



Lecture 12: Exploring Data Through Preprocessing and Unsupervised ML Part 2

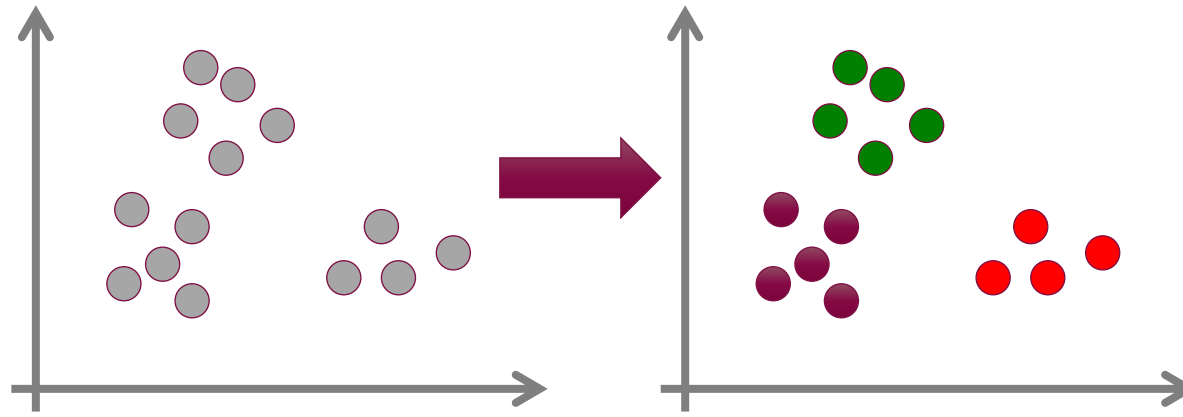
Feb 22, 2023

CIS 4190/5190

Spring 2023

Recap: Clustering

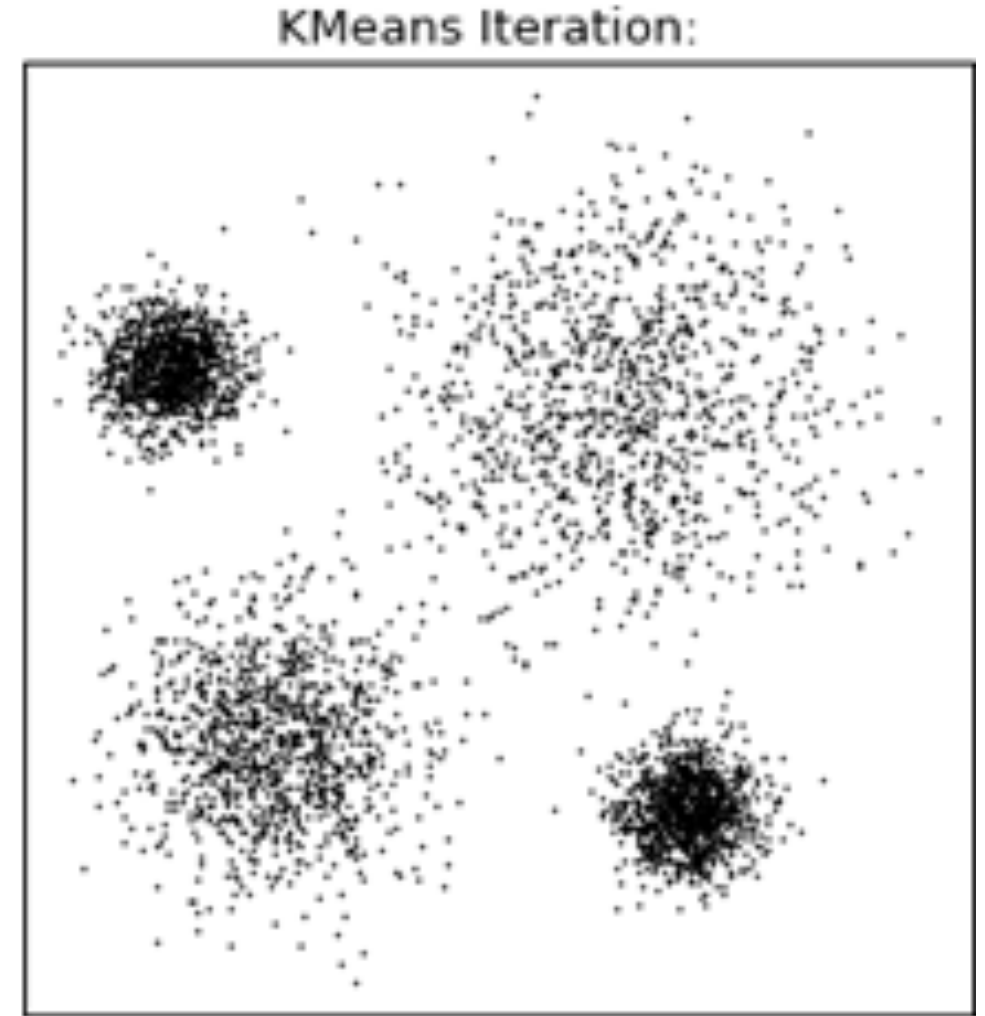
What natural groupings exist in this data?



Recap: K-Means Clustering

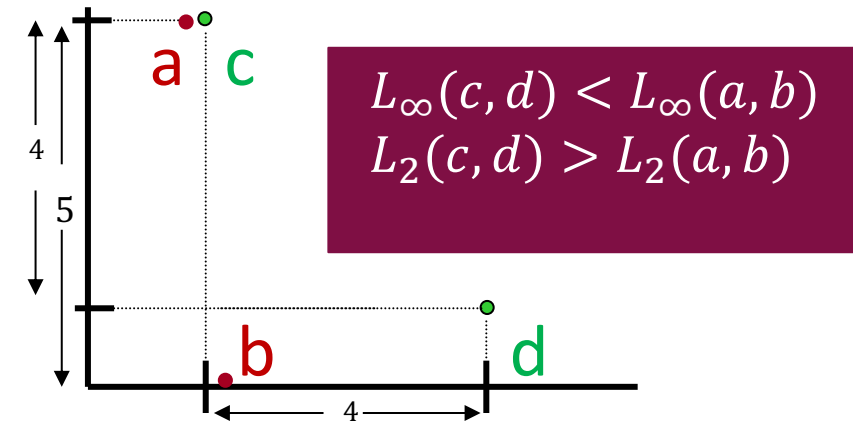
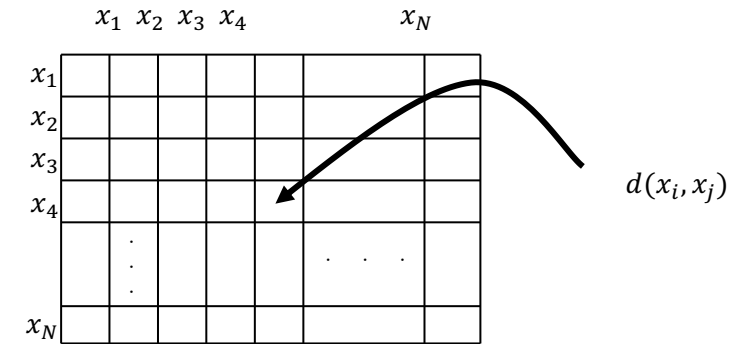
K-Means (K, X)

- Randomly choose K cluster center locations (centroids)
- Loop until convergence, do:
 - Assign each point to the cluster of the closest centroid
 - Re-estimate the cluster centroids based on the data assigned to each cluster



Recap: The Choice of Distance Function

- Clustering techniques all usually accept a matrix of pairwise distances between data points as input.
- The choice of distance function affects the clustering outcomes. This boils down to: different distance functions might consider different point pairs more similar.



$$L_{\infty}(a, b) = 5$$

$$L_2(a, b) = (5^2 + \varepsilon^2)^{\frac{1}{2}} = 5 + \varepsilon$$

$$L_{\infty}(c, d) = 4$$

$$L_2(c, d) = (4^2 + 4^2)^{\frac{1}{2}} = 4\sqrt{2} = 5.66$$

Mahalanobis distance

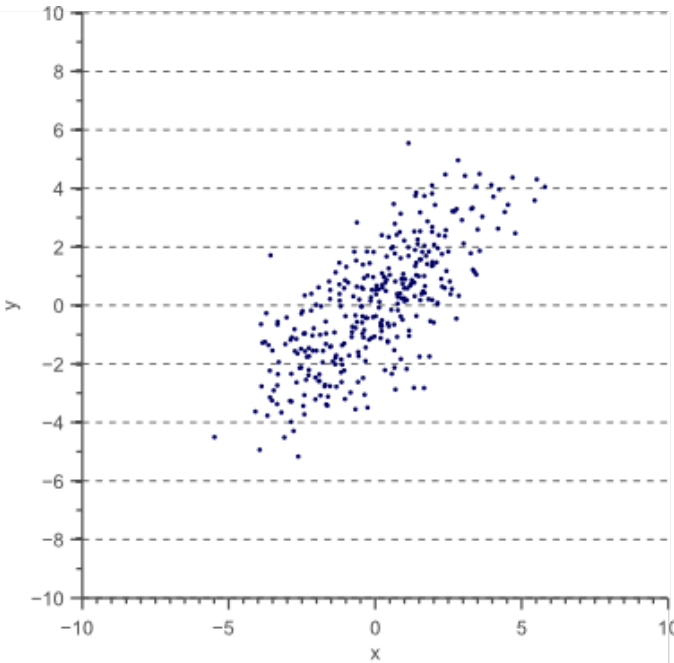


- One common choice is to tie the distance measure itself to the structure of the data.
- **Mahalanobis Distance:** $d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$
 - $\mu = \frac{1}{m} \sum_{i=1}^m x_i$ is the mean vector, which represents the average of the data
 - $\Sigma = \frac{1}{m} \sum_{i=1}^m (x - \mu)(x - \mu)^T$ is the covariance matrix of the data.
- When Σ is identity, this is the same as Euclidean distance.
- In 1D, this measures how many standard deviations away two points are.
- The Mahalanobis distance generalizes this to higher dimensions ...

Covariance Matrix Of Data

For zero-centered data,

$$\text{Covariance} = \Sigma = \mathbb{E}[\mathbf{x}_i \mathbf{x}_i^T] = \mathbb{E} \begin{bmatrix} x_{i1}x_{i1} & \cdots & x_{i1}x_{iD} \\ \vdots & x_{ij}x_{ik} & \vdots \\ x_{iD}x_{i1} & \cdots & x_{iD}x_{iD} \end{bmatrix}$$



$$\sigma(x, y) = \mathbb{E}[(x - \mathbb{E}(x))(y - \mathbb{E}(y))]$$

$$\Sigma = \begin{bmatrix} \sigma(x, x) & \sigma(x, y) \\ \sigma(y, x) & \sigma(y, y) \end{bmatrix}$$

Covariance Matrix in Terms of Data Matrix X

$$\text{Covariance} = \Sigma = \mathbb{E}[\mathbf{x}_i \mathbf{x}_i^T] = \mathbb{E} \begin{bmatrix} x_{i1}x_{i1} & \cdots & x_{i1}x_{iD} \\ \vdots & x_{ij}x_{ik} & \vdots \\ x_{iD}x_{i1} & \cdots & x_{iD}x_{iD} \end{bmatrix} = \frac{1}{N} \sum_i \mathbf{x}_i \mathbf{x}_i^T$$

