

# **PCA**

**Reading list for this week:**  
**Bishop Chap 12.1; optional add ons 12.2, 12.3 and Hastie Chap 14.5**

**Slides in this presentation courtesy of Matt Gormley and MIT 10-301**

<https://forms.gle/Dbhp6UoLhpw3ToaJ7>

We need to fix this class!

# High Dimension Data

Examples of high dimensional data:

- Brain Imaging Data (100s of MBs per scan)

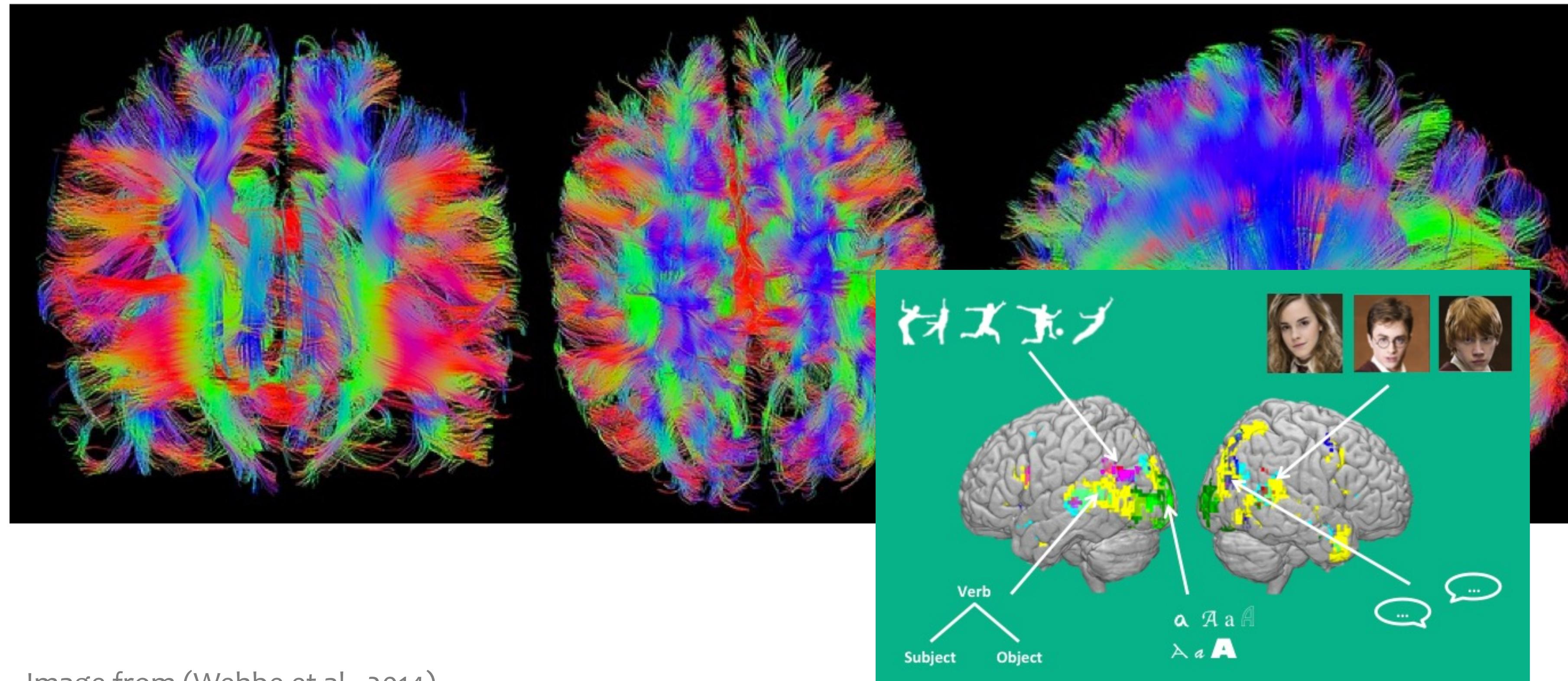


Image from (Wehbe et al., 2014)

Image from <https://pixabay.com/en/brain-mrt-magnetic-resonance-imaging-1728449/>

# High Dimension Data

Examples of high dimensional data:

- High resolution images (millions of pixels)



# High Dimension Data

Examples of high dimensional data:

– Multilingual News Stories

Vocabulary of millions of words!



# Learning Representations

## Dimensionality Reduction Algorithms:

Powerful (often unsupervised) learning techniques for extracting hidden (potentially lower dimensional) structure from high dimensional datasets.

## Examples:

PCA, Kernel PCA, ICA, CCA, t-SNE, Autoencoders, Matrix Factorization

## Useful for:

- Visualization
- More efficient use of resources (e.g., time, memory, communication)
- Statistical: fewer dimensions → better generalization
- Noise removal (improving data quality)

# Dimensionality reduction by random projection

# Random Projection

Goal: project from M-dimensions down to K-dimensions

Data:

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \text{ where } \mathbf{x}^{(i)} \in \mathbb{R}^M$$

Algorithm:

1. Randomly sample matrix:  $\mathbf{V} \in \mathbb{R}^{K \times M}$

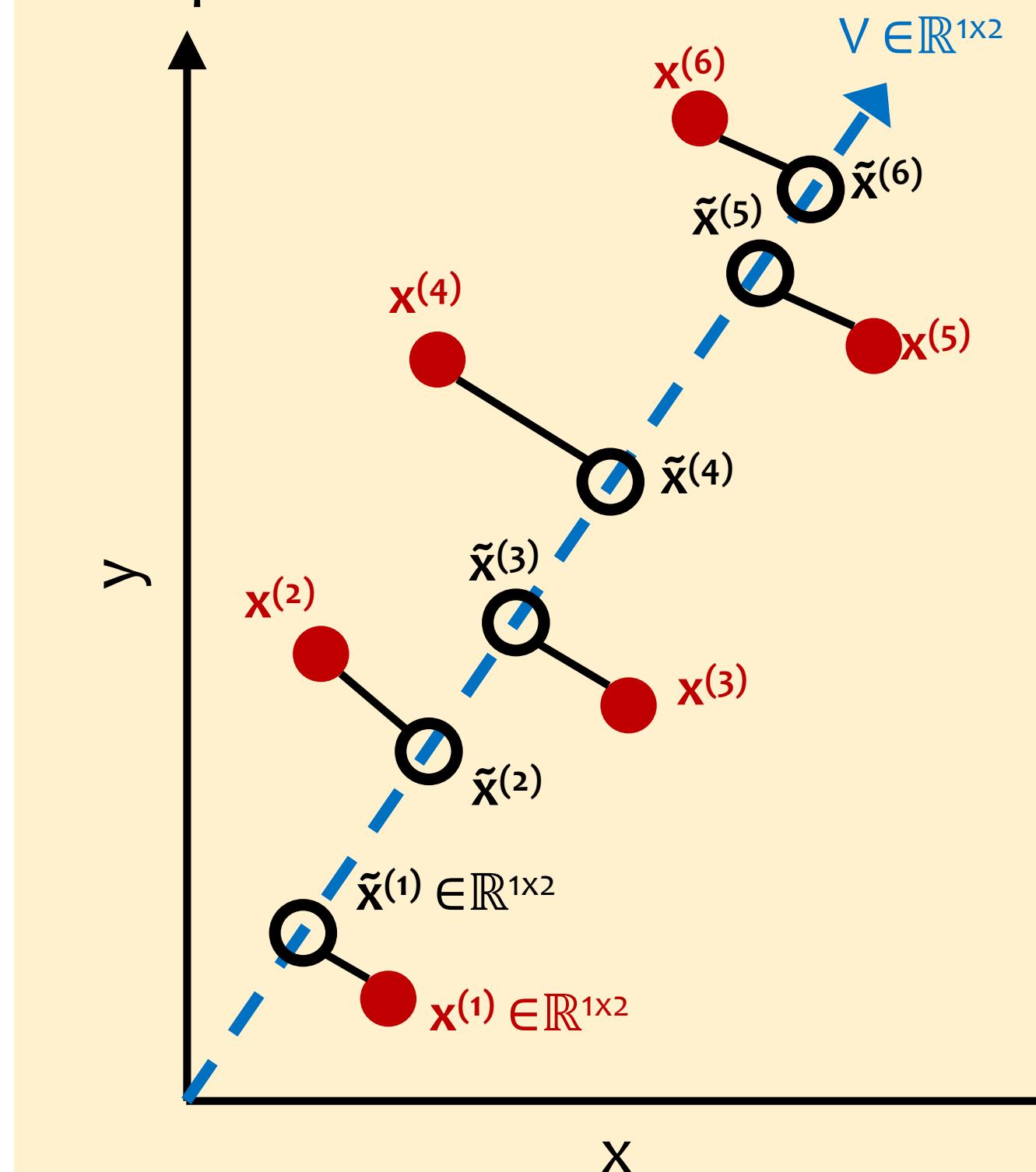
$$V_{km} \sim \text{Gaussian}(0, 1)$$

2. Project down:  $\underbrace{\mathbf{u}^{(i)}}_{K \times 1} = \underbrace{\mathbf{V}}_{K \times M} \underbrace{\mathbf{x}^{(i)}}_{M \times 1}$

