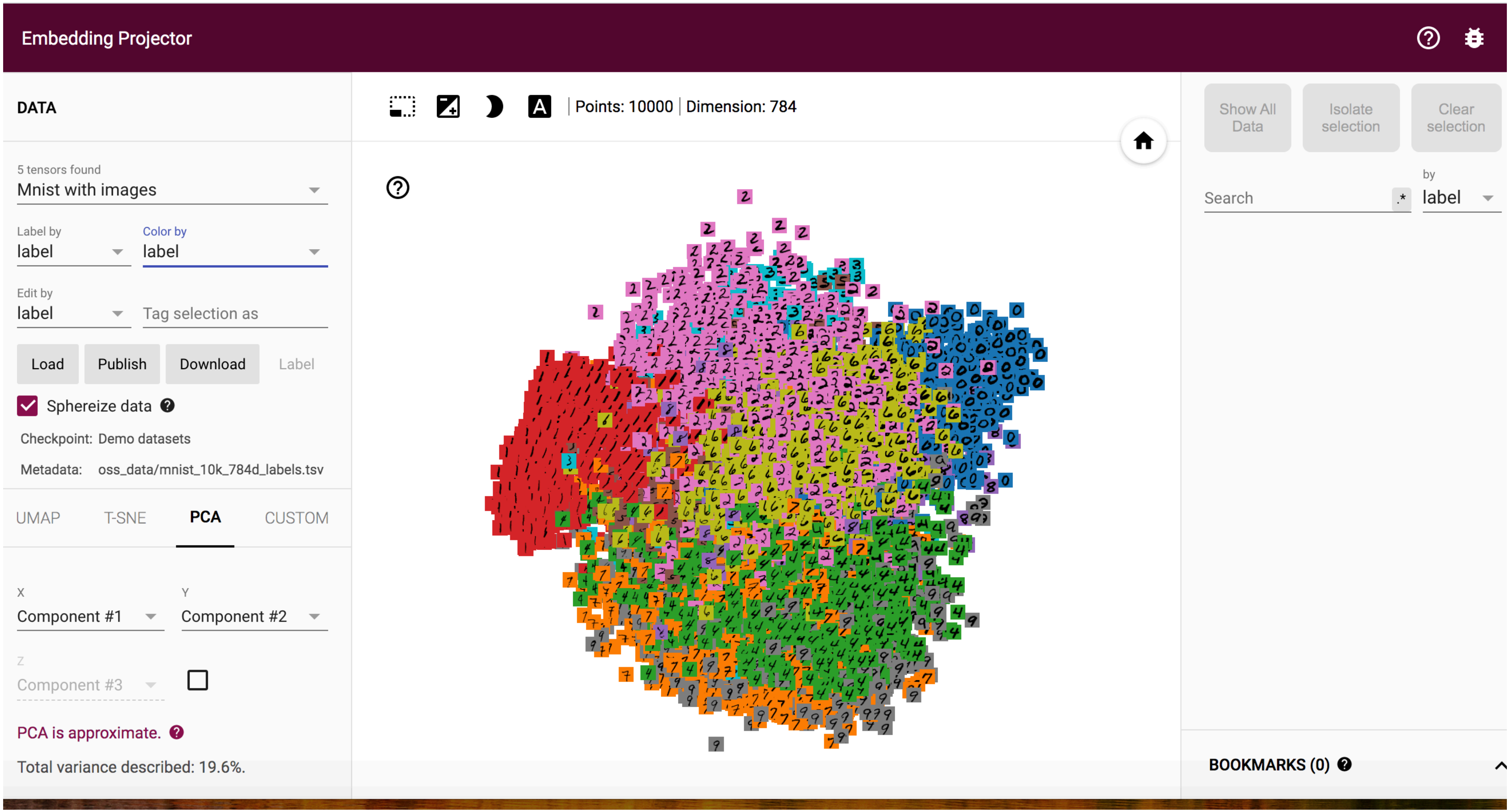


The principal components matrix, V, acts as a rotation:

