Dimensionality Reduction

Curse of Dimensionality

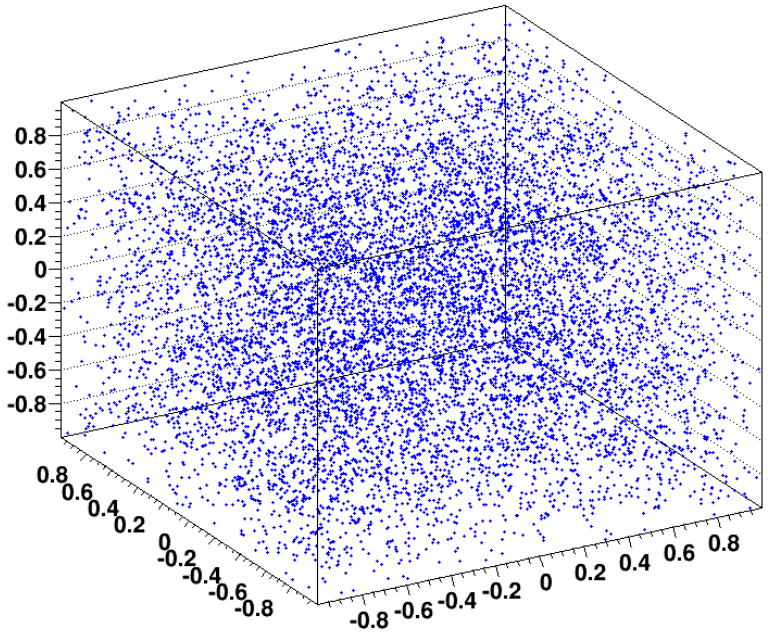
[Wikipedia] The *curse of dimensionality* (Bellman, 1961) refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience.

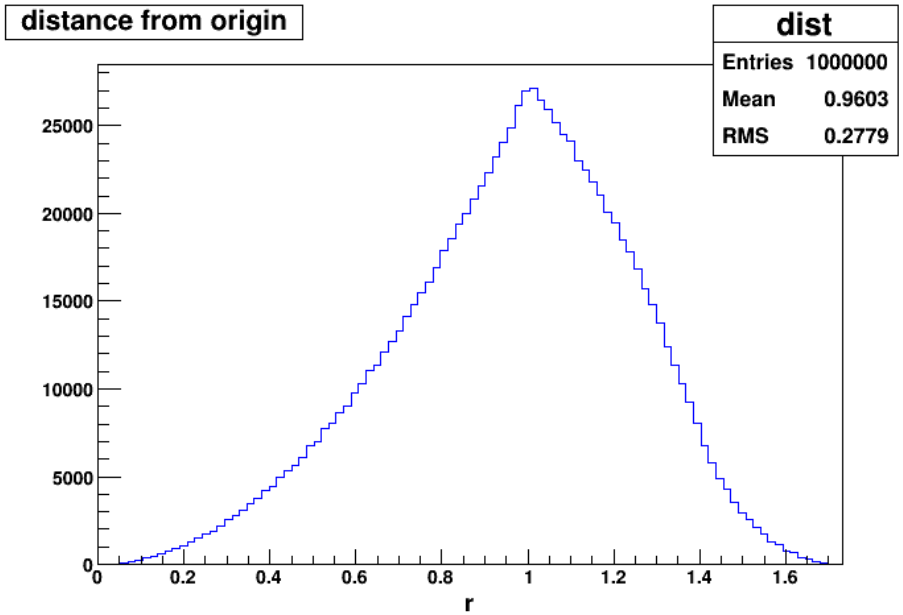
- Counterintuitive geometry of hyperspace!
- Sparsity of sample data points!

Distance in Hyperspace

Distance between nearest neighbors is very large!

Simple simulation: uniformly distributed points in a cube.