3. Project up:  $\underbrace{\tilde{\mathbf{x}}^{(i)}}_{M \times 1} = \underbrace{\mathbf{V}^T}_{M \times K} \underbrace{\mathbf{u}^{(i)}}_{K \times 1} = \mathbf{V}^T(\mathbf{V}\mathbf{x}^{(i)})$

Example: 2D to 1D

2D input data



1D projection onto the real line

$$u^{(1)} \in \mathbb{R} \quad u^{(2)} \quad u^{(3)} \quad u^{(4)} \quad u^{(5)} \quad u^{(6)}$$



# Random Projection

Goal: project from M-dimensions down to K-dimensions

Data:

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N \text{ where } \mathbf{x}^{(i)} \in \mathbb{R}^M$$

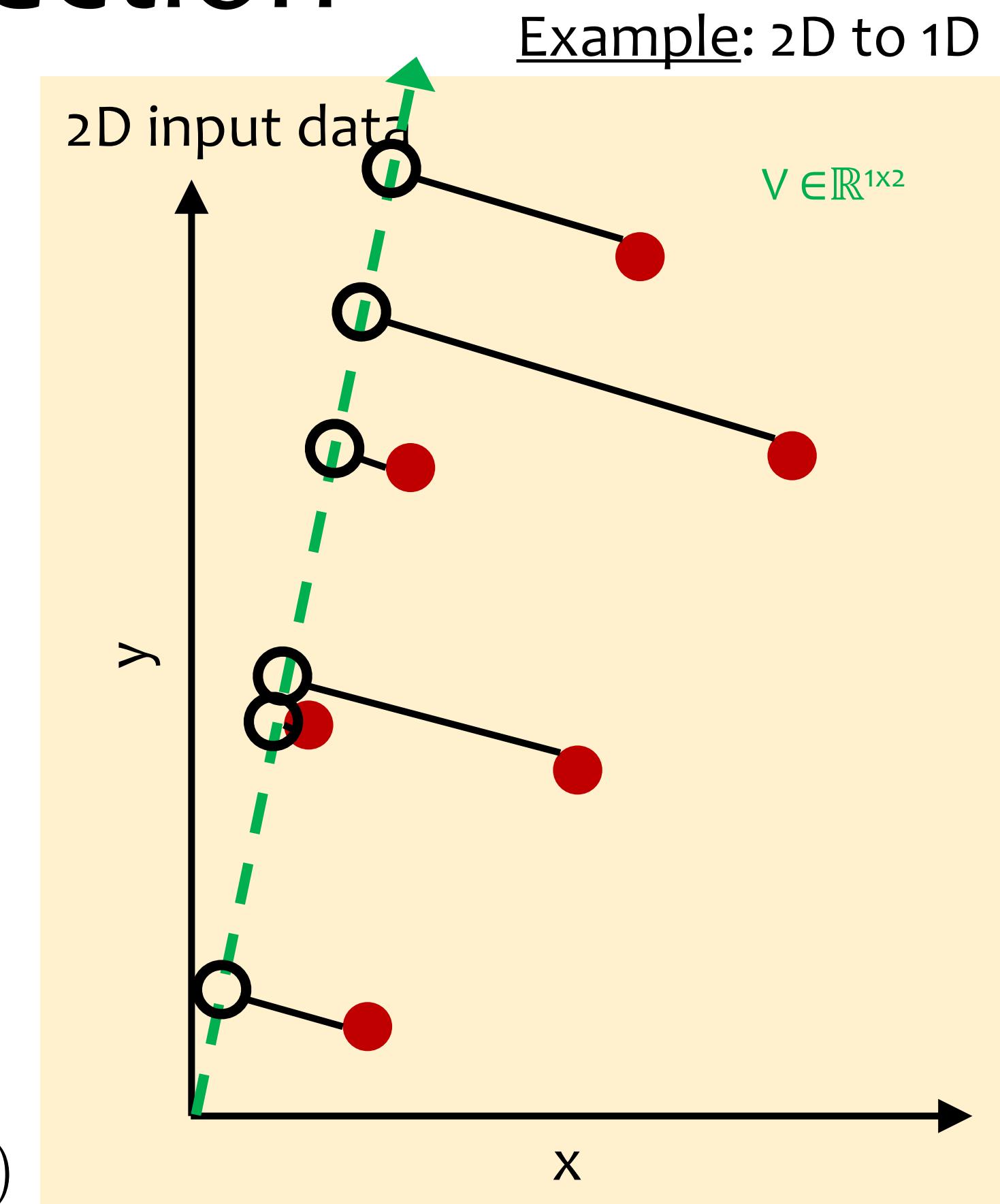
Algorithm:

1. Randomly sample matrix:  $\mathbf{V} \in \mathbb{R}^{K \times M}$

$$V_{km} \sim \text{Gaussian}(0, 1)$$

2. Project down:  $\underbrace{\mathbf{u}^{(i)}}_{K \times 1} = \underbrace{\mathbf{V}}_{K \times M} \underbrace{\mathbf{x}^{(i)}}_{M \times 1}$

3. Project up:  $\underbrace{\mathbf{x}^{(i)}}_{M \times 1} = \underbrace{\mathbf{V}^T}_{M \times K} \underbrace{\mathbf{u}^{(i)}}_{K \times 1} = \mathbf{V}^T(\mathbf{V}\mathbf{x}^{(i)})$



**Problem:** a random projection might give us a poor low dimensional representation of the data

# Johnson-Lindenstrauss Lemma

**Q:** But how could we ever hope to preserve any useful information by randomly projecting into a low-dimensional space?

**A:** Even random projection enjoys some surprisingly impressive properties. In fact, a standard of the J-L lemma starts by assuming we have a random linear projection obtained by sampling each matrix entry from a  $\text{Gaussian}(0,1)$ .