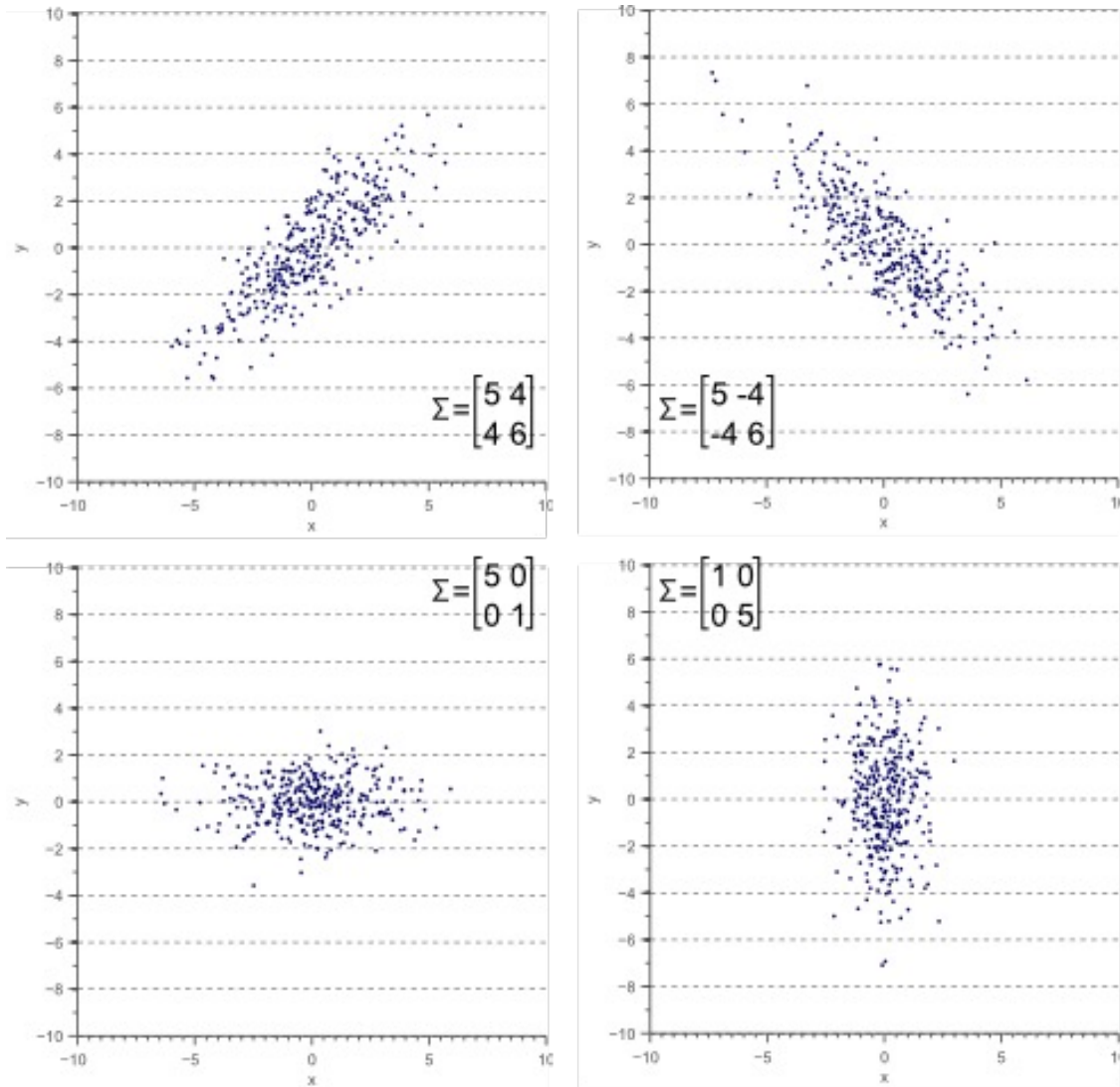
$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

$$X^T = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_n]$$

$$\frac{1}{N} X^T X = \frac{1}{N} (\mathbf{x}_1 \mathbf{x}_1^T + \mathbf{x}_2 \mathbf{x}_2^T + \cdots + \mathbf{x}_N \mathbf{x}_N^T)$$

Thus, the data covariance matrix is typically computed as $\frac{1}{N} X^T X$

Covariance Matrix Is Related to Dataset “Shape”



“Distances matter more when they are along directions in which the data varies less.”



Mahalanobis Distance: $d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$

Covariance Matrix Of Data

“Distances matter more when they are along directions in which the data varies less.”



Mahalanobis Distance: $d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$

pink and green distance are equal in the Euclidean distance sense.

pink distance > green distance in the Mahalanobis distance sense.

Summary of Clustering

- Critical to understanding the structure of our data
- Often useful for creating high-level features useful for supervised learning
- We saw one approach in detail: K-Means

Optional readings: Clustering

- Bishop Ch 9.1 on K-Means Clustering: <https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf>
- Hastie and Tibshirani, Elements of Statistical Learning, Ch 14.5.1 and 14.5.2. <https://hastie.su.domains/ElemStatLearn/>
- Hands-On ML Unsupervised ML: https://github.com/ageron/handson-ml2/blob/master/09_unsupervised_learning.ipynb (Play with lots of clustering approaches, including K-Means in detail)
- Scikit-Learn documentation of clustering approaches: <https://scikit-learn.org/stable/modules/clustering.html#clustering>



Dimensionality Reduction

Dimensionality Reduction

Dimensionality Reduction

Map samples $\mathbf{x}_i \in \mathbb{R}^D$ to $f(\mathbf{x}_i) \in \mathbb{R}^{D' \ll D}$

Can think of this as generalizing clustering, $f(\mathbf{x}_i) \in \mathbb{N}^1 \rightarrow f(\mathbf{x}_i) \in \mathbb{R}^{D' \ll D}$

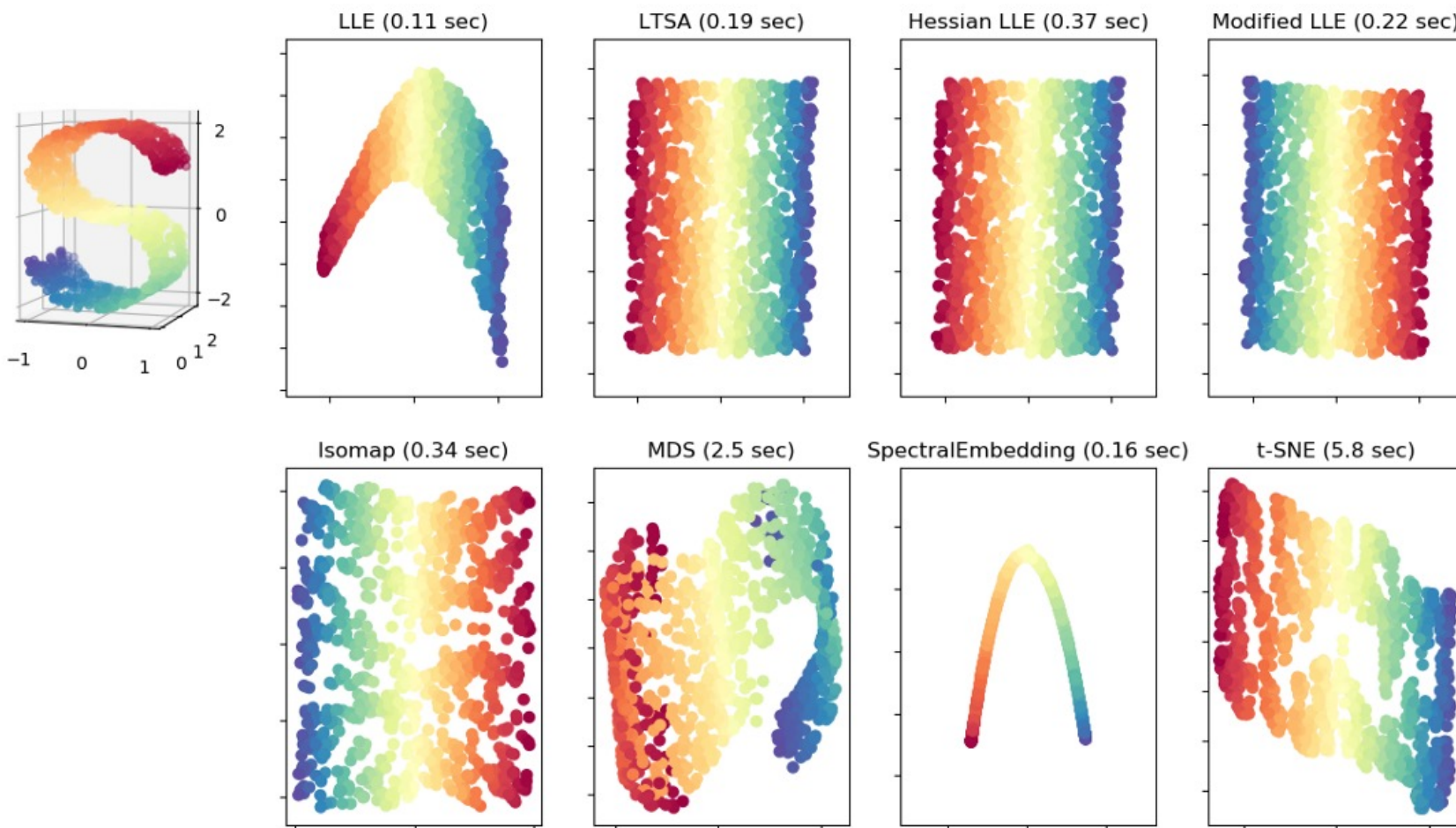
- Rather than groupings, we want to recover “low-dimensional structure”

Also a generalization of “feature selection”.

- Dimensionality-reduced $f(\mathbf{x}_i)$ need not just have a subset of the elements of the original vector \mathbf{x}_i .

What Is The “Structure” Of A Dataset?

Manifold Learning with 1000 points, 10 neighbors



The Uses of Dimensionality Reduction

- **Feature Learning:** For preprocessing inputs to an ML algorithm, since lower-dimensional features permit smaller models and fewer data samples.
- **Compression (for storage):** e.g. JPEG standard for images is now adopting unsupervised ML approaches https://jpeg.org/items/20190327_press.html
- **Visualization:** Exploring a dataset, or an ML model's outputs

Consider: Visualizing High-Dimensional Data

	LotFrontage	LotArea	Street	LotShape	Utilities	LandSlope	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	ExterQual	ExterCond	BsmtQual	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	...	SaleCondition_Abnorml
0	65.0	8450	2	4	4	3	7	5	2003	2003	196.0	4	3	4	0	6	706	1	...	0
1	80.0	9600	2	4	4	3	6	8	1976	1976	0.0	3	3	4	3	5	978	1	...	0
2	68.0	11250	2	3	4	3	7	5	2001	2002	162.0	4	3	4	1	6	486	1	...	0
3	60.0	9550	2	3	4	3	7	5	1915	1970	0.0	3	3	3	0	5	216	1	...	1
4	84.0	14260	2	3	4	3	8	5	2000	2000	350.0	4	3	4	2	6	655	1	...	0
5	85.0	14115	2	3	4	3	5	5	1993	1995	0.0	3	3	4	0	6	732	1	...	0
6	75.0	10084	2	4	4	3	8	5	2004	2005	186.0	4	3	5	2	6	1369	1	...	0
7	0.0	10382	2	3	4	3	7	6	1973	1973	240.0	3	3	4	1	5	859	4	...	0
8	51.0	9142	2	3	4	3	7	5	1994	1999	0.0	3	3	4	0	6	732	1	...	1
9	50.0	9600	2	4	4	3	6	8	1976	1976	0.0	3	3	4	3	5	978	1	...	0
10	70.0	11250	2	3	4	3	7	5	2001	2002	162.0	4	3	4	1	6	486	1	...	0
11	85.0	14260	2	3	4	3	8	5	2000	2000	350.0	4	3	4	2	6	655	1	...	0
12	0.0	10382	2	3	4	3	7	6	1973	1973	240.0	3	3	4	1	5	859	4	...	0
13	91.0	15113	2	3	4	3	8	5	2000	2000	350.0	4	3	4	2	6	655	1	...	0
14	0.0	10382	2	3	4	3	7	6	1973	1973	240.0	3	3	4	1	5	859	4	...	0
15	51.0	9142	2	3	4	3	7	5	1994	1999	0.0	3	3	4	0	6	732	1	...	0
16	0.0	10382	2	3	4	3	7	6	1973	1973	240.0	3	3	4	1	5	859	4	...	0
17	72.0	10791	2	4	4	3	4	5	1967	1967	0.0	3	3	0	0	0	0	0	...	0
18	66.0	13695	2	4	4	3	5	5	2004	2004	0.0	3	3	3	0	6	646	1	...	0
19	70.0	7560	2	4	4	3	5	6	1958	1965	0.0	3	3	3	0	2	504	1	...	1
20	101.0	14215	2	3	4	3	8	5	2005	2006	380.0	4	3	5	2	1	0	1	...	0
21	57.0	7449	2	4	4	3	7	7	1930	1950	0.0	3	3	3	0	1	0	1	...	0
22	75.0	9742	2	4	4	3	8	5	2002	2002	281.0	4	3	4	0	1	0	1	...	0
23	44.0	4224	2	4	4	3	5	7	1976	1976	0.0	3	3	4	0	6	840	1	...	0

“To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say 'fourteen' to yourself very loudly. Everyone does it.”