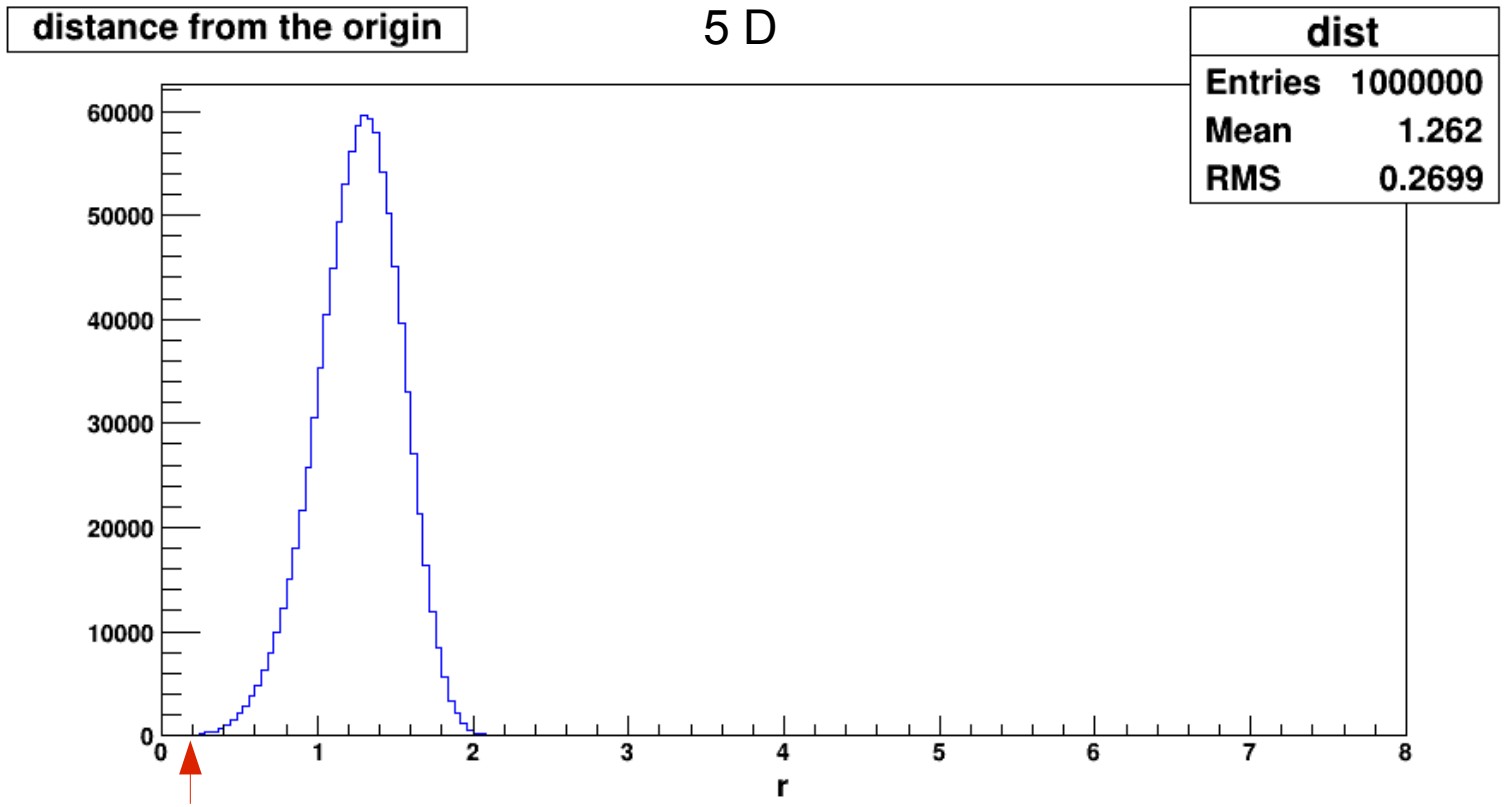




Distance in Hyperspace

Distance between nearest neighbors is very large!