## *An Elementary Proof of a Theorem of Johnson and Lindenstrauss*

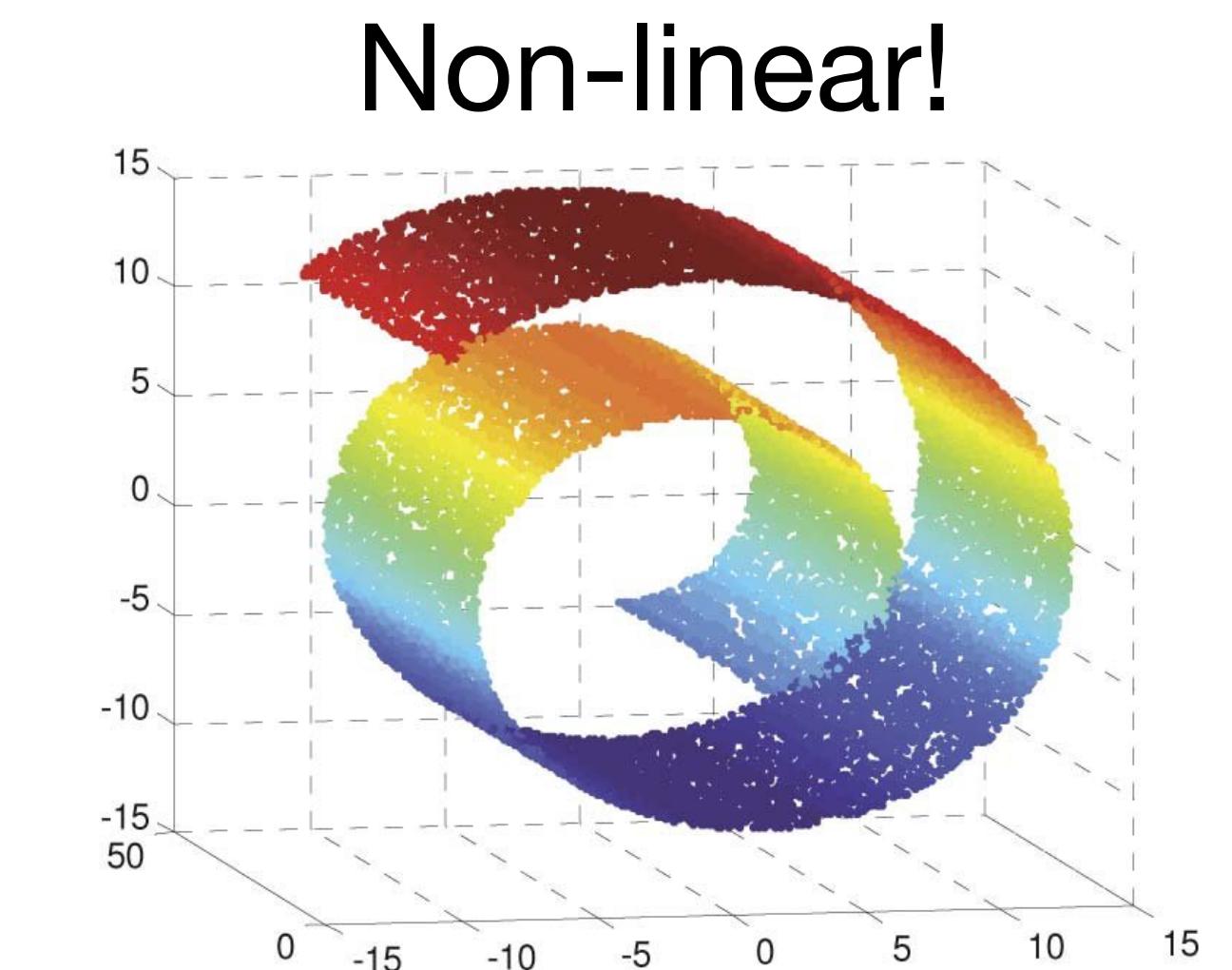
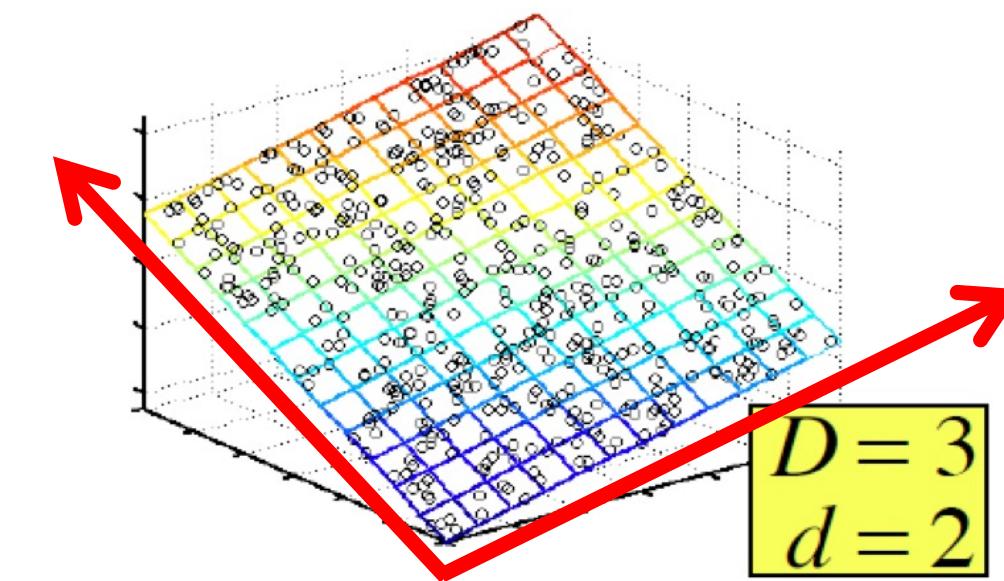
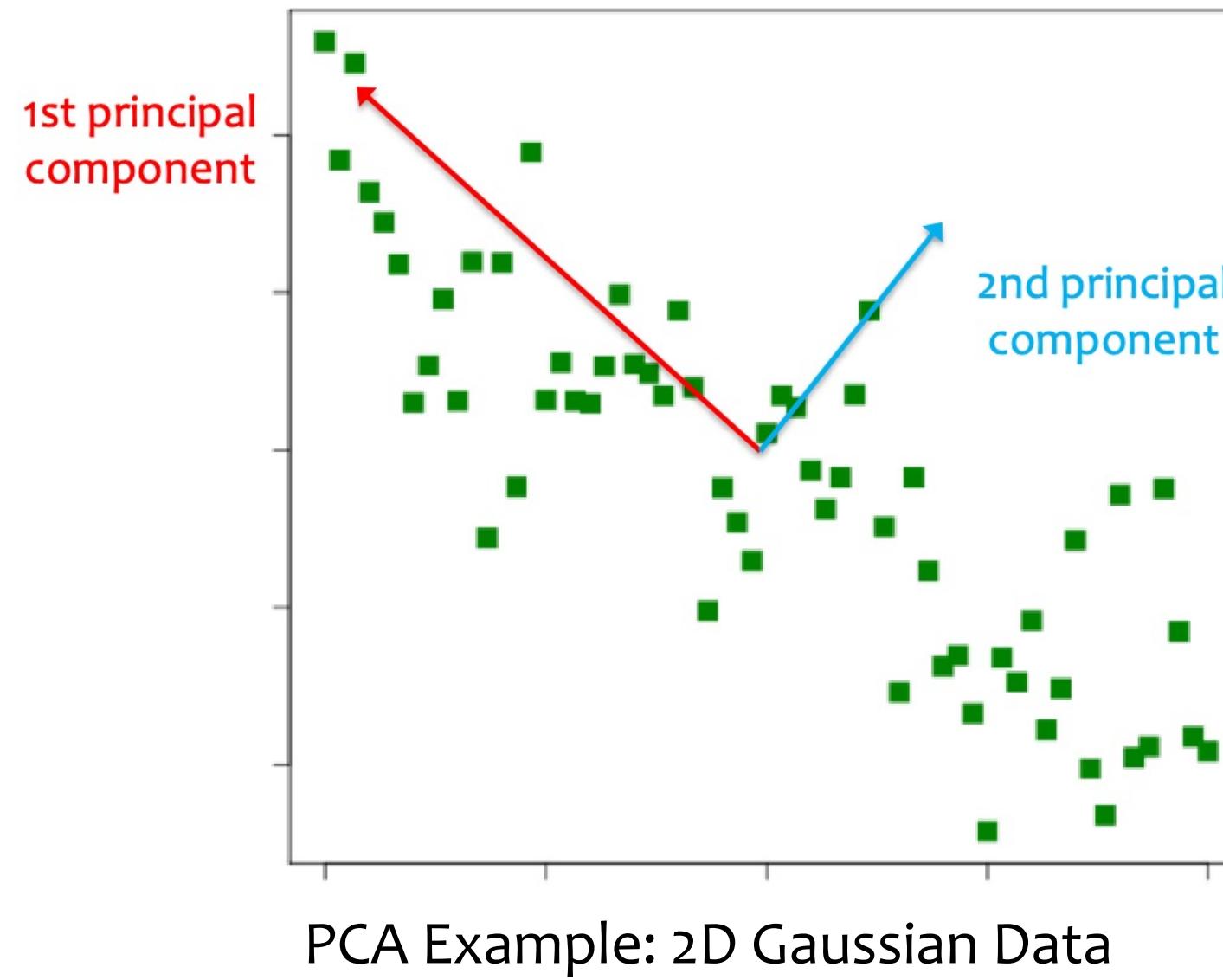
Sanjoy Dasgupta,<sup>1</sup> Anupam Gupta<sup>2</sup>

**ABSTRACT:** A result of Johnson and Lindenstrauss [13] shows that a set of  $n$  points in high dimensional Euclidean space can be mapped into an  $O(\log n/\epsilon^2)$ -dimensional Euclidean space such that the distance between any two points changes by only a factor of  $(1 \pm \epsilon)$ . In this note, we prove this theorem using elementary probabilistic techniques. © 2003 Wiley Periodicals, Inc. Random Struct. Alg., 22: 60–65, 2002

# Principle Components Analysis

# Principal Component Analysis (PCA)

- **Assumption:** the data lies on a low K-dimensional linear subspace
- **Goal:** identify the axes of that subspace, and project each point onto hyperplane
- **Algorithm:** find the K eigenvectors with largest eigenvalue using classic matrix decomposition tools



# Data for PCA

$$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^N$$

$$\mathbf{x}^{(i)} \in \mathbb{R}^M$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(N)})^T \end{bmatrix}$$

We assume the data is **centered**,  
i.e. the **sample mean** is zero

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}^{(i)} = \mathbf{0}$$

**Q:** What if  
your data is  
**not centered?**

**A:** Subtract off the sample mean

$$\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \hat{\mu}, \forall i$$

# Sample Covariance Matrix

The **sample covariance matrix**  $\Sigma \in \mathbb{R}^{M \times M}$  is given by:

$$\Sigma_{jk} = \frac{1}{N} \sum_{i=1}^N (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

Since the data matrix is centered, we rewrite as:

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

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(N)})^T \end{bmatrix}$$

# Principal Component Analysis (PCA)

## Linear Projection:

Given  $K \times M$  matrix  $\mathbf{V}$ , and  $M \times 1$  vector  $\mathbf{x}^{(i)}$  we obtain the  $K \times 1$  projection  $\mathbf{u}^{(i)}$  by:

$$\mathbf{u}^{(i)} = \mathbf{V} \mathbf{x}^{(i)}$$

$$\mathbf{V} = \begin{bmatrix} \vdash \mathbf{v}_1^T \vdash \\ \vdash \mathbf{v}_2^T \vdash \\ \vdash \vdash \vdash \\ \vdash \mathbf{v}_k^T \vdash \end{bmatrix}$$

## Definition of PCA:

PCA repeatedly chooses a next vector  $\mathbf{v}_j$  that **minimizes the reconstruction error** s.t.  $\mathbf{v}_j$  is orthogonal to  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{j-1}$ .

Vector  $\mathbf{v}_j$  is called the **jth principal component**.

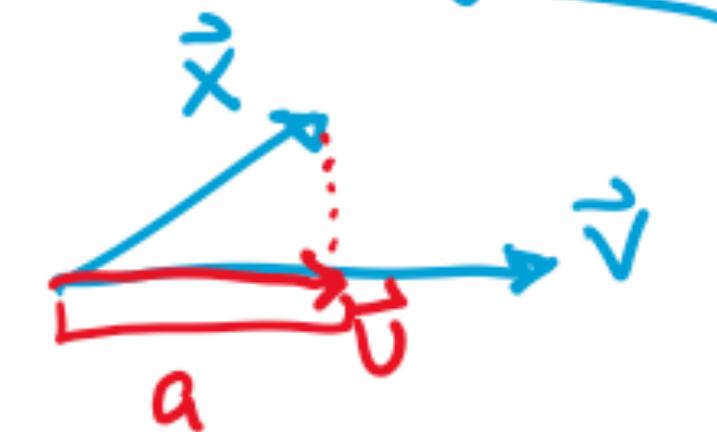
Notice: Two vectors  $\mathbf{a}$  and  $\mathbf{b}$  are **orthogonal** if  $\mathbf{a}^T \mathbf{b} = 0$ .