- Geoff Hinton

227 features

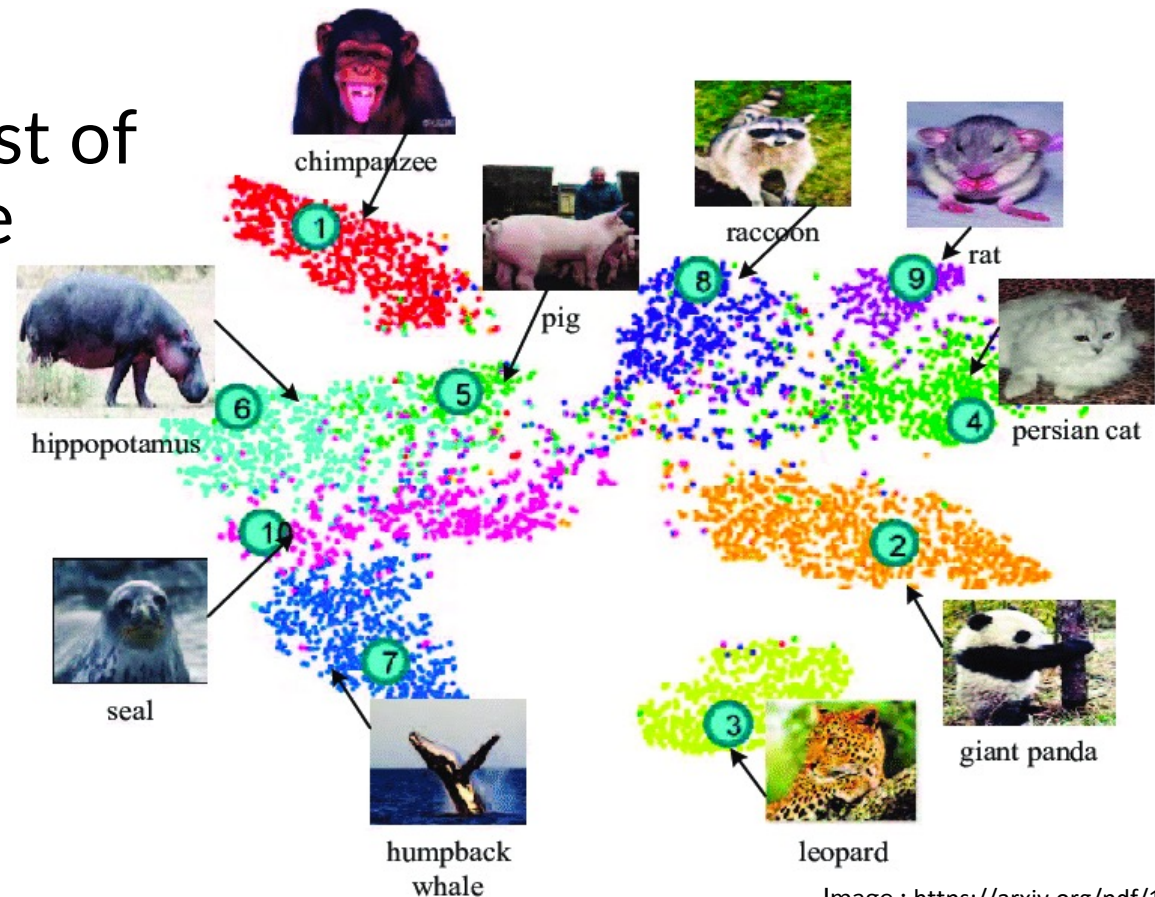
Data Visualization

Is there a representation better than the raw features?

Maybe it isn't necessary to visualize all 227 dimensions

Idea: find a **lower-dimensional subspace** that retains most of the information about the original data

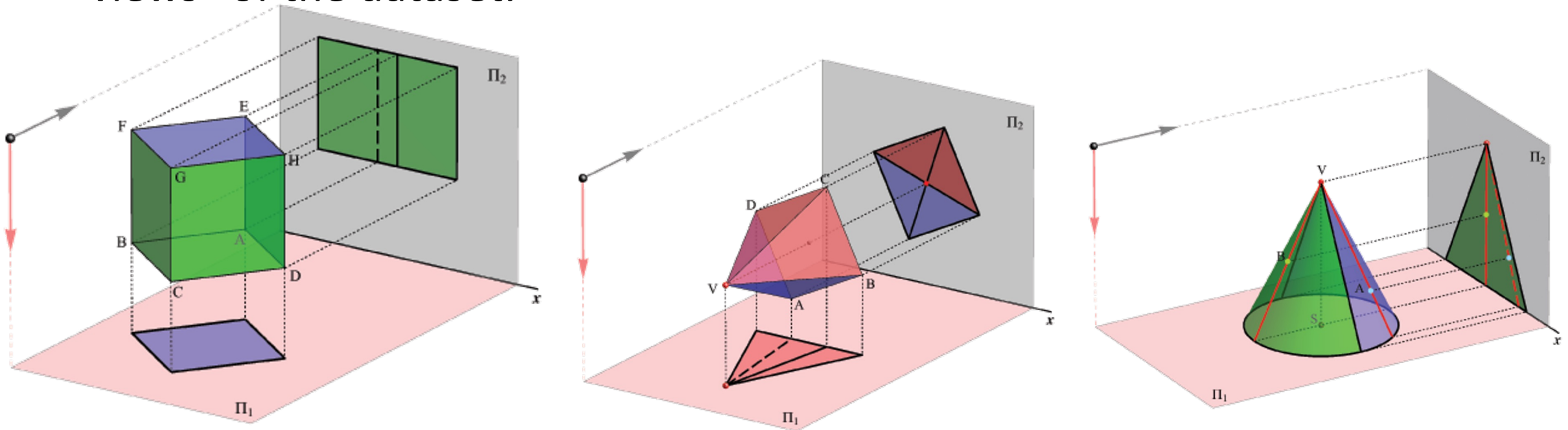
There are many methods;
our focus will be on **Principal Components Analysis**



Principal Components Analysis

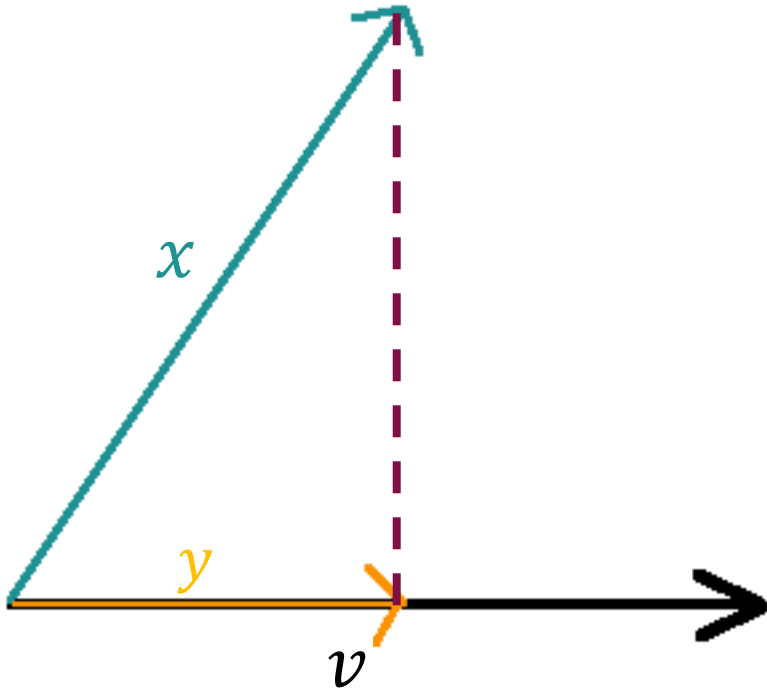
Dimensionality Reduction Through Orthogonal Projections?

- We often view 3D objects in 2D by “projecting them” onto a plane. Drop perpendicular lines from every point on the object to the plane.
- “Good projections” are views that preserve information about the shape of the data.
- PCA does something similar to every instance in a dataset. Finds good “views” of the dataset.



Orthogonal Projection Example: from 2D to 1D

- Let's project $x \in \mathbb{R}^2$ down to a new vector $v \in \mathbb{R}^1$ (i.e., a scalar), by orthogonally projecting onto the direction represented by the unit vector v



$$y = (x^T v)v$$

Orthogonal Projection Of An Entire Dataset?

- Every point in the set is projected
- E.g., projecting a 3D dataset in XYZ (see figure, left) onto:
 - the XY plane (top), or
 - the YZ plane (bottom)
- Which of these “views” is better in terms of preserving info about the structure of the data?
- In general, projections need not be axis-aligned. How to find good structure-preserving views?
 - Solution: PCA!

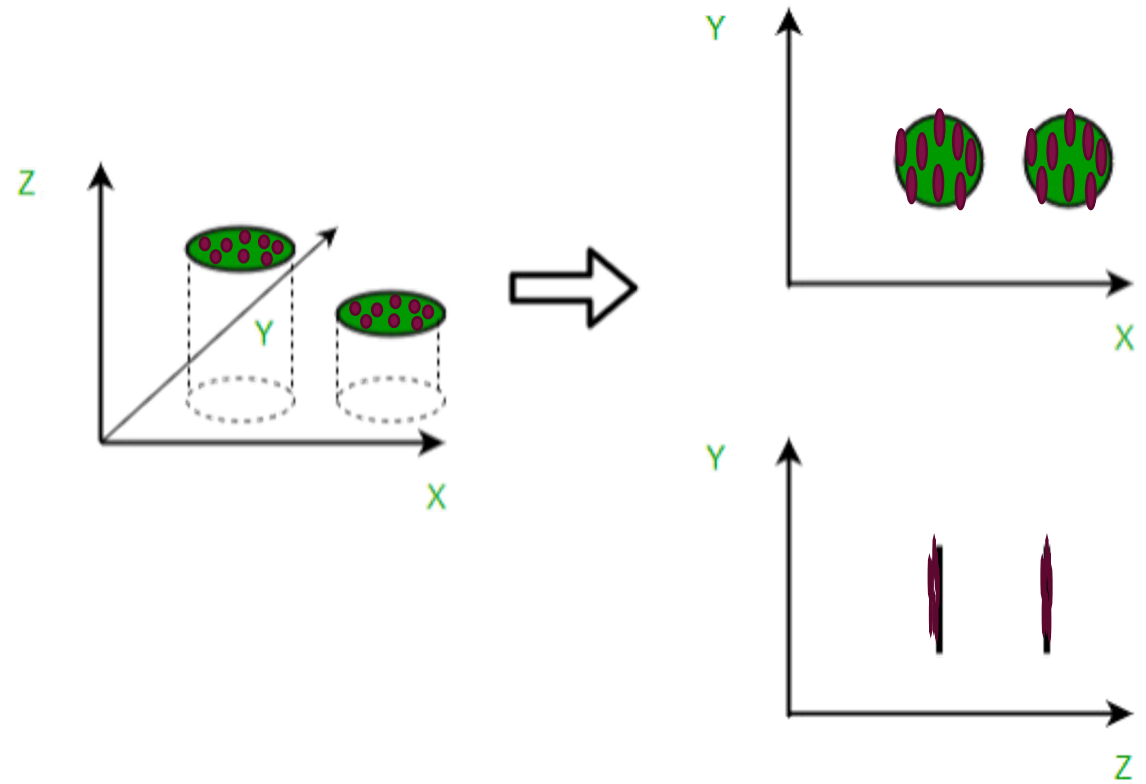
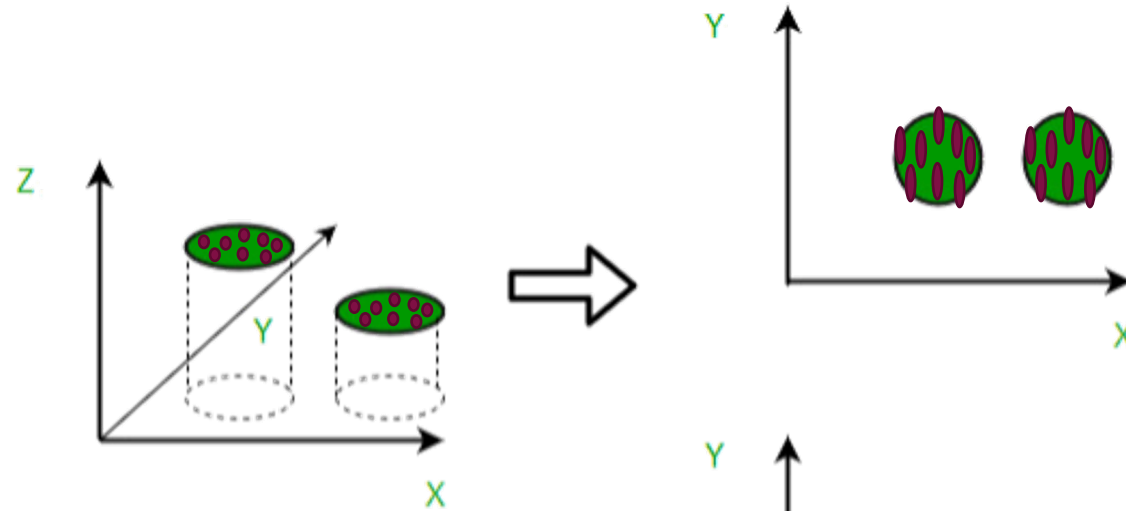


Fig: <https://www.geeksforgeeks.org/dimensionality-reduction/>

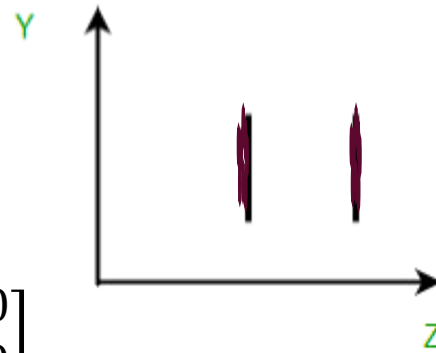
Orthogonal Projection Of An Entire Dataset?



$$\begin{pmatrix} x_i^T & \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{pmatrix} x_i^T & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The new dimensionality-reduced vector has only these two elements.

$$\begin{aligned} \mathbf{x}_i &= \begin{bmatrix} x_{i1} \\ \vdots \\ x_{i3} \end{bmatrix}_D = (x_i^T e_X) e_X + (x_i^T e_Y) e_Y + (x_i^T e_Z) e_Z \\ &= \begin{pmatrix} x_i^T & \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{pmatrix} x_i^T & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{pmatrix} x_i^T & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$



$$\begin{pmatrix} x_i^T & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{pmatrix} x_i^T & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{pmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Thus, each choice of view can be parameterized by the **basis vectors**

So, finding good views = finding good **basis vectors**.