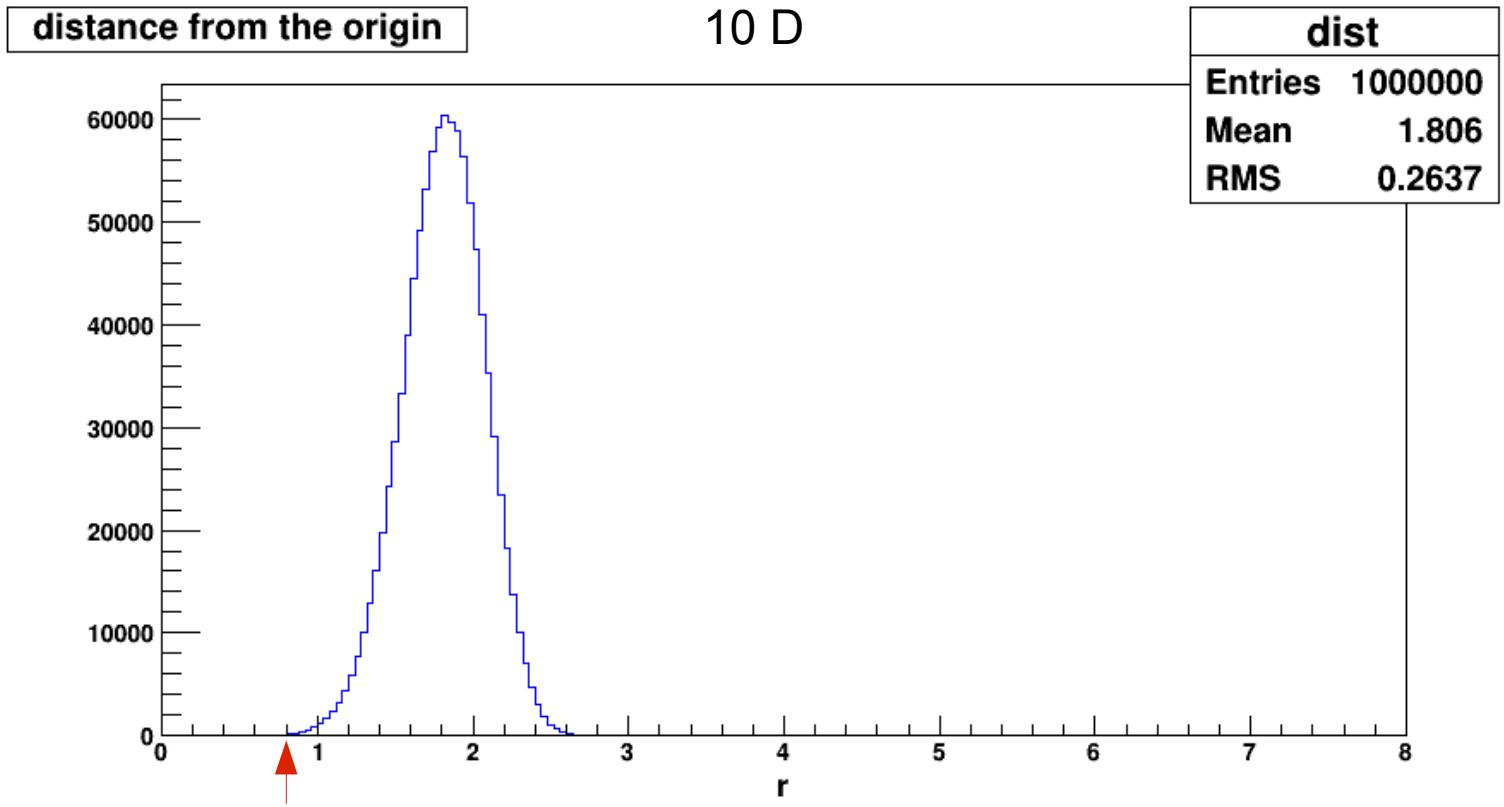
Simple simulation: uniformly distributed points in a hypercube.



Distance in Hyperspace

Distance between nearest neighbors is very large!