→ the  $K$ -dimensions in PCA are uncorrelated

# Vector projection

# Vector Projection

Recall: Projection



length of projection of  $\vec{x}$  onto  $\vec{v}$

$$a = \frac{\vec{v}^T \vec{x}}{\|\vec{v}\|_2} \quad \text{if } \|\vec{v}\|_2 = 1$$
$$\frac{1}{\|\vec{v}\|_2} \quad \text{otherwise}$$

projection of  $\vec{x}$  onto  $\vec{v}$

$$\vec{u} = a \vec{v} = \frac{(\vec{v}^T \vec{x}) \vec{v}}{\|\vec{v}\|_2^2} \quad \text{if } \|\vec{v}\|_2 = 1$$
$$\frac{0}{\|\vec{v}\|_2^2} \quad \text{otherwise}$$

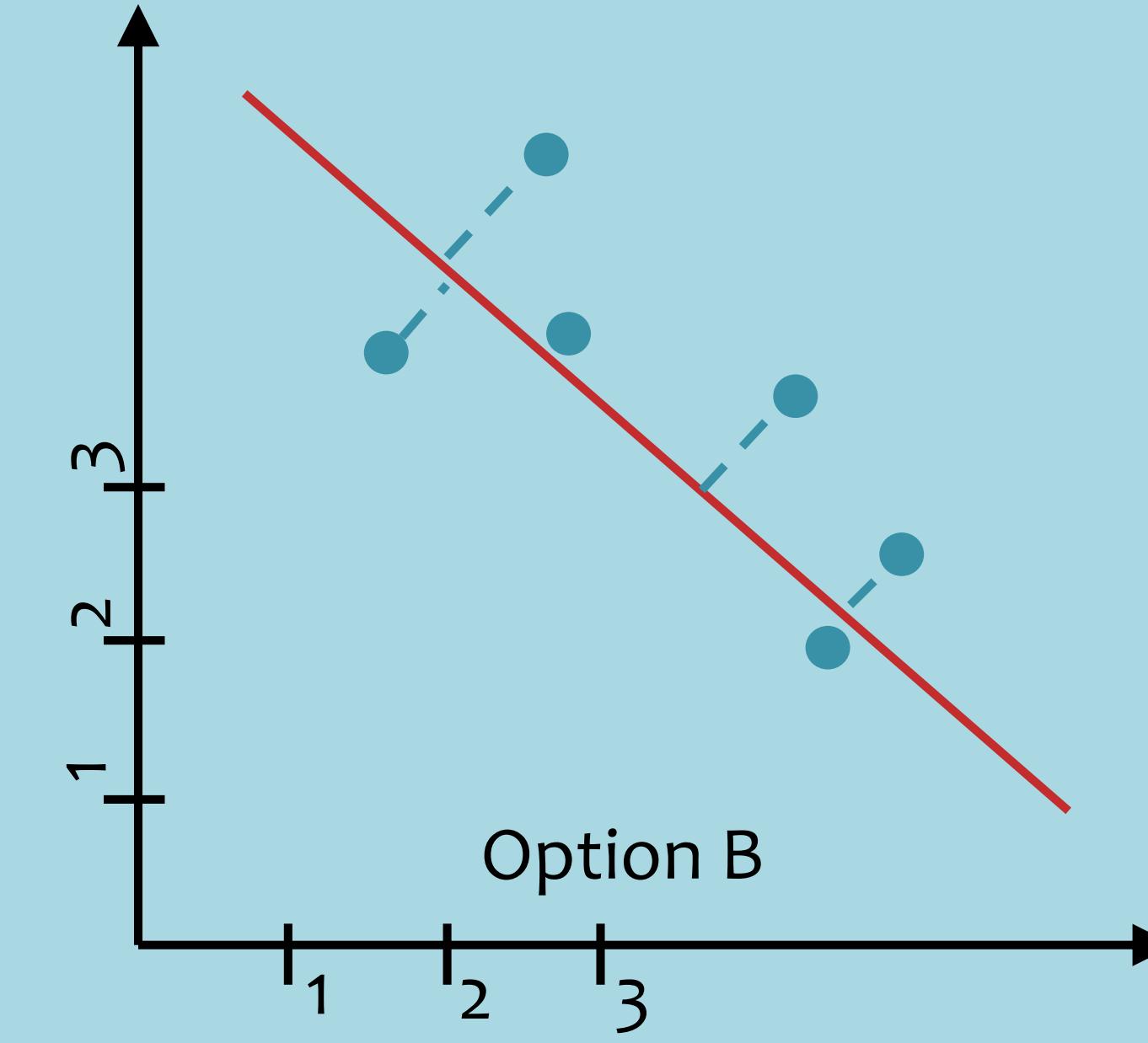
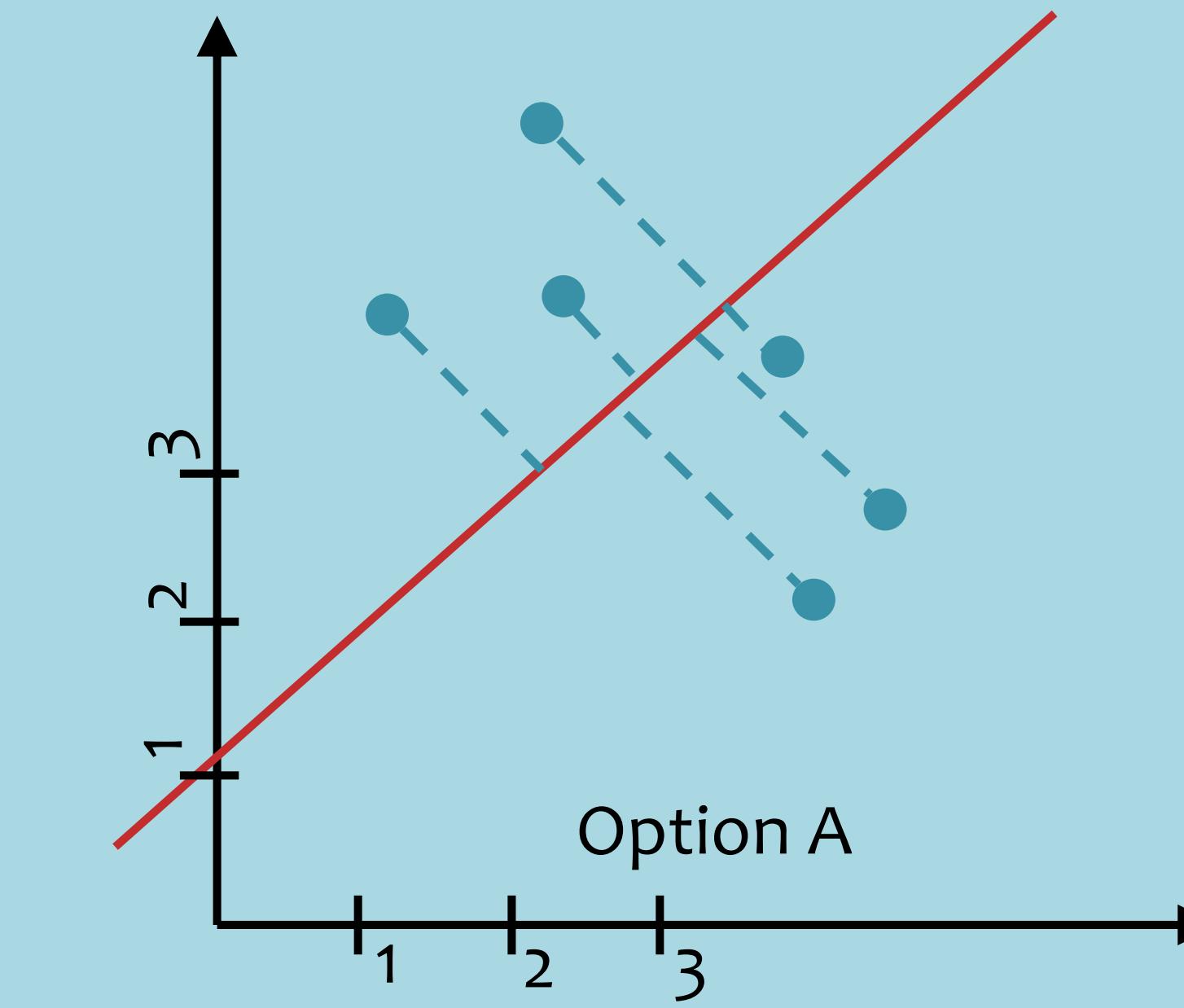
# Projection Example

## Question:

Below are two plots of the same dataset D. Consider the two projections shown.