PCA Dimensionality Reduction Objective

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix}_{N \times D}$$

We can write each row (each data sample) \mathbf{x}_i as:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix}_D = \sum_d (\underbrace{x_{id}}_{\text{Projections}} \cdot \underbrace{e_d}_{\text{Original axes}}) e_d$$

We are looking for a new coordinate system $\mathbf{v}_1, \dots, \mathbf{v}_{D'}$ to approximate all \mathbf{x}_i :

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx (\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1 + (\mathbf{x}_i \cdot \mathbf{v}_2) \mathbf{v}_2 + \cdots + (\mathbf{x}_i \cdot \mathbf{v}_{D'}) \mathbf{v}_{D'}$$

where the new axes \mathbf{v}_d 's are all D -dimensional unit norm, and $D' \ll D$

Terminology

We are looking for a new coordinate system $\mathbf{v}_1, \dots, \mathbf{v}_{D'}$ to approximate all \mathbf{x}_i :

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx (\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1 + (\mathbf{x}_i \cdot \mathbf{v}_2) \mathbf{v}_2 + \dots + (\mathbf{x}_i \cdot \mathbf{v}_{D'}) \mathbf{v}_{D'}$$

where the new axes \mathbf{v}_d 's are all D -dimensional unit norm, and $D' \ll D$

- The axis unit vectors \mathbf{v}_d of the projection are also called “basis” vectors

- The final D' -dimensional vector representation is simply the vector of

projections $\begin{bmatrix} (\mathbf{x}_i \cdot \mathbf{v}_1) \\ \vdots \\ (\mathbf{x}_i \cdot \mathbf{v}_{D'}) \end{bmatrix}$

Simplest Case: Reduce to $D' = 1$ dimension

We are looking for a new coordinate system $\mathbf{v}_1, \dots, \mathbf{v}_{D'}$ to approximate all \mathbf{x}_i :

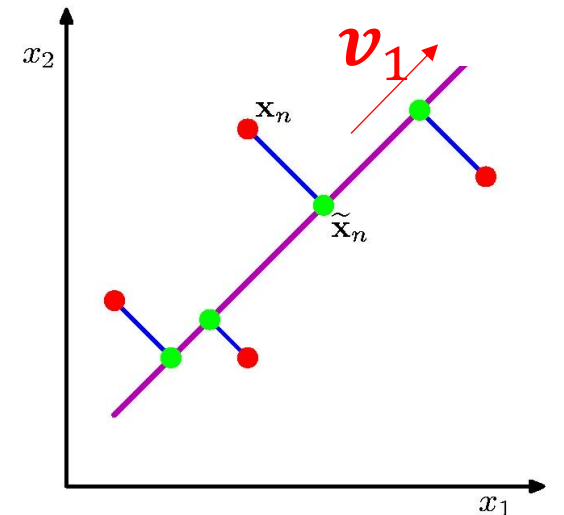
$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx (\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1 + (\mathbf{x}_i \cdot \mathbf{v}_2) \mathbf{v}_2 + \dots + (\mathbf{x}_i \cdot \mathbf{v}_{D'}) \mathbf{v}_{D'}$$

where the new axes \mathbf{v}_d 's are all D -dimensional unit norm, and $D' \ll D$

Simplest case: $D' = 1$?

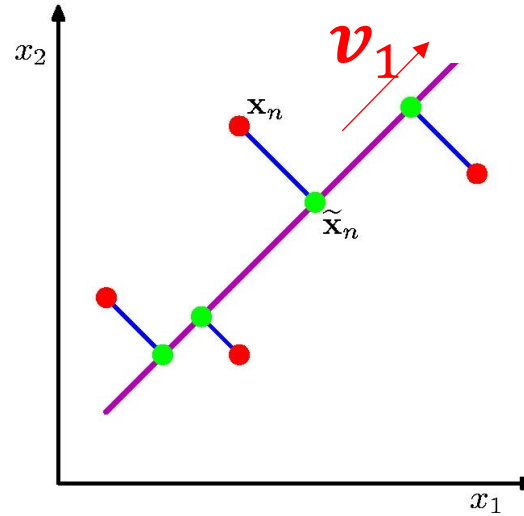
We want to find unit \mathbf{v}_1 such that:

$(\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1$ best approximates \mathbf{x}_i



The Meaning Of “Approximating” The Data

Here,
 $D = 2$
 $D' = 1$



PCA looks for the projection that:

- minimizes mean squared distance between data point and projections (sum of squared blue lines)
- maximizes variance of projected data (roughly, length of purple line)

Objective Function: Maximizing Variance

Find unit vector \mathbf{v}_1 (with $\|\mathbf{v}_1\|_2 = 1$), to optimize:

Reconstruction
MSE

$$\min_{\|\mathbf{v}_1\|_2=1} \frac{1}{N} \sum_i \|(\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1 - \mathbf{x}_i\|_2^2$$

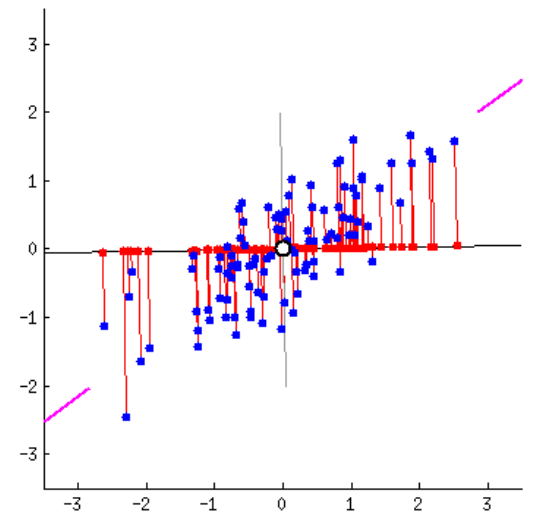
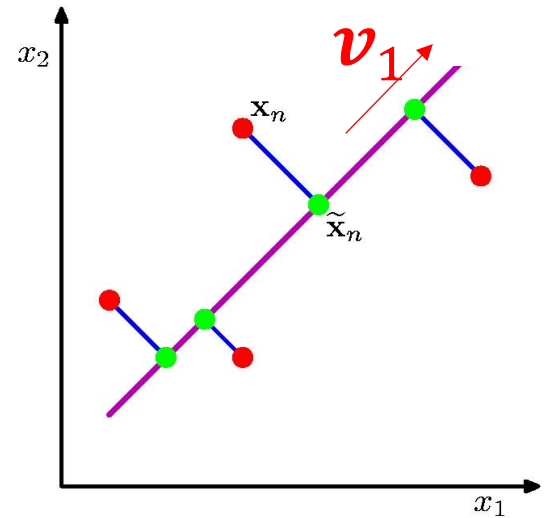
Projection error

Can show, exactly equal to:

$$\max_{\|\mathbf{v}_1\|_2=1} \text{variance}(\mathbf{x}_i \cdot \mathbf{v}_1)$$

Intuitively, if the variance of the projection on \mathbf{v}_1 was low, then \mathbf{v}_1 would not be very informative about samples \mathbf{x}_i .

Conversely, directions with high variance projections preserve the most information.



(Fig: stats.stackexchange)

So, how to find this direction of maximum variance?

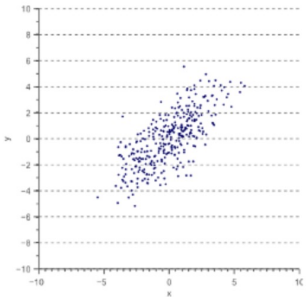
Covariance Matrix To The Rescue Again

- Recall:

Covariance Matrix Of Data

For zero-centered data,

$$\text{Covariance} = \Sigma = \mathbb{E}[\mathbf{x}_i \mathbf{x}_i^T] = \mathbb{E} \begin{bmatrix} x_{i1}x_{i1} & \cdots & x_{i1}x_{iD} \\ \vdots & x_{ij}x_{ik} & \vdots \\ x_{iD}x_{i1} & \cdots & x_{iD}x_{iD} \end{bmatrix}$$

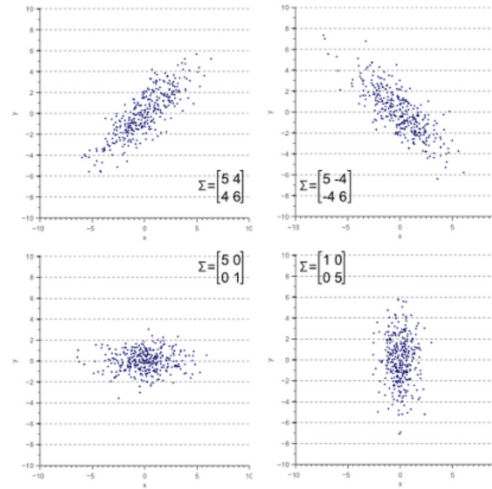


$$\sigma(x, y) = \mathbb{E}[(x - \mathbb{E}(x))(y - \mathbb{E}(y))]$$

$$\Sigma = \begin{bmatrix} \sigma(x, x) & \sigma(x, y) \\ \sigma(y, x) & \sigma(y, y) \end{bmatrix}$$

<https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/>

Covariance Matrix Is Related To Dataset “Shape”



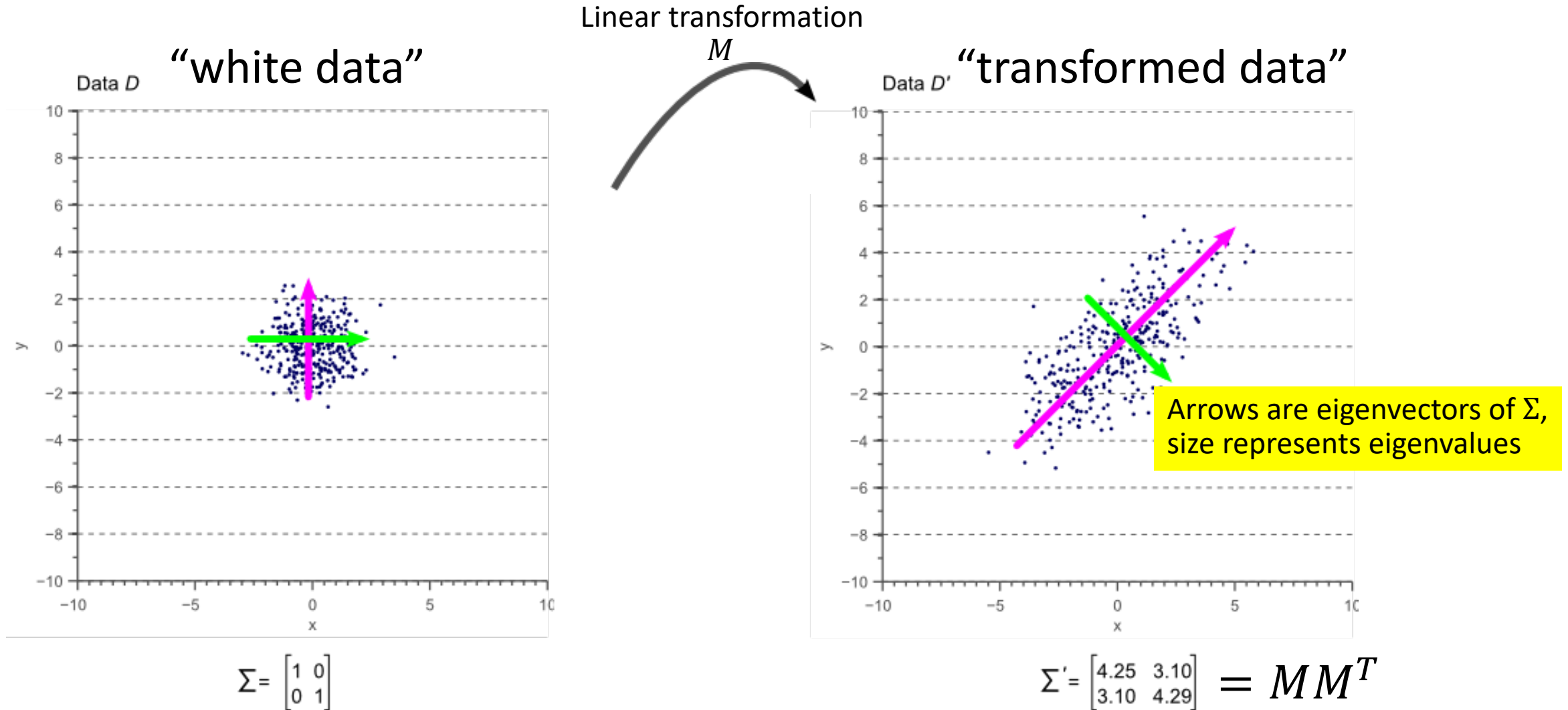
“Distances matter more when they are along directions in which the data varies less.”