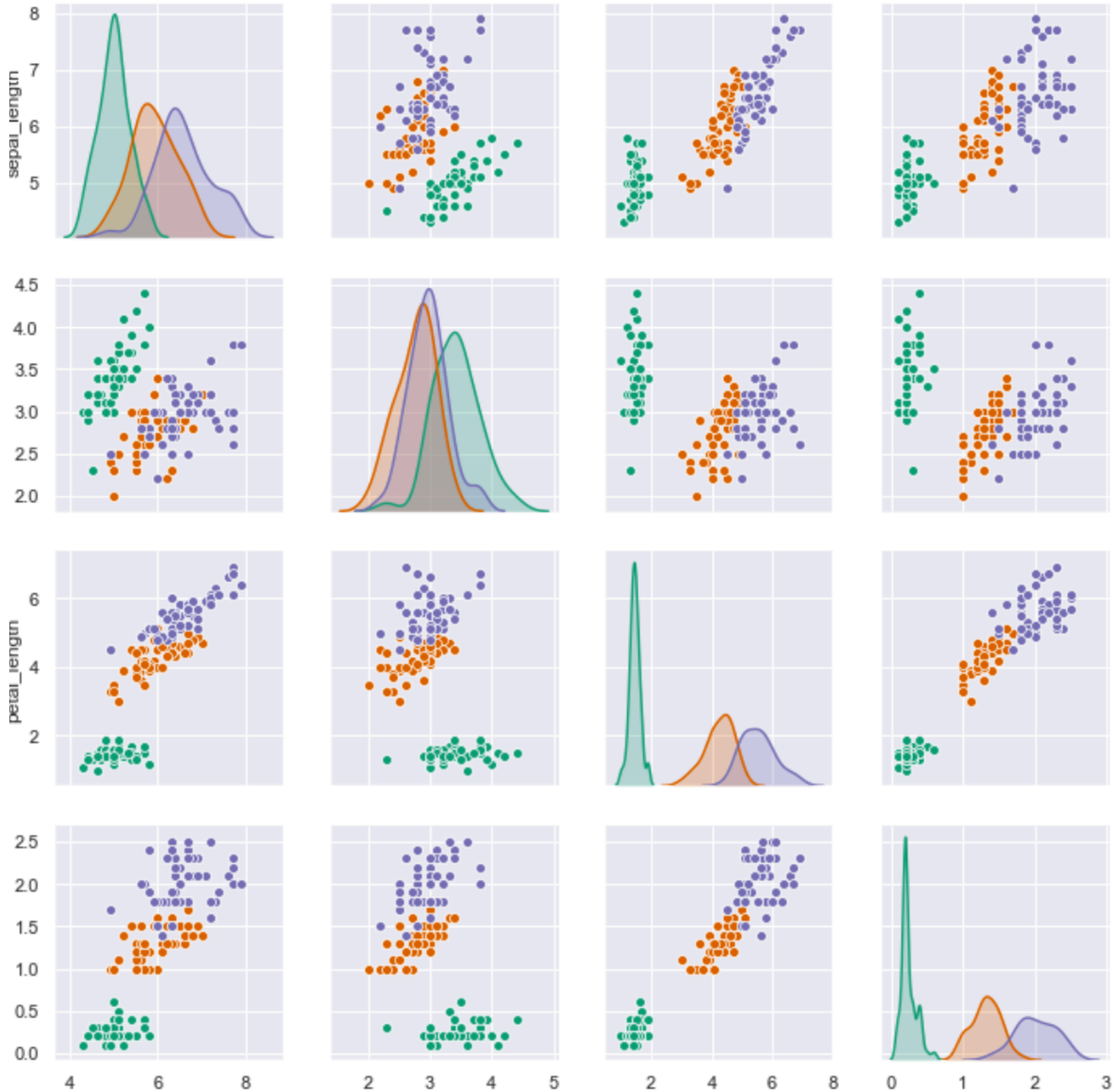
$$Z = XV$$

Columns of Z are new coordinates, called **loadings**.