1. Which maximizes the variance?
2. Which minimizes the reconstruction error?

## Answer:



# PCA Objective Functions

What is the first principal component  $\mathbf{v}_1$  chosen by PCA?

Option 1: The vector that *minimizes the reconstruction error*

$$\mathbf{v}_1 = \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}^{(i)} - (\mathbf{v}^T \mathbf{x}^{(i)}) \mathbf{v}\|^2$$

Option 2: The vector that *maximizes the variance*

$$\mathbf{v}_1 = \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N (\mathbf{v}^T \mathbf{x}^{(i)})^2$$

Subject to constraints on magnitude of  $\mathbf{v}$  because otherwise a dumb solution exists!

# Equivalence of Maximizing Variance and Minimizing Reconstruction Error

## PCA

**Claim:** Minimizing the reconstruction error is equivalent to maximizing the variance.

**Proof:** First, note that:

$$\|\mathbf{x}^{(i)} - (\mathbf{v}^T \mathbf{x}^{(i)})\mathbf{v}\|^2 = \|\mathbf{x}^{(i)}\|^2 - (\mathbf{v}^T \mathbf{x}^{(i)})^2 \quad (1)$$

since  $\mathbf{v}^T \mathbf{v} = \|\mathbf{v}\|^2 = 1$ .

Substituting into the minimization problem, and removing the extraneous terms, we obtain the maximization problem.

$$\mathbf{v}^* = \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}^{(i)} - (\mathbf{v}^T \mathbf{x}^{(i)})\mathbf{v}\|^2 \quad (2)$$

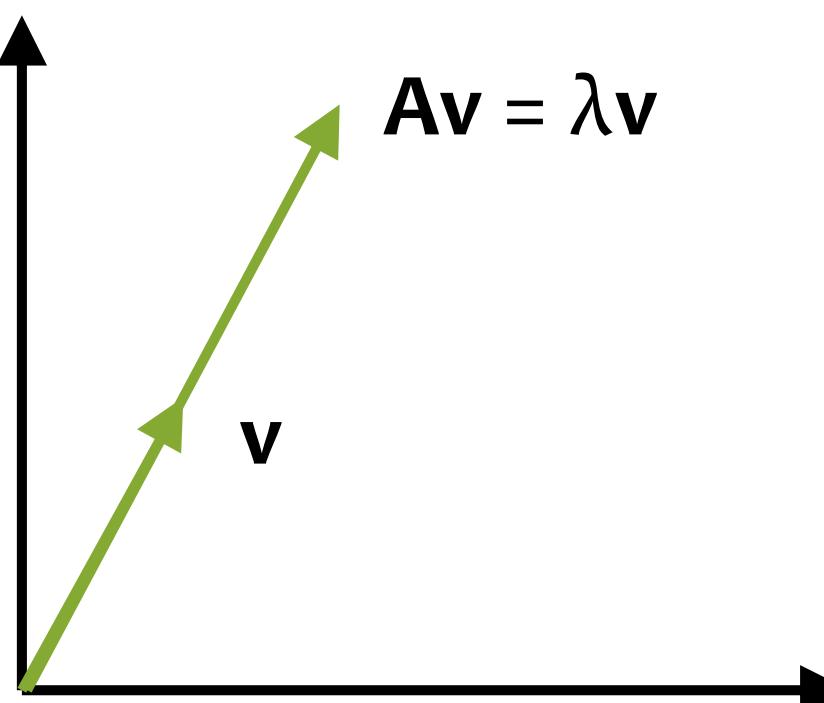
$$= \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}^{(i)}\|^2 - (\mathbf{v}^T \mathbf{x}^{(i)})^2 \quad (3)$$

$$= \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^N (\mathbf{v}^T \mathbf{x}^{(i)})^2 \quad (4)$$

# Background: Eigenvectors & Eigenvalues

For a square matrix  $\mathbf{A}$  ( $n \times n$  matrix), the vector  $\mathbf{v}$  ( $n \times 1$  matrix) is an **eigenvector** iff there exists **eigenvalue**  $\lambda$  (scalar) such that:

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



The linear transformation  $\mathbf{A}$  is only stretching vector  $\mathbf{v}$ .

That is,  $\lambda\mathbf{v}$  is a scalar multiple of  $\mathbf{v}$ .

# Background: Eigenvectors & Eigenvalues

Fact #1: The eigenvectors of a **symmetric matrix** are **orthogonal** to each other.

Fact #2: The **covariance matrix  $\Sigma$**  is **symmetric**.

## Single constraint [ edit ]

For the case of only one constraint and only two choice variables (as exemplified in Figure 1), consider the [optimization problem](#)

$$\underset{x,y}{\text{maximize}} \quad f(x,y)$$

$$\text{subject to} \quad g(x,y) = 0.$$

(Sometimes an additive constant is shown separately rather than being included in  $g$ , in which case the constraint is written  $g(x,y) = c$ , as in Figure 1.) We assume that both  $f$  and  $g$  have continuous first [partial derivatives](#). We introduce a new variable ( $\lambda$ ) called a **Lagrange multiplier** (or **Lagrange undetermined multiplier**) and study the **Lagrange function** (or **Lagrangian** or **Lagrangian expression**) defined by

$$\mathcal{L}(x,y,\lambda) = f(x,y) + \lambda \cdot g(x,y),$$

where the  $\lambda$  term may be either added or subtracted. If  $f(x_0, y_0)$  is a maximum of  $f(x, y)$  for the original constrained problem and  $\nabla g(x_0, y_0) \neq 0$ , then there exists  $\lambda_0$  such that

$(x_0, y_0, \lambda_0)$  is a [stationary point](#) for the Lagrange function (stationary points are those points where the first partial derivatives of  $\mathcal{L}$  are zero). The assumption  $\nabla g \neq 0$  is called constraint qualification. However, not all stationary points yield a solution of the original problem, as the method of Lagrange multipliers yields only a [necessary condition](#) for optimality in constrained problems.[\[9\]](#)[\[10\]](#)[\[11\]](#)[\[12\]](#)[\[13\]](#) Sufficient conditions for a minimum or maximum [also exist](#), but if a particular [candidate solution](#) satisfies the sufficient conditions, it is only guaranteed that that solution is the best one *locally* – that is, it is better than any permissible nearby points. The *global* optimum can be found by comparing the values of the original objective function at the points satisfying the necessary and locally sufficient conditions.

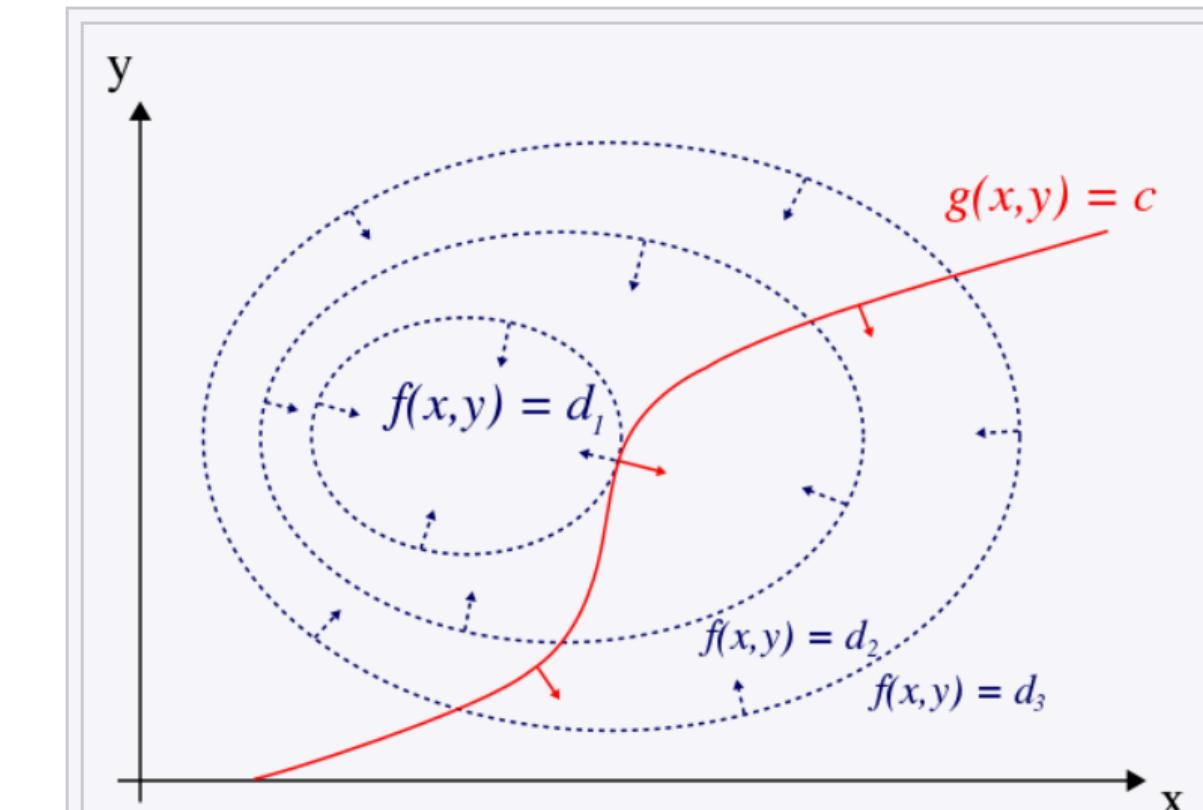


Figure 1: The red curve shows the constraint  $g(x,y) = c$ . The blue curves are contours of  $f(x,y)$ .