Mahalanobis Distance: $d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$

<https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/>

Covariance Matrix Represents a Linear Transformation



Refresher on Eigenvectors & Singular vectors

Eigendecomposition

A square matrix $A_{D \times D}$ can be decomposed as:

$$A = U\Lambda U^{-1}$$

Λ is a $D \times D$ diagonal matrix of “eigenvalues” $diag(\lambda_1, \dots, \lambda_D)$ usually sorted in descending order. Hence, “first eigenvalue” means “largest eigenvalue”

U is a $D \times D$ matrix $[\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_D]$, whose columns are called “eigenvectors”. We usually assume these are normalized to be unit length, i.e., unit eigenvectors.

“First eigenvector” = “largest eigenvector” = “eigenvector with largest eigenvalue”

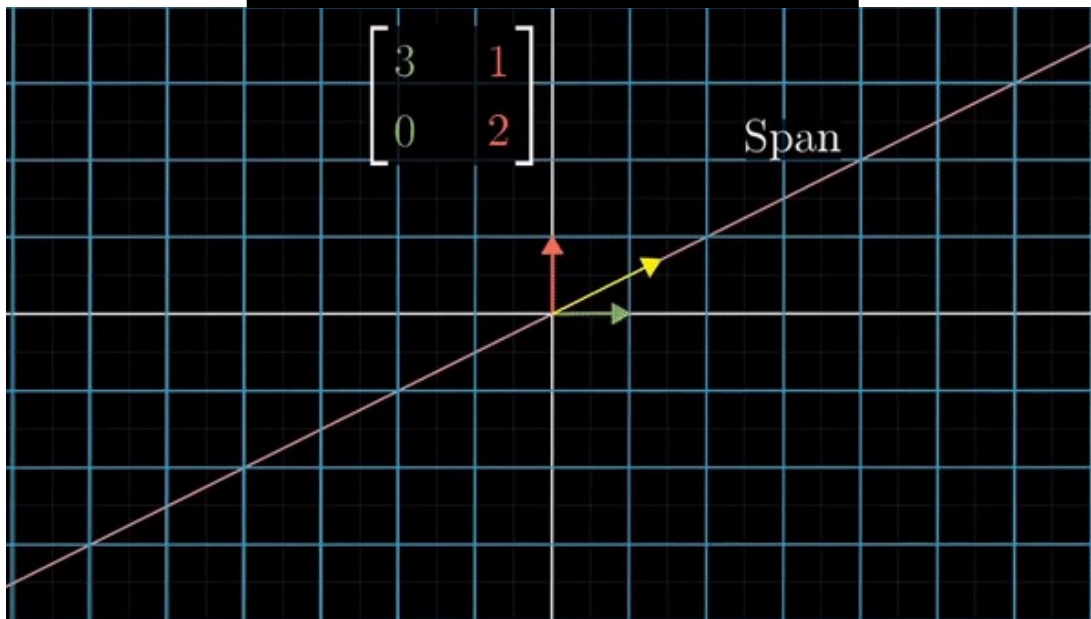
Eigenvectors: geometric intuition

The eigenvectors \mathbf{u}_i of a matrix A are vectors that remain invariant under the linear transformation represented by A i.e. $\mathbf{x} \rightarrow A\mathbf{x}$

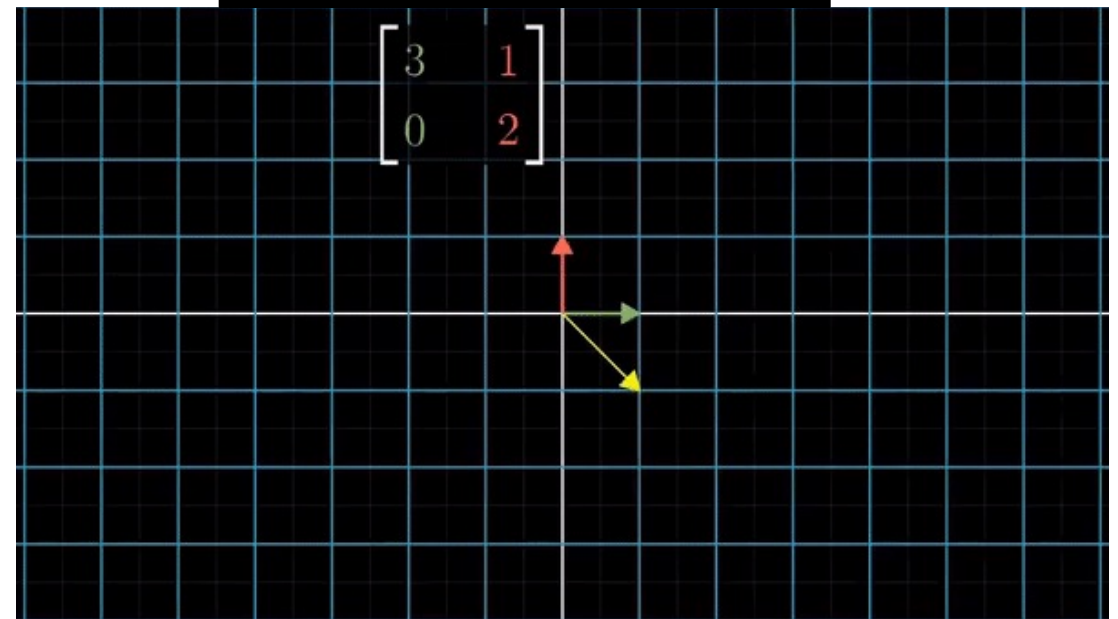
$$A\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

λ_i is the eigenvalue corresponding to \mathbf{u}_i .

Not an eigenvector

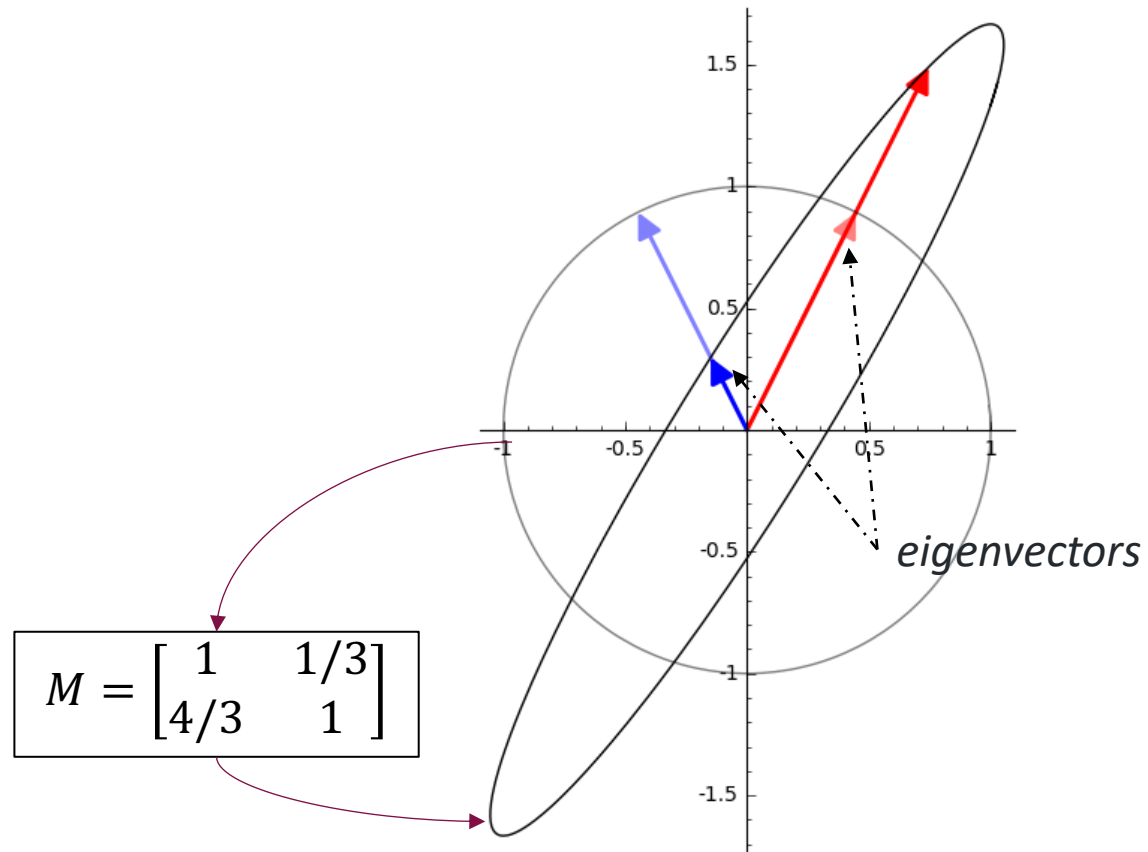


An eigenvector



Singular vectors: geometric intuition

Eigenvectors of M



Vectors that remain unchanged
after the transformation

Singular value decomposition (SVD)

Any matrix A can be decomposed as:

$$A = U\hat{\Lambda}V^T$$

Note: $\hat{\Lambda}$ is usually denoted as Σ , we are using non-standard notation to avoid clashing with covariance matrix Σ

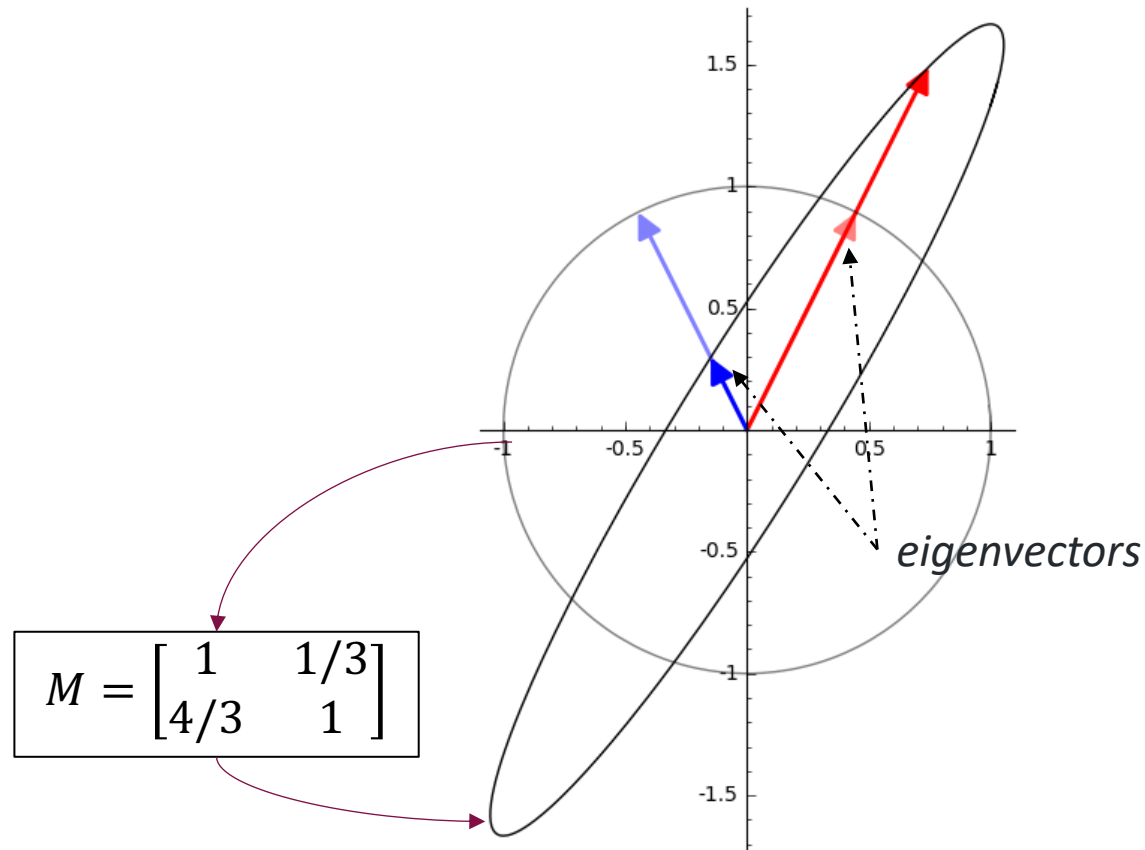
$\hat{\Lambda}$ is a $D \times D$ diagonal matrix of “singular values” $diag(\hat{\lambda}_1, \dots, \hat{\lambda}_D)$ usually sorted in descending order. Hence, “first singular value” means “largest” etc.