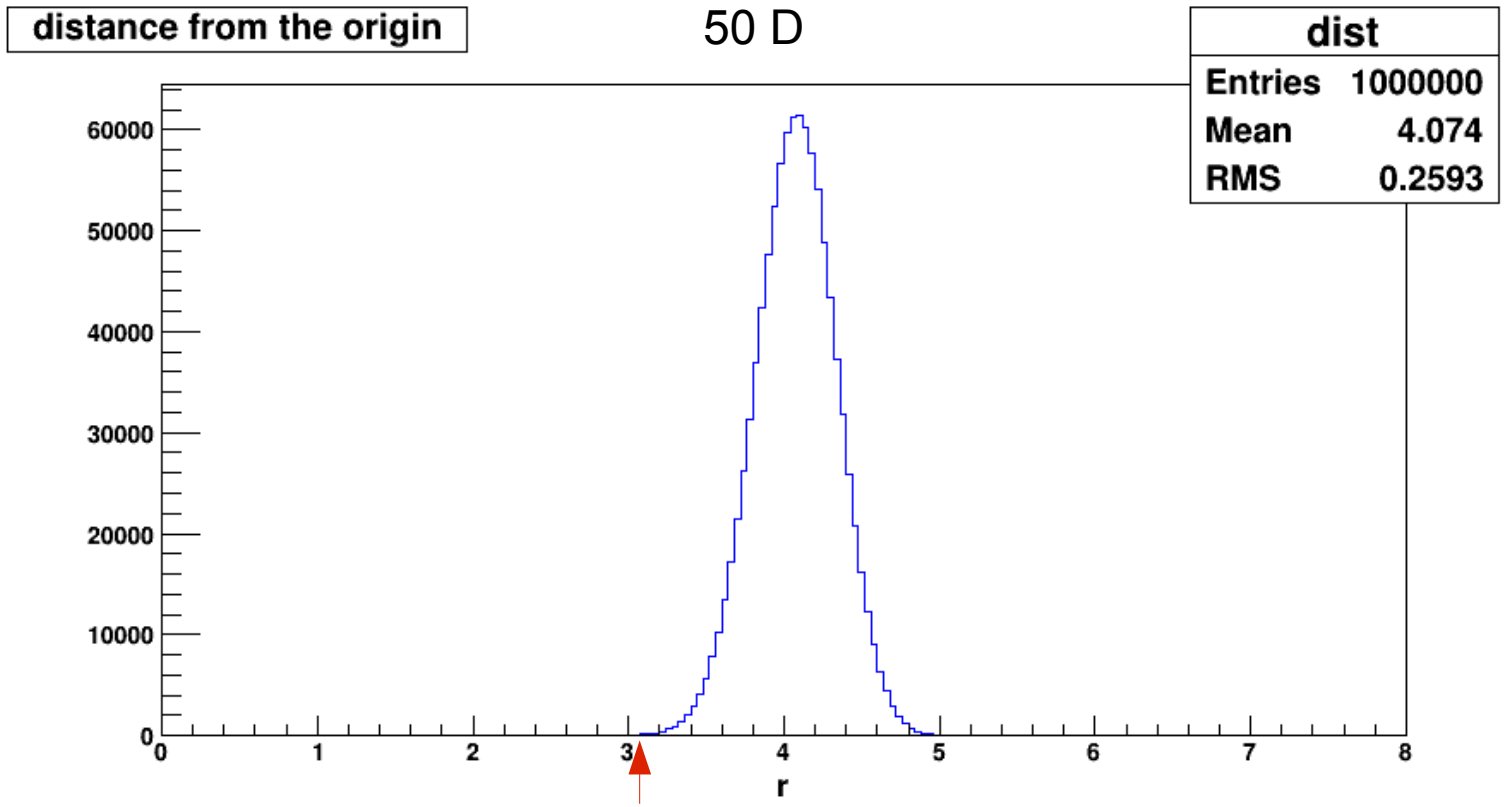
Simple simulation: uniformly distributed points in a hypercube.



Distance in Hyperspace

Distance between nearest neighbors is very large!