The point where the red constraint tangentially touches a blue contour is the maximum of  $f(x,y)$  along the constraint, since  $d_1 > d_2$ .

## The First Principal Component

# PCA

**Claim:** The vector that maximizes the variances is the eigenvector of  $\Sigma$  with largest eigenvalue.

**Proof Sketch:** To find the first principal component, we wish to solve the following constrained optimization problem (variance minimization).

$$\mathbf{v}_1 = \underset{\mathbf{v}: \|\mathbf{v}\|^2=1}{\operatorname{argmax}} \mathbf{v}^T \Sigma \mathbf{v} \quad (1)$$

So we turn to the method of Lagrange multipliers. The Lagrangian is:

$$\mathcal{L}(\mathbf{v}, \lambda) = \mathbf{v}^T \Sigma \mathbf{v} - \lambda(\mathbf{v}^T \mathbf{v} - 1) \quad (2)$$

Taking the derivative of the Lagrangian and setting to zero gives:

$$\frac{d}{d\mathbf{v}} (\mathbf{v}^T \Sigma \mathbf{v} - \lambda(\mathbf{v}^T \mathbf{v} - 1)) = 0 \quad (3)$$

$$\Sigma \mathbf{v} - \lambda \mathbf{v} = 0 \quad (4)$$

$$\Sigma \mathbf{v} = \lambda \mathbf{v} \quad (5)$$

**Recall:** For a square matrix  $\mathbf{A}$ , the vector  $\mathbf{v}$  is an **eigenvector** iff there exists **eigenvalue**  $\lambda$  such that:

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \quad (6)$$

Rewriting the objective of the maximization shows that not only will the optimal vector  $\mathbf{v}_1$  be an eigenvector, it will be one with maximal eigenvalue.

$$\mathbf{v}^T \Sigma \mathbf{v} = \mathbf{v}^T \lambda \mathbf{v} \quad (7)$$

$$= \lambda \mathbf{v}^T \mathbf{v} \quad (8)$$

$$= \lambda \|\mathbf{v}\|^2 \quad (9)$$

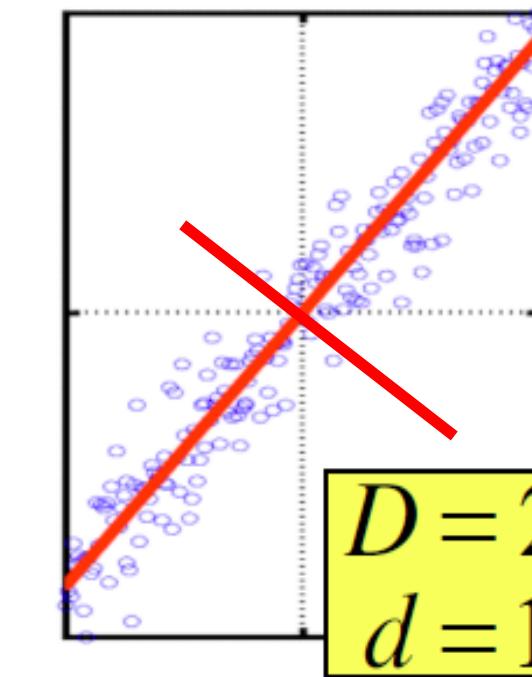
$$= \lambda \quad (10)$$

# Principal Component Analysis (PCA)

$(X X^T)v = \lambda v$ , so  $v$  (the first PC) is the eigenvector of sample correlation/covariance matrix  $X X^T$

Sample variance of projection  $v^T X X^T v = \lambda v^T v = \lambda$

Thus, the eigenvalue  $\lambda$  denotes the amount of variability captured along that dimension (aka amount of energy along that dimension).



Eigenvalues  $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \dots$

- The 1<sup>st</sup> PC  $v_1$  is the eigenvector of the sample covariance matrix  $X X^T$  associated with the largest eigenvalue
- The 2nd PC  $v_2$  is the eigenvector of the sample covariance matrix  $X X^T$  associated with the second largest eigenvalue
- And so on ...

# Algorithms for PCA

## Singular Value Decomposition (SVD)

---

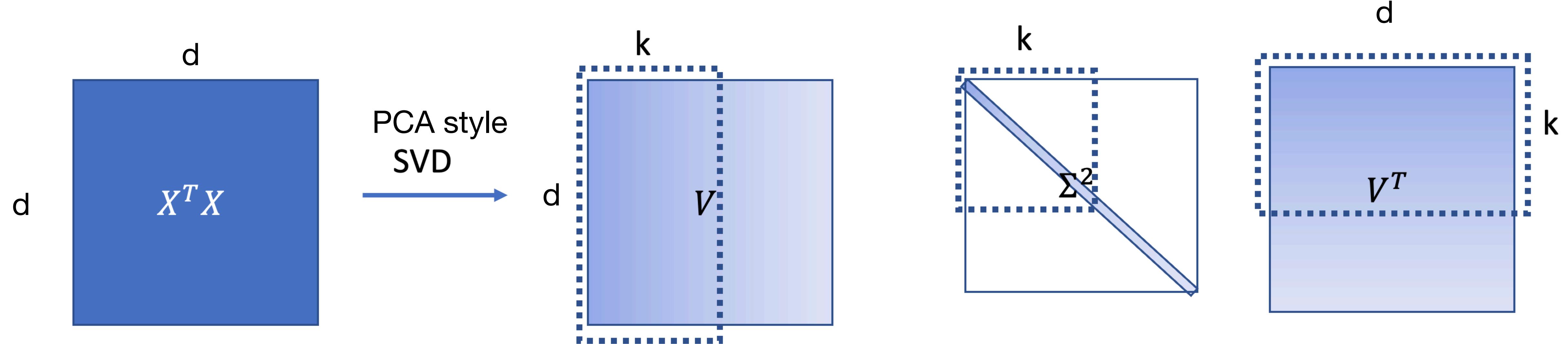
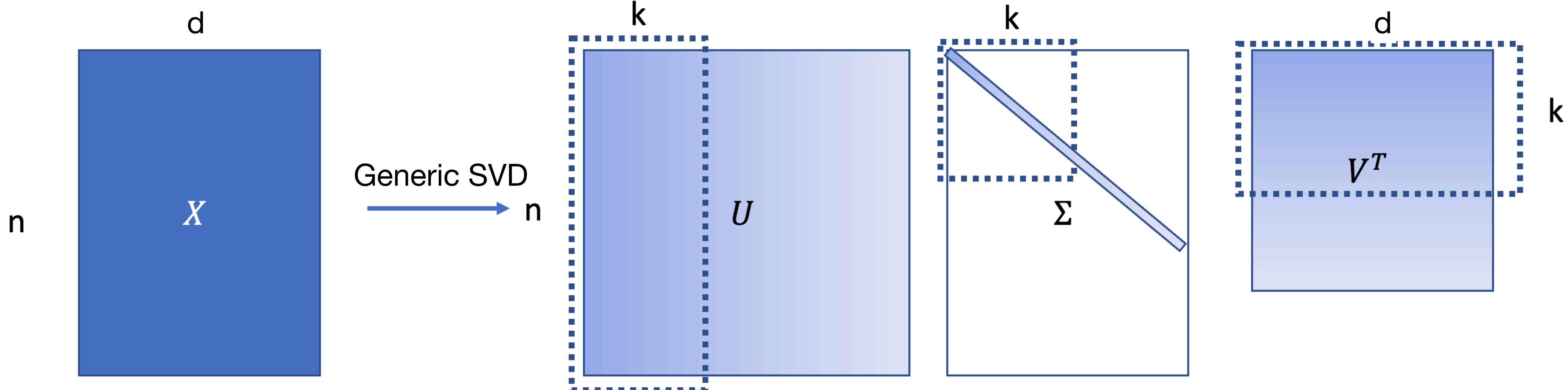
Singular value decomposition is the key part of principal components analysis.

The SVD of the  $N \times p$  matrix  $\mathbf{X}$  has the form  $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ .

- $\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_N)$  is an  $N \times N$  *orthogonal matrix*.  $\mathbf{u}_j, j = 1, \dots, N$ , form an orthonormal basis for the space spanned by the column vectors of  $\mathbf{X}$ .
- $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_p)$  is an  $p \times p$  *orthogonal matrix*.  $\mathbf{v}_j, j = 1, \dots, p$ , form an orthonormal basis for the space spanned by the row vectors of  $\mathbf{X}$ .
- $\mathbf{D}$  is a  $N \times p$  *rectangular matrix* with nonzero elements along the first  $p \times p$  submatrix diagonal.  $\text{diag}(\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_p)$ ,  $\mathbf{d}_1 \geq \mathbf{d}_2 \geq \dots \geq \mathbf{d}_p \geq 0$  are the singular values of  $\mathbf{X}$  with  $N > p$ .

The columns of  $\mathbf{V}$  (i.e.,  $\mathbf{v}_j, j = 1, \dots, p$ ) are the eigenvectors of  $\mathbf{X}^T\mathbf{X}$ . They are called *principal component direction* of  $\mathbf{X}$ .