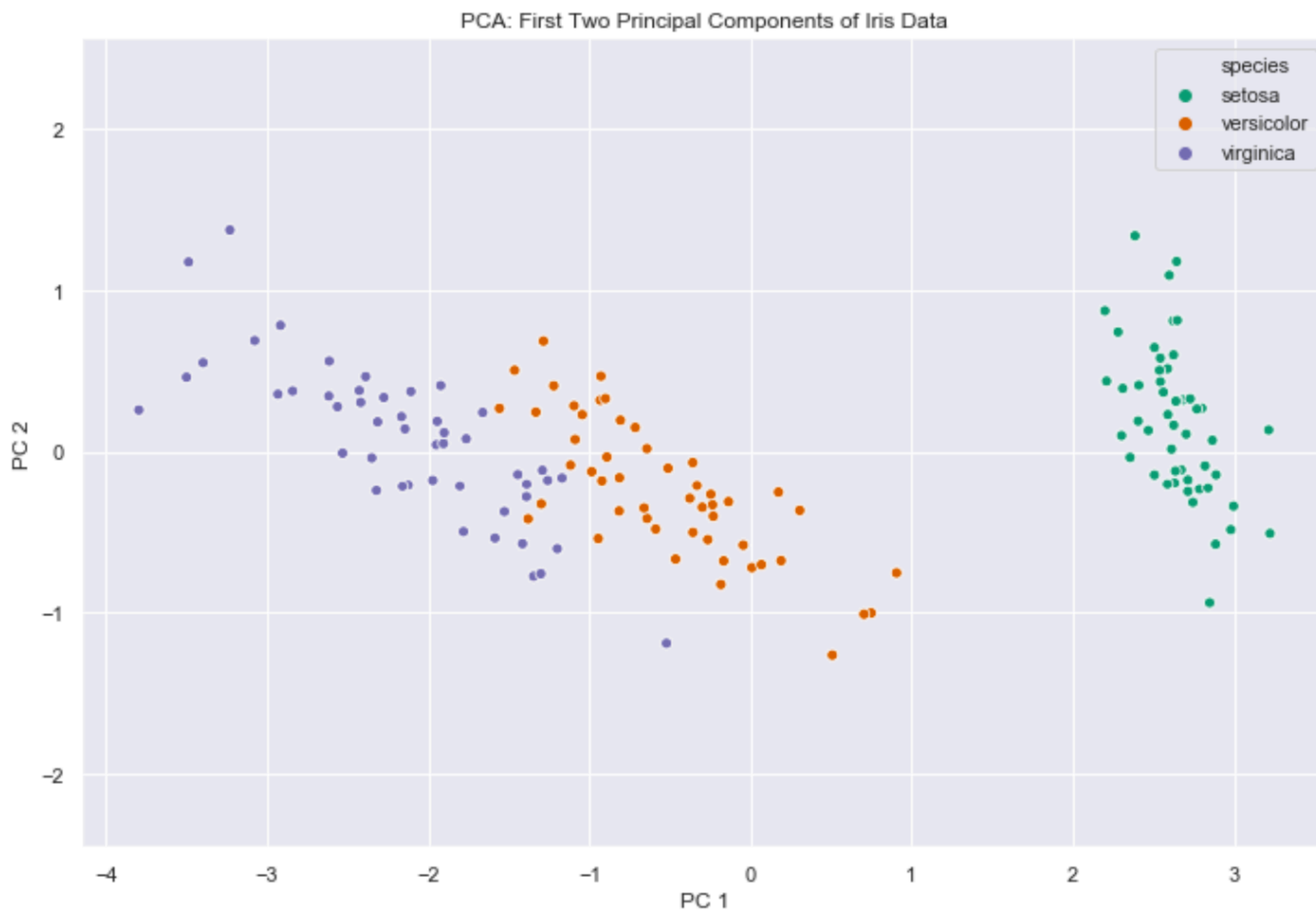
PCA to Project MNIST into 2D Space



Example: Iris Data



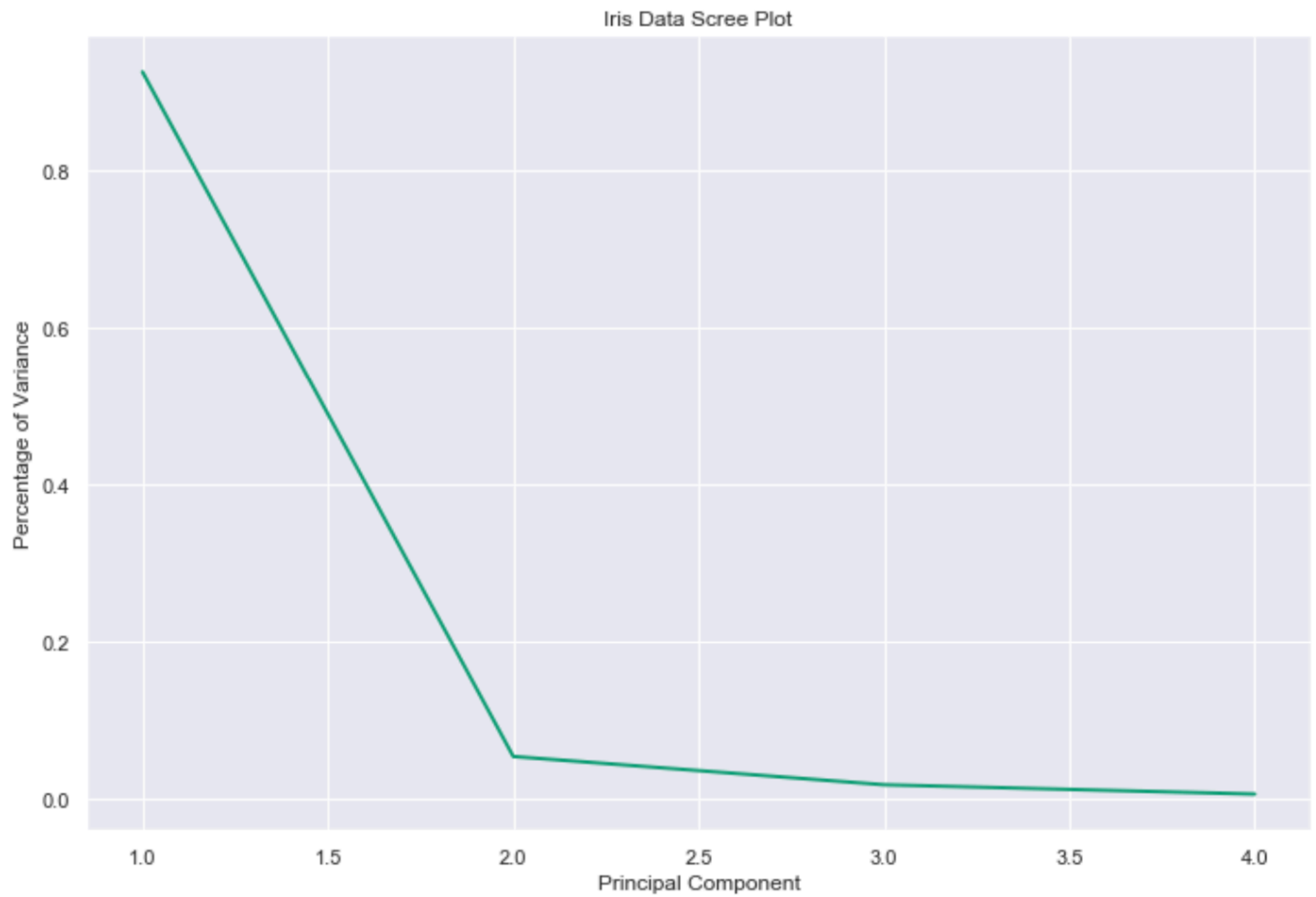
Example: Iris Data PCA



Eigenvectors: $$V = \begin{pmatrix} -0.361387, & 0.656589, & 0.582030, & 0.315487 \\ 0.084523, & 0.730161, & -0.597911, & -0.319723 \\ -0.856671, & -0.173373, & -0.076236, & -0.479839 \\ -0.358289, & -0.075481, & -0.545831, & 0.753657 \end{pmatrix}$$

Eigenvalues: $$\lambda = (\boxed{4.22824171, 0.24267075,}\ 0.0782095, 0.02383509)$$

Scree Plot: Eigenvalues (Variance)



Horizontal axis: which principal component (index k)

Vertical axis: proportion of variance: $\dfrac{\lambda_k}{\sum_{j=1}^{d} \lambda_j}$

Application as Face Recognition

How to Recognize An Unknown Face?

Training dataset

New data

Challenge?

- Curse of dimensionality

- Images are in high-dimensional space

$\boxed{\text{Data dimensionality reduction}}$

PCA

Reduce the dimensionality of the data while preserving as much information as possible in the original dataset.