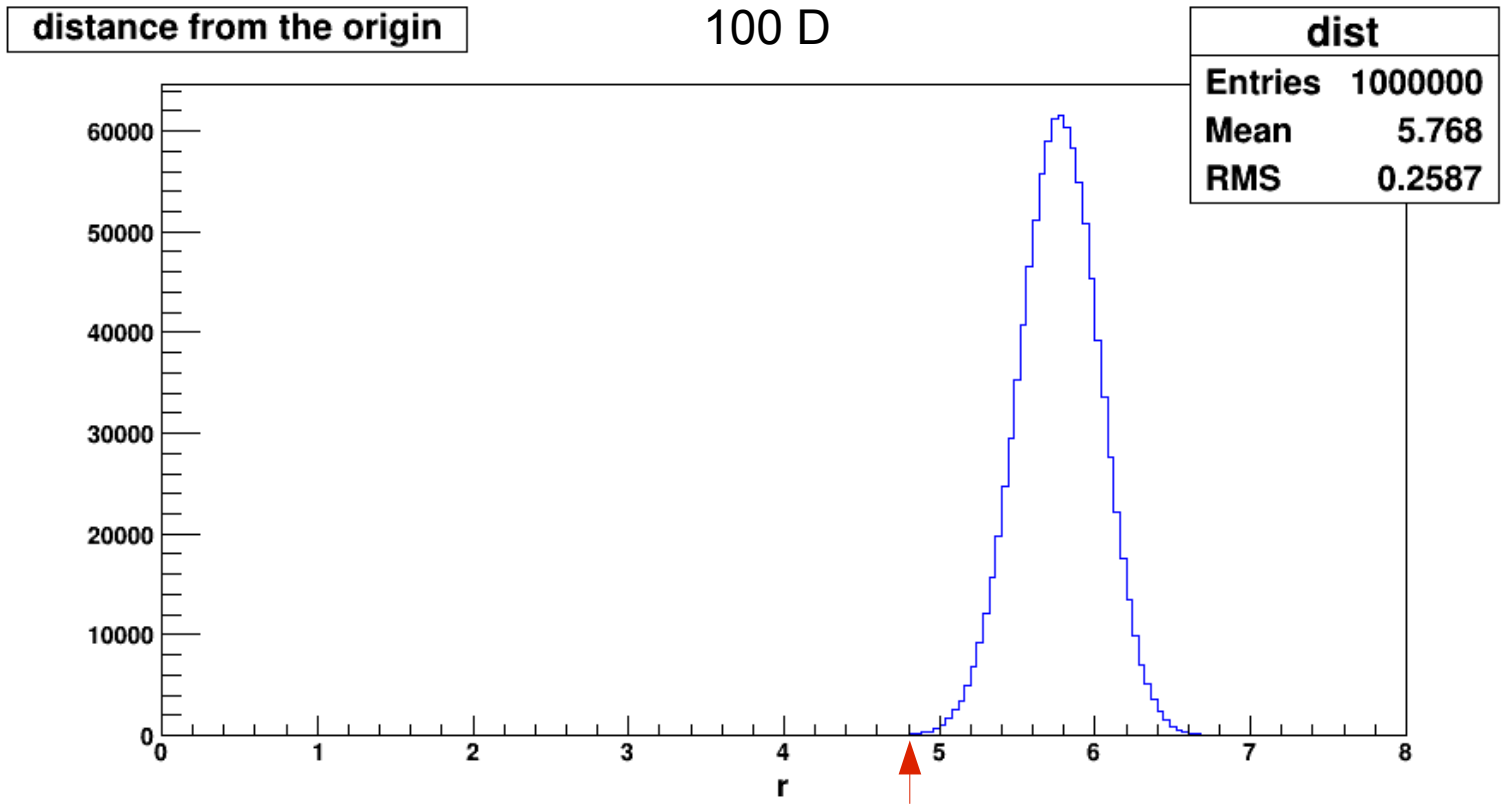
Simple simulation: uniformly distributed points in a hypercube.



Distance in Hyperspace

Distance between nearest neighbors is very large!

Simple simulation: uniformly distributed points in a hypercube.