The diagonal values in  $\mathbf{D}$  (i.e.,  $\mathbf{d}_j, j = 1, \dots, p$ ) are the square roots of the eigenvalues of  $\mathbf{X}^T\mathbf{X}$ .



# Singular Value Decomposition

To generate principle components:

- Subtract mean  $\bar{x} = \frac{1}{N} \sum_{n=1}^N x^n$  from each data point, to create zero-centered data
- Create matrix  $X$  with one row vector per (zero centered) data point
- Solve SVD:  $X = USV^T$
- Output Principle components: columns of  $V$  (= rows of  $V^T$ )
  - Eigenvectors in  $V$  are sorted from largest to smallest eigenvalues
  - $S$  is diagonal, with  $s_k^2$  giving eigenvalue for kth eigenvector

# Singular Value Decomposition

To project a point (column vector  $x$ ) into PC coordinates:

$$V^T x$$

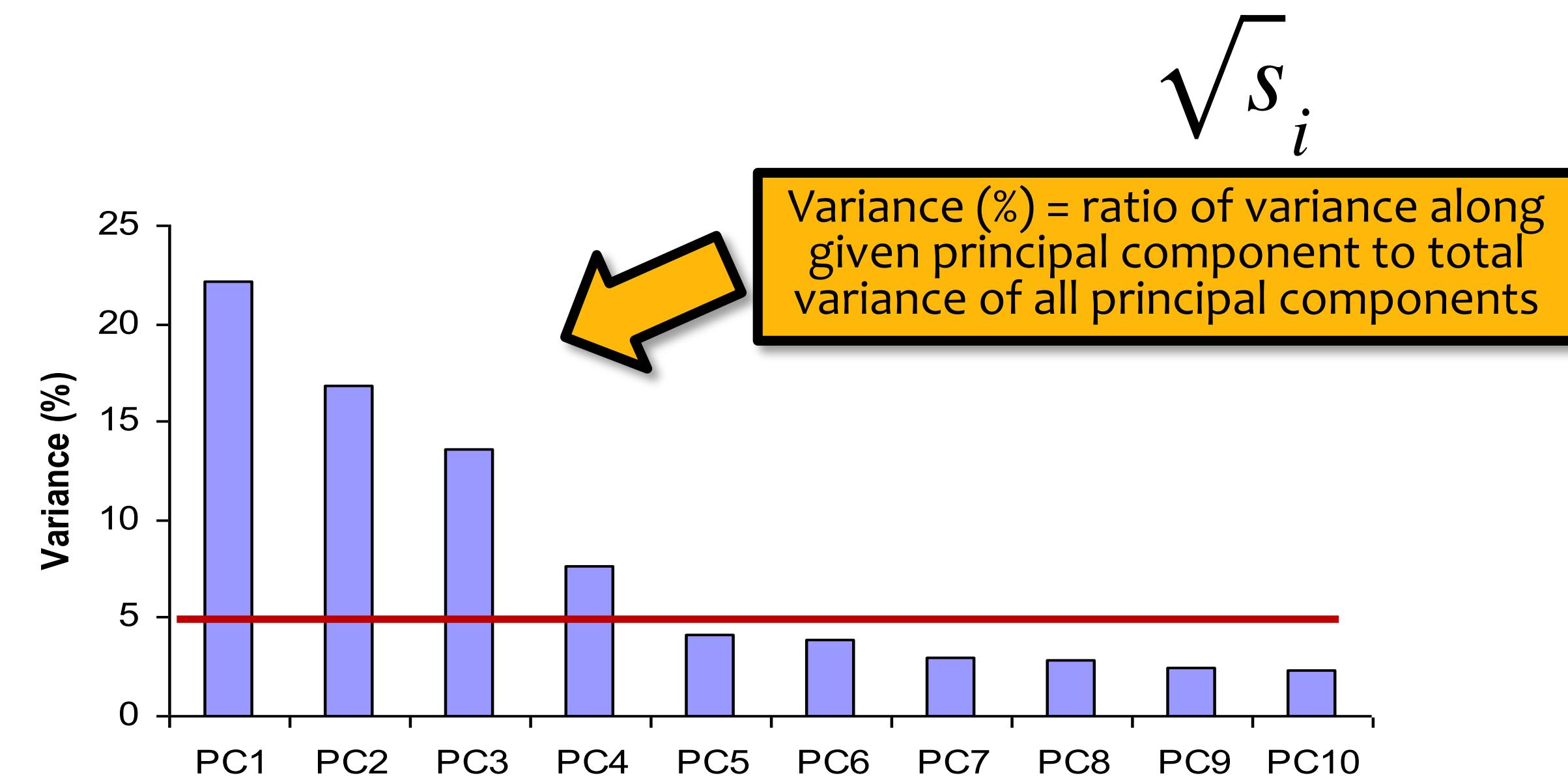
If  $x_i$  is  $i^{\text{th}}$  row of data matrix  $X$ , then

- ( $i^{\text{th}}$  row of  $US$ ) =  $V^T x_i^T$
- $(US)^T = V^T X^T$

To project a column vector  $x$  to  $M$  dim Principle Components subspace, take just the first  $M$  coordinates of  $V^T x$

# Dimensionality REDUCTION

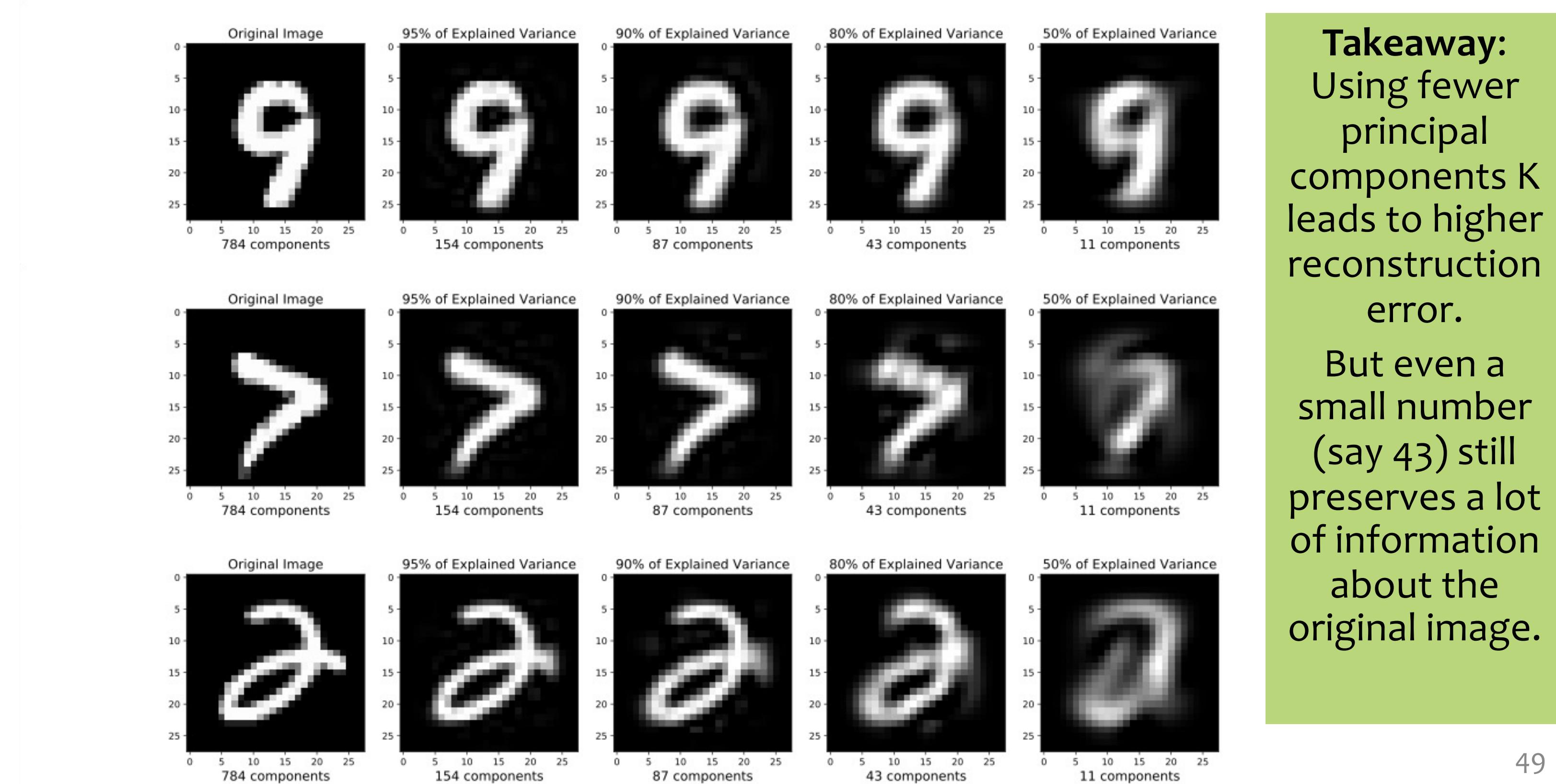
## Choose k biggest eigenvectors



# Projecting MNIST digits

## Task Setting:

1. Take each 28x28 image of a digit (i.e. a vector  $\mathbf{x}^{(i)}$  of length 784) and project it down to K components (i.e. a vector  $\mathbf{u}^{(i)}$ )
2. Report percent of variance explained for K components
3. Then project back up to 28x28 image (i.e. a vector  $\tilde{\mathbf{x}}^{(i)}$  of length 784) to visualize how much information was preserved