U, V are $D \times D$ orthogonal matrices $[\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_D]$ and $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_D]$, whose columns are called “left singular vectors” and “right singular vectors”.

$$\text{Orthogonal} \Rightarrow U^T U = V^T V = I$$

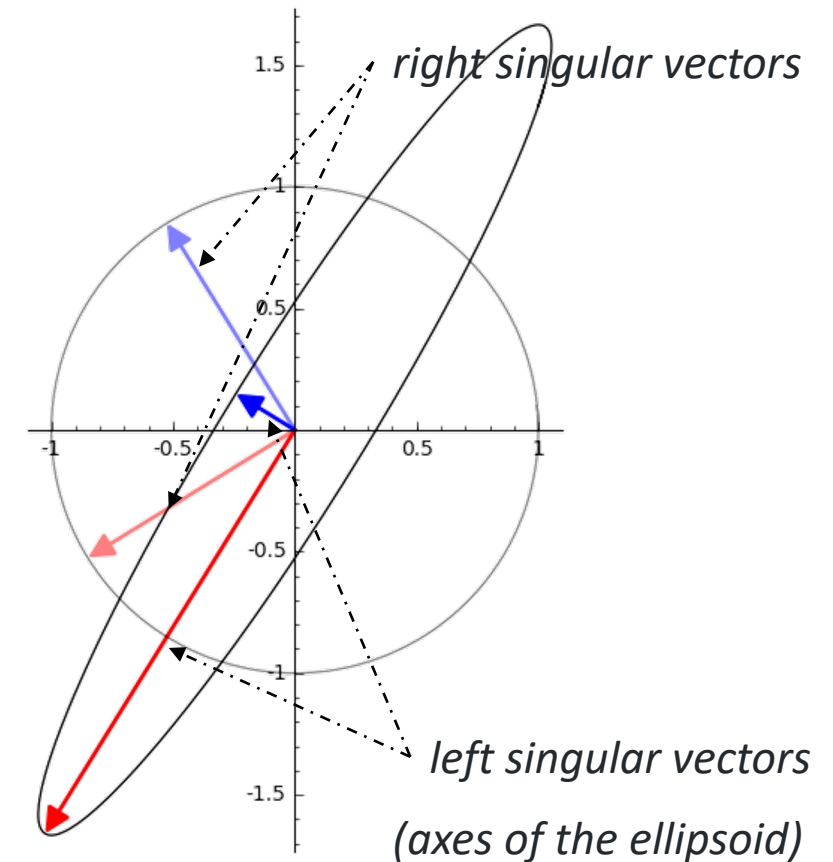
Singular vectors: geometric intuition

Eigenvectors of M



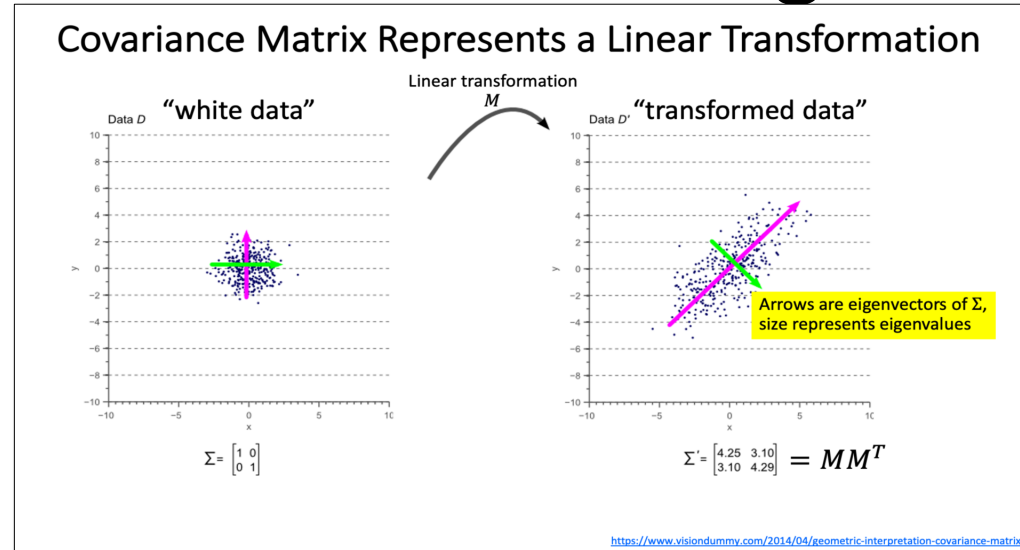
Vectors that remain unchanged after the transformation

Singular vectors of M



Orthogonal set of vectors that remain orthogonal after the transformation

Note: Left Singular Vectors of M = Eigenvectors of MM^T



- Suppose the SVD of $M = U\hat{\Lambda}V^T$
- Then $MM^T = U\hat{\Lambda}V^TV\hat{\Lambda}U^T = U\hat{\Lambda}^2U^T$ = eigendecomposition of MM^T
- In other words,
 - Eigenvectors U of $\Sigma = MM^T$ are the same as left singular vectors of M
 - Also implies that they are orthogonal!
 - Eigenvalues $\hat{\Lambda}^2$ of $\Sigma = MM^T$ are the squares of the singular values of M

So, remember: eigenvectors of covariance matrix = left singular vectors of the corresponding linear transformation

Back to PCA

Objective Function: Maximizing Variance

Find unit vector \mathbf{v}_1 (with $\|\mathbf{v}_1\|_2 = 1$), to optimize:

Reconstruction
MSE

$$\min_{\|\mathbf{v}_1\|_2=1} \frac{1}{N} \sum_i \|(x_i \cdot \mathbf{v}_1) \mathbf{v}_1 - \mathbf{x}_i\|_2^2$$

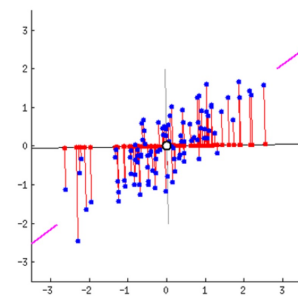
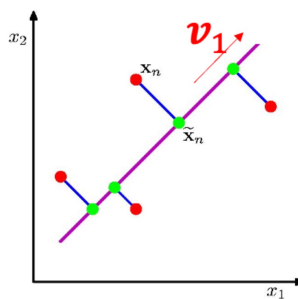
Projection error

Can show, exactly equal to:

$$\max_{\|\mathbf{v}_1\|_2=1} \text{variance}(x_i \cdot \mathbf{v}_1)$$

Intuitively, if the variance of the projection on \mathbf{v}_1 was low, then \mathbf{v}_1 would not be very informative about samples \mathbf{x}_i .

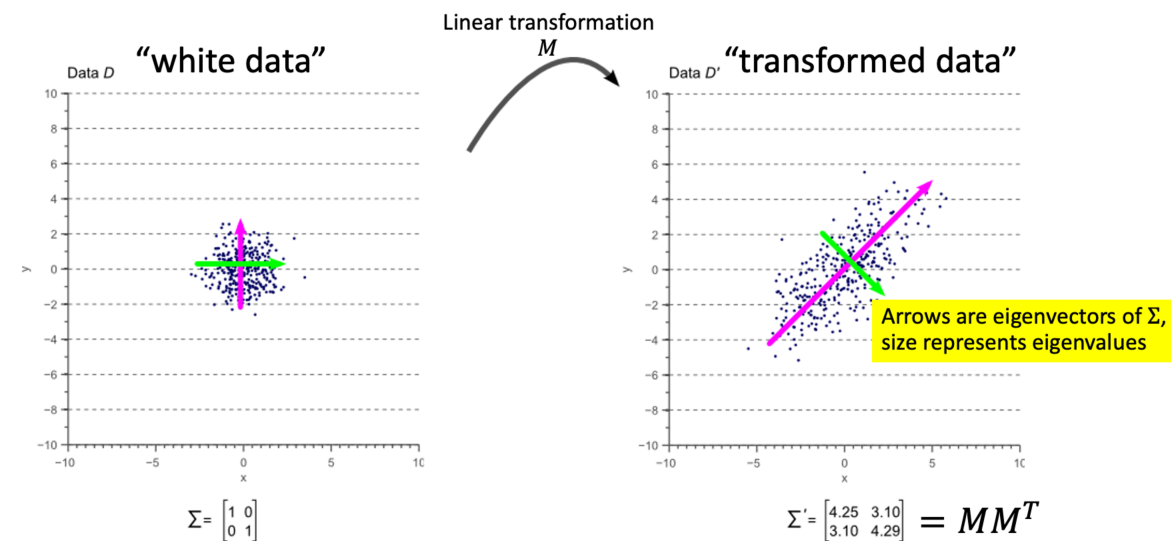
Conversely, directions with high variance projections preserve the most information.



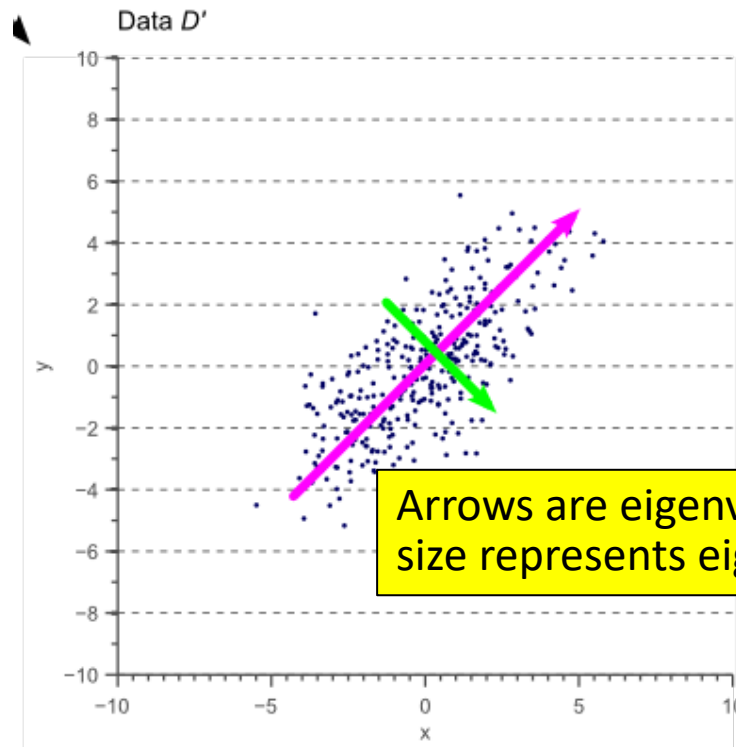
(Fig: stats.stackexchange)

So, how to find this direction of maximum variance?