High-dimensional space $\Rightarrow$ Low-dimensional space

Face Recognition: Training

- Train the recognizer
- Select the K most important Eigen faces $V^{D \times K}, K << D$
- Project each face into estimated subspace and store the associated weight vectors

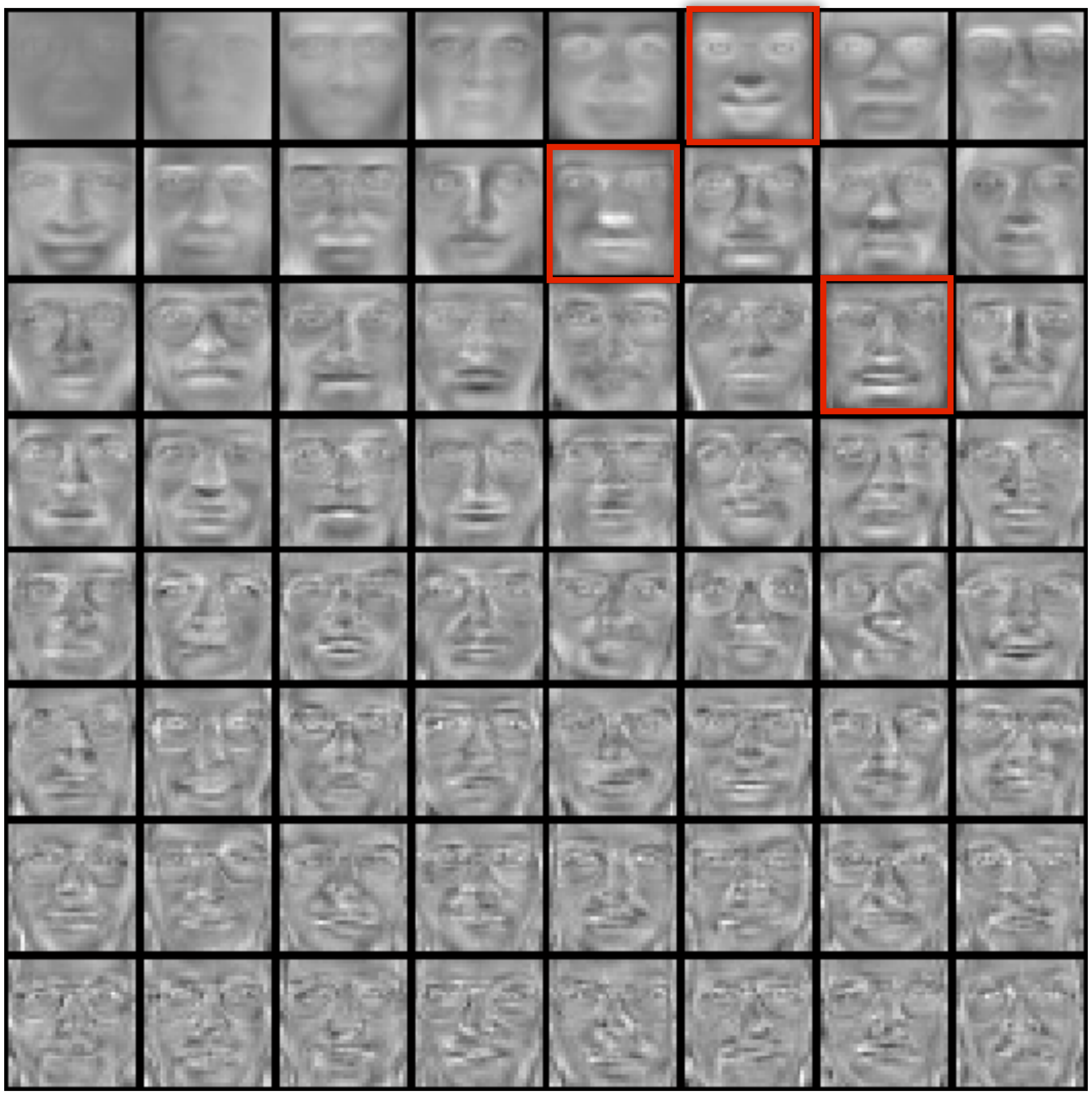
$$X^{N \times D} \times V^{D \times K} = \hat{X}^{N \times K}$$