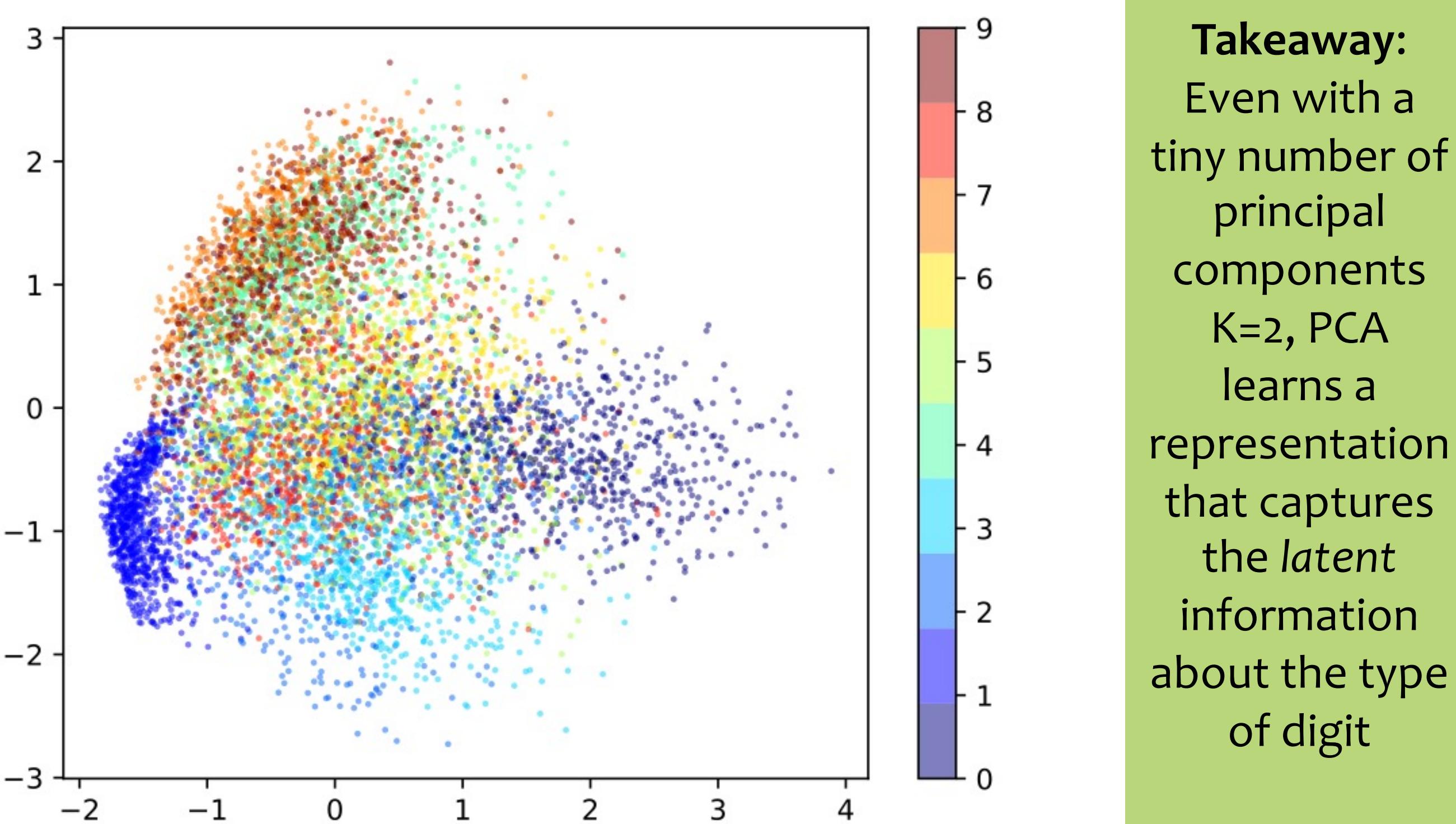


**Takeaway:**  
Using fewer principal components K leads to higher reconstruction error.  
But even a small number (say 43) still preserves a lot of information about the original image.

# Projecting MNIST digits

## Task Setting:

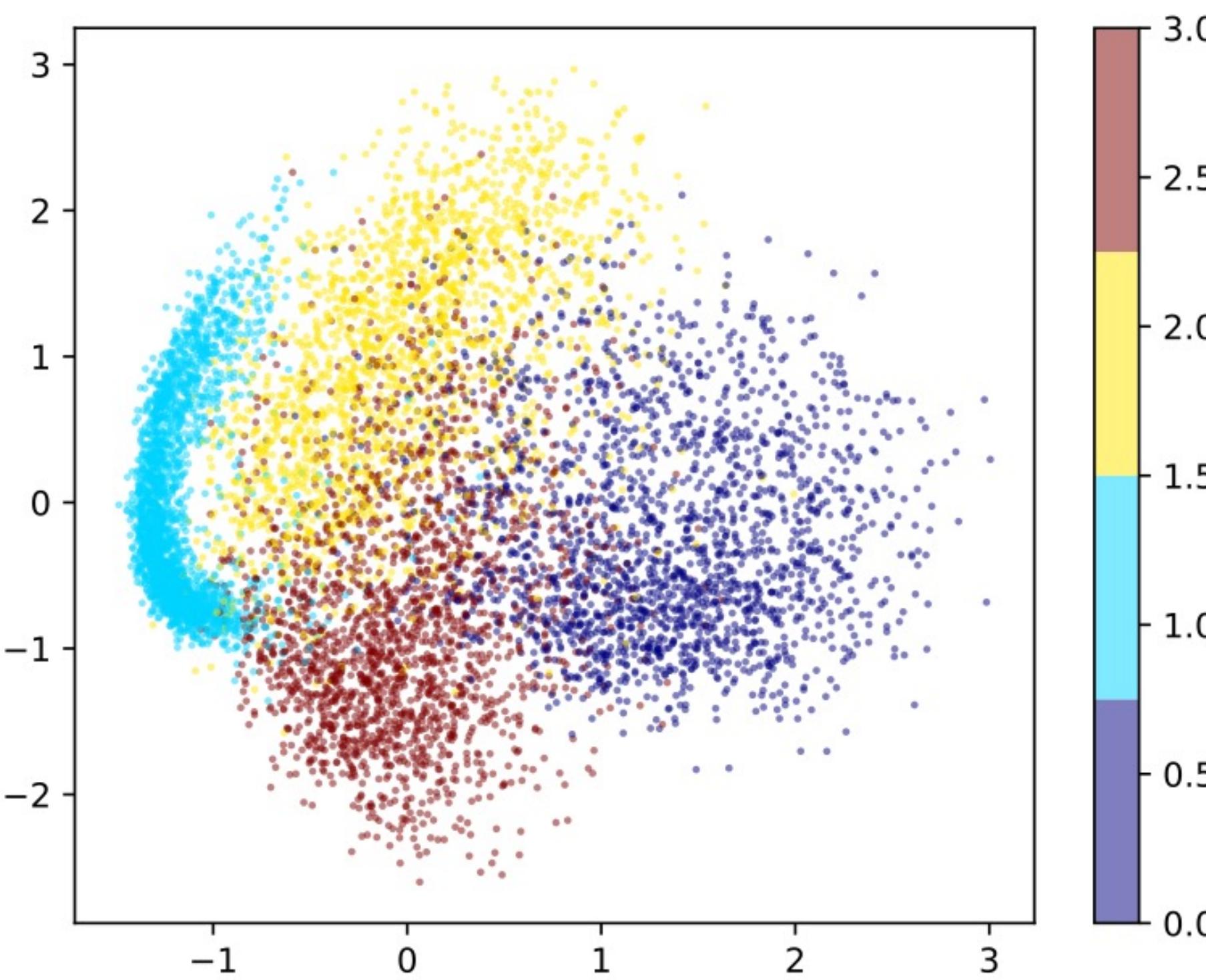
1. Take each 28x28 image of a digit (i.e. a vector  $\mathbf{x}^{(i)}$  of length 784) and project it down to K=2 components (i.e. a vector  $\mathbf{u}^{(i)}$ )
2. Plot the 2 dimensional points  $\mathbf{u}^{(i)}$  and label with the (unknown to PCA) label  $y^{(i)}$  as the color
3. Here we look at all ten digits 0 - 9



# Projecting MNIST digits

## Task Setting:

1. Take each  $28 \times 28$  image of a digit (i.e. a vector  $\mathbf{x}^{(i)}$  of length 784) and project it down to  $K=2$  components (i.e. a vector  $\mathbf{u}^{(i)}$ )
2. Plot the 2 dimensional points  $\mathbf{u}^{(i)}$  and label with the (unknown to PCA) label  $y^{(i)}$  as the color
3. Here we look at just four digits 0, 1, 2, 3



**Takeaway:**  
Even with a tiny number of principal components  $K=2$ , PCA learns a representation that captures the *latent* information about the type of digit