Covariance Matrix Represents a Linear Transformation



The Largest Eigenvector of the Covariance Matrix



$$\max_{\|v_1\|_2=1} \text{variance}(x_i \cdot v_1)$$

Arrows are eigenvectors of Σ ,
size represents eigenvalues

We can show:

To maximize $\text{variance}(x_i \cdot v_1)$, we can set $v_1 = e_1(\Sigma)$, the first unit eigenvector of Σ

(proof sketch on the next slide)

Proof Sketch (For Your Curiosity)

$$\max_{\|v_1\|_2=1} \text{variance}(x_i \cdot v_1)$$

First, easy to show: $\text{variance}(x_i \cdot v_1) = v_1^T \Sigma v_1$

Claim: To maximize $v_1^T \Sigma v_1$, we can set $v_1 = e_1(\Sigma)$, the first unit eigenvector of Σ

Unit eigenvectors $e_d(\Sigma)$ for symmetric matrices form an orthonormal basis, so any v can be written:

Step 1 \rightarrow

$$v = \sum_{d=1}^D (v \cdot e_d(\Sigma)) e_d(\Sigma)$$

Step 2 \rightarrow

$$\Sigma v = \sum_{d=1}^D (v \cdot e_d(\Sigma)) \Sigma e_d(\Sigma) = \sum_{d=1}^D (v \cdot e_d(\Sigma)) \lambda_d e_d(\Sigma)$$
$$\Sigma v = \sum_{d=1}^D (v \cdot e_d(\Sigma)) \lambda_d e_d(\Sigma)$$

Step 3 \rightarrow

$$v^T \Sigma v = v \cdot (\Sigma v) = \sum_{d=1}^D (v \cdot e_d(\Sigma)) \lambda_d (v \cdot e_d(\Sigma)) = \sum_{d=1}^D (v \cdot e_d(\Sigma))^2 \lambda_d$$

Weighted avg of λ_d s. (Why?)

To maximize the weighted average, assign all your weight to the highest number!

$$\text{So, must set } v \cdot e_1(\Sigma) = 1 \quad \Rightarrow \quad v = e_1(\Sigma)$$

Recap for $D' = 1$ case

- Subtract means, then compute covariance matrix as $\Sigma_1 = X^T X$
- Compute eigendecomposition of Σ_1 (e.g., using singular value decomposition)
- Set $\boldsymbol{v}_1 = \boldsymbol{e}_1(\Sigma_1)$

Note: Right Singular Vectors (X) = Eigenvectors (Σ)

- Let data matrix $X = U\hat{\Lambda}V^T$ (SVD)
- Then $\Sigma = \frac{1}{N}X^TX = \frac{1}{N}V\hat{\Lambda}U^TU\hat{\Lambda}V^T = \frac{1}{N}V\hat{\Lambda}^2V^T$
- So eigenvectors of covariance matrix are also the *right* singular vectors of the data matrix!

More than 1 dimension?

Repeat for $d = 1, \dots, D'$

- Subtract means of all dimensions of X
- Compute $\Sigma_d = X^T X$
- Set $\mathbf{v}_d = \mathbf{e}_1(\Sigma_d)$
- Set $\mathbf{x}_i = \mathbf{x}_i - (\mathbf{x}_i \cdot \mathbf{v}_d) \mathbf{v}_d$ (i.e., subtract current reconstructions to compute residuals... a little bit like gradient boosting!)

Equivalent to simply:

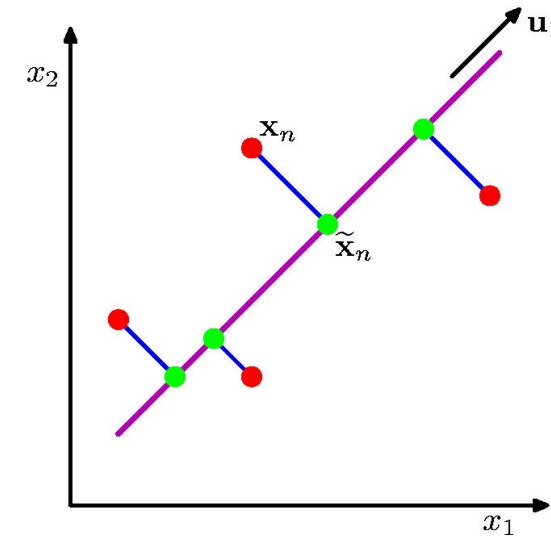
Repeat for $d = 1, \dots, D'$

- Set $\mathbf{v}_d = \mathbf{e}_d(\Sigma_1)$

We are looking for a new coordinate system $\mathbf{v}_1, \dots, \mathbf{v}_{D'}$, to approximate \mathbf{x}_i :

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx (\mathbf{x}_i \cdot \mathbf{v}_1) \mathbf{v}_1 + (\mathbf{x}_i \cdot \mathbf{v}_2) \mathbf{v}_2 + \dots + (\mathbf{x}_i \cdot \mathbf{v}_{D'}) \mathbf{v}_{D'}$$

where the new axes \mathbf{v}_d 's are all D -dimensional unit norm, and $D' \ll D$



$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx \sum_{d=1}^{D'} (\mathbf{x}_i \cdot \mathbf{v}_d) \mathbf{v}_d$$

So, the new low-dimensional representation is:

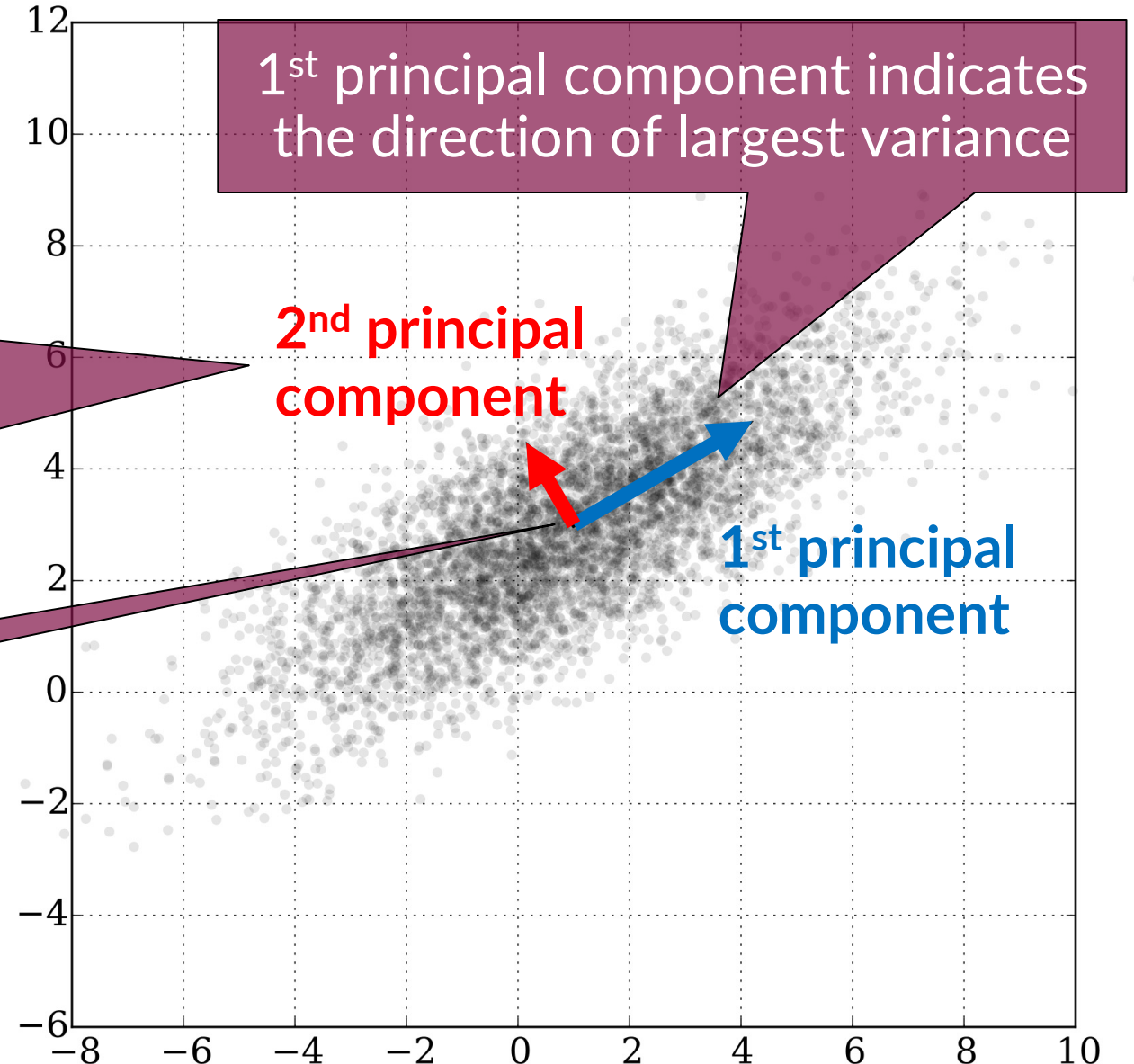
$$f(\mathbf{x}_i) = [\mathbf{x}_i \cdot \mathbf{v}_1, \mathbf{x}_i \cdot \mathbf{v}_2, \dots, \mathbf{x}_i \cdot \mathbf{v}_{D'}]$$

PCA on a 2D Gaussian Dataset

Each subsequent principal component:

- is orthogonal to all previous components
- indicates the direction of largest variance of the residuals

Basis vectors originate from the mean



PCA Algorithm Summary So Far

Given data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, compute covariance matrix Σ

- \mathbf{X} is the $N \times D$ data matrix
- Compute data mean (average over all rows of \mathbf{X})
- Subtract mean from each row of \mathbf{X} (centering the data)
- Compute covariance matrix $\Sigma = \frac{1}{N} \mathbf{X}^T \mathbf{X}$ (Σ is $D \times D$)

PCA **basis vectors (new coordinate axes)** are given by the eigenvectors of Σ

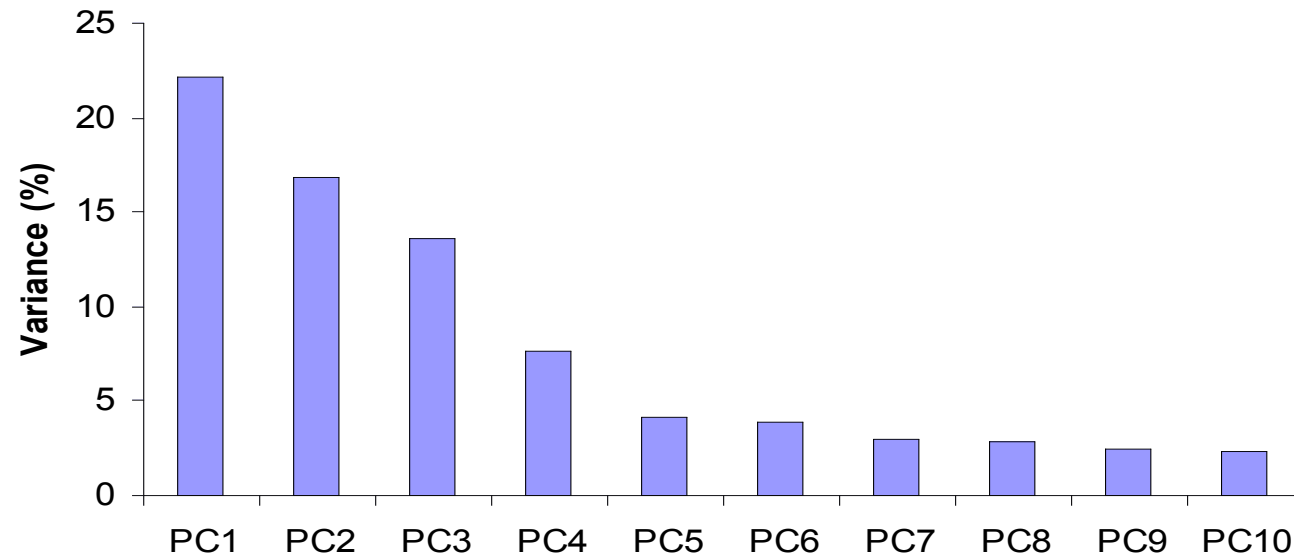
- $U, \Lambda = \text{numpy.linalg.eig}(\Sigma)$
- $\{\mathbf{u}_d, \lambda_d\}_{d=1, \dots, D}$ are the eigenvectors/eigenvalues of Σ
($\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_D$)

But there are D eigenvectors, so where is the dimensionality reduction?