Data matrix projected onto a lower dimensional space K

Eigen Faces

Mean Image

Examples of eigen faces $V^{D \times K}$

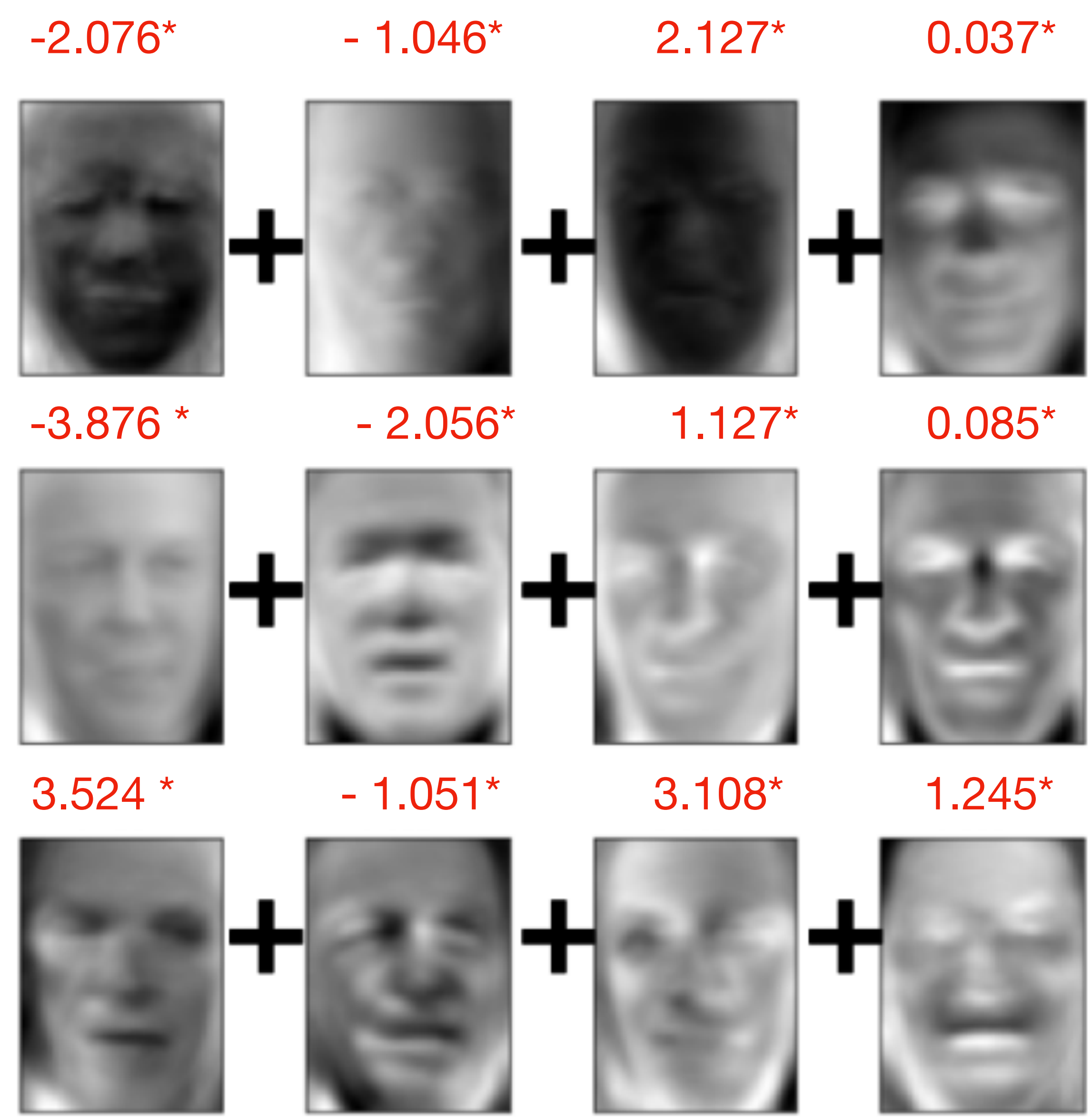


Each eigen face is the column vector of $V^{D \times K}$

Represent Training Images By Eigen Faces

Training image

=



$$X^{N\times D} \times V^{D\times K} = \hat{X}^{N\times K}$$

$$X = \hat{X} \times V^{T}$$

Eigen faces

Weighting/ loadings

Face Recognition: Testing

- Generate a face vector by subtracting mean out
- Project the unknown face into the K-dimensional subspace and compute the associated weighting/ loading vector
- Compute the distance between input weight vector and all the weight vectors in the training dataset

$\text{distance} < \epsilon$

Yes! She is Shelly!

No

Recognition Accuracy