A: Larger eigenvalue \Rightarrow “more important” eigenvectors

Dimensionality Reduction

- Can *ignore* the components of lesser significance



- You do lose some information, but if the eigenvalues are small, you don't lose much
 - choose only the first D' eigenvectors, based on their eigenvalues
 - final data set has only D' dimensions

Recap

- Want to reconstruct data approximately in a new **coordinate space**
- Must find axes of this **coordinate space**, because the **weights** on those axes are just projections
- Objective: **axes** with lowest reconstruction error
 - Same as **axes** with high variance projections
- Solution straight from linear algebra. **Axes** are eigenvectors of covariance matrix

PCA Example

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & \dots \\ 1 & 1 & 0 & 1 & 1 & \dots \\ 0 & 0 & 1 & 1 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 1 & 0 & 1 & \dots \end{bmatrix}$$

X has D columns

(just happens to be binary)

U is the eigenvectors of $\Sigma = X^T X$;
columns are ordered by importance
(highest eigenvalues first)

$$U = \begin{bmatrix} 0.34 & 0.23 & -0.30 & -0.23 & \dots \\ 0.04 & 0.13 & -0.40 & 0.21 & \dots \\ -0.64 & 0.93 & 0.61 & 0.28 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ -0.20 & -0.83 & 0.78 & -0.93 & \dots \end{bmatrix}$$

u_1 u_2 ... u_D

Each row of U corresponds to a feature; keep only first D' columns of U

U is $D \times D$

PCA

- Each column of U gives weights for a linear combination of the original features

$$U = \begin{bmatrix} 0.34 & 0.23 & -0.30 & -0.23 & \cdots \\ 0.04 & 0.13 & -0.40 & 0.21 & \cdots \\ -0.64 & 0.93 & 0.61 & 0.28 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ -0.20 & -0.83 & 0.78 & -0.93 & \cdots \end{bmatrix}$$



$$= 0.34 \times \text{feature}_1 + 0.04 \times \text{feature}_2 - 0.64 \times \text{feature}_3 + \cdots$$

PCA

Compute $\mathbf{x} \cdot \mathbf{e}_d$ to get the new representation for each instance \mathbf{x}

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 1 & 0 & 1 & \cdots \end{bmatrix} \mathbf{x}_3 \quad \hat{U} = \begin{bmatrix} 0.34 & 0.23 \\ 0.04 & 0.13 \\ -0.64 & 0.93 \\ \vdots & \vdots \\ -0.20 & -0.83 \end{bmatrix}$$

The new 2D representation for \mathbf{x}_3 is given by $[\hat{\mathbf{x}}_{31} = \mathbf{x}_3 \cdot \mathbf{u}_1, \hat{\mathbf{x}}_{32} = \mathbf{x}_3 \cdot \mathbf{u}_2]$:

$$\hat{\mathbf{x}}_{31} = 0.34(0) + 0.04(0) - 0.64(1) + \cdots$$

$$\hat{\mathbf{x}}_{32} = 0.23(0) + 0.13(0) + 0.93(1) + \cdots$$

The re-projected data matrix can be conveniently computed as $\hat{X} = X\hat{U}$

Eigenfaces

What happens when you compute the principal components of face images?



(1000 64×64 images)

Eigenfaces

What happens when you compute the principal components of face images?

“Eigenfaces”: main directions of deviation from the mean face



Figure #5: mean face



Eigenfaces

Let's try reconstructing these faces with the eigenfaces now!



(1000 64×64 images)

Eigenfaces

... with 1000 eigenvectors



Eigenfaces

... with 250 eigenvectors



Eigenfaces

... with 100 eigenvectors



Eigenfaces

... with 50 eigenvectors



PCA Visualization of Digits

3	6	8	1	7	9	6	6	4	1
6	7	5	7	8	6	3	4	8	5
2	1	7	9	7	1	2	1	4	5
4	8	1	9	0	1	8	3	9	4
7	6	1	8	6	4	1	5	6	0
7	5	9	2	6	5	8	1	9	7
2	2	2	2	2	3	9	4	3	0
0	2	3	8	0	7	3	8	5	7
0	1	4	6	4	6	0	2	4	8
7	1	2	3	7	6	9	8	6	1

PCA

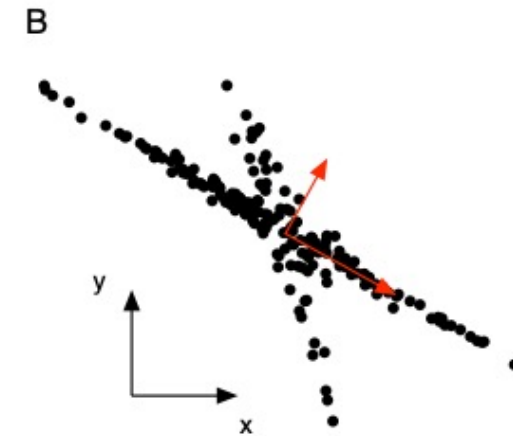


Utility of PCA

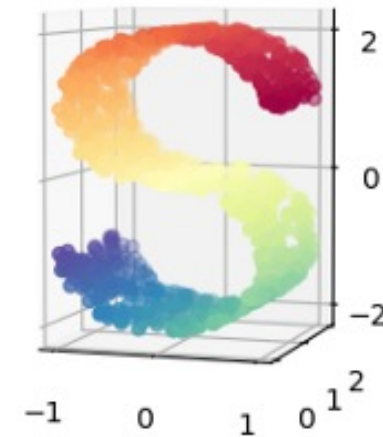
- PCA is often used as a preprocessing step for supervised learning
 - reduces dimensionality
 - eliminates redundant features (i.e. linearly dependent features)
- Can also be used to aid in visualization

PCA Doesn't Always Work Well

- Here, principal components in red don't capture the main directions in the data.
- In general, PCA is not guaranteed to recover semantically aligned features from the data.
- The true data “shape” might not be captured by a simple linear projection of the original data.

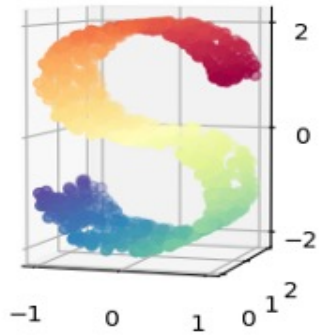


Shlens 2014, A Tutorial on PCA

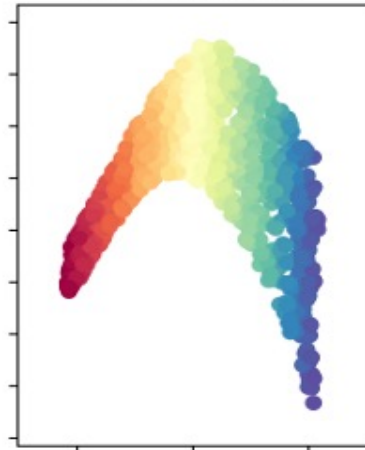


Beyond PCA: Non-linear dimensionality reduction

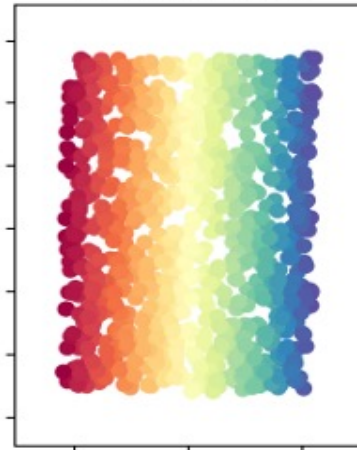
Manifold Learning with 1000 points, 10 neighbors



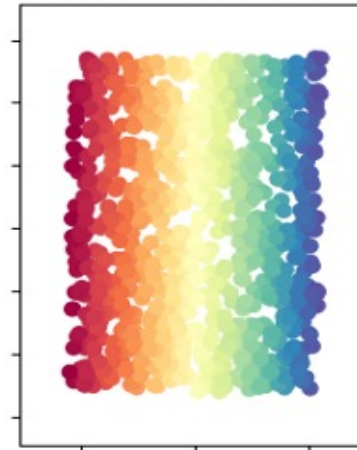
LLE (0.11 sec)



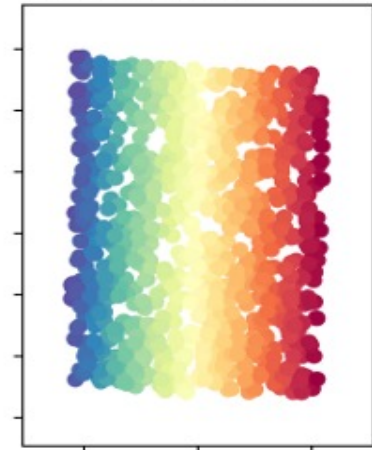
LTSA (0.19 sec)



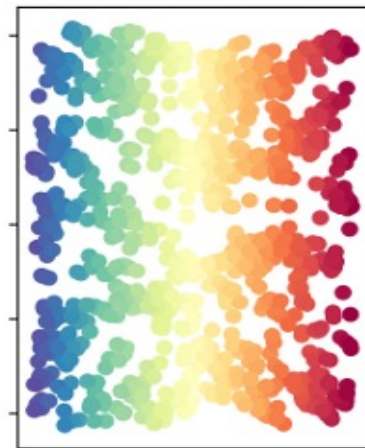
Hessian LLE (0.37 sec)



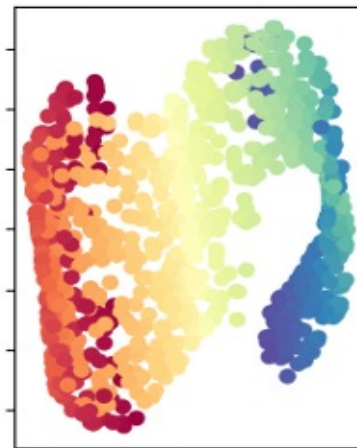
Modified LLE (0.22 sec)



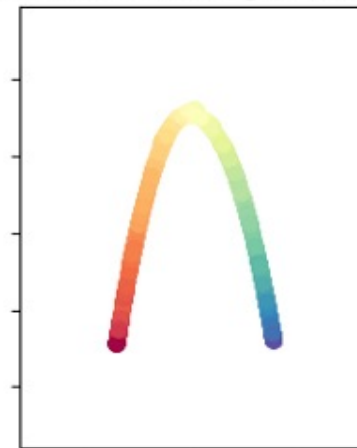
Isomap (0.34 sec)



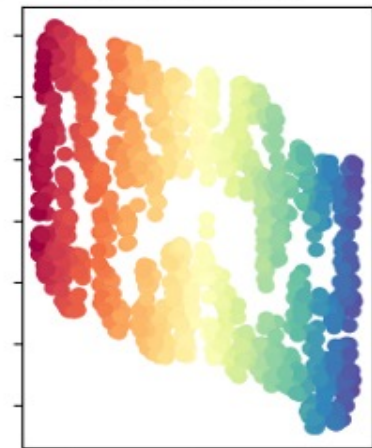
MDS (2.5 sec)



SpectralEmbedding (0.16 sec)



t-SNE (5.8 sec)



Beyond PCA: Non-linear dimensionality reduction



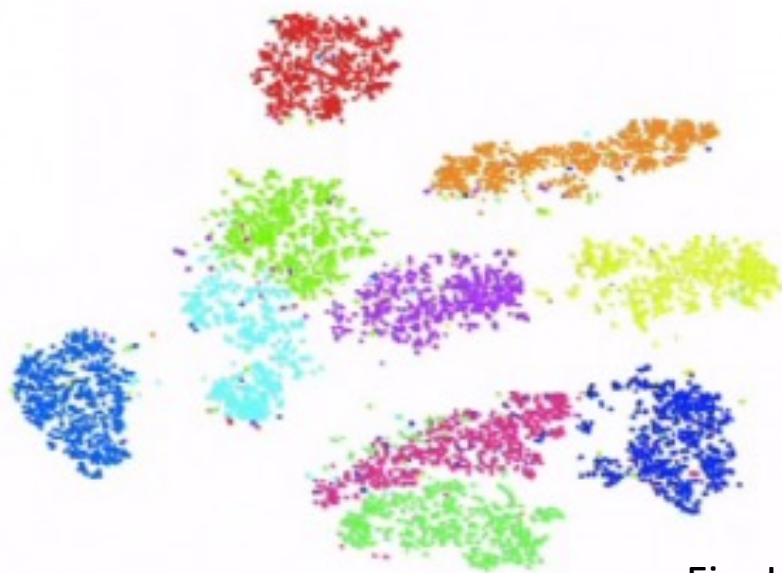
PCA



ISOMAP



T-SNE



T-SNE and ISOMAP are popular and powerful nonlinear approaches, but:

- Require careful hyperparameter tuning
- Harder to optimize
- Not as easy to interpret, no easy projection back to original data

Recap: Unsupervised Learning

Basic idea: reduce feature space to a much lower set of dimensions

- Clustering: find structural similarity, return one k-valued higher-level feature
- PCA: find orthonormal dimensions in order of most to least variance
- Can be useful for human inspection (visualization) as well as supervised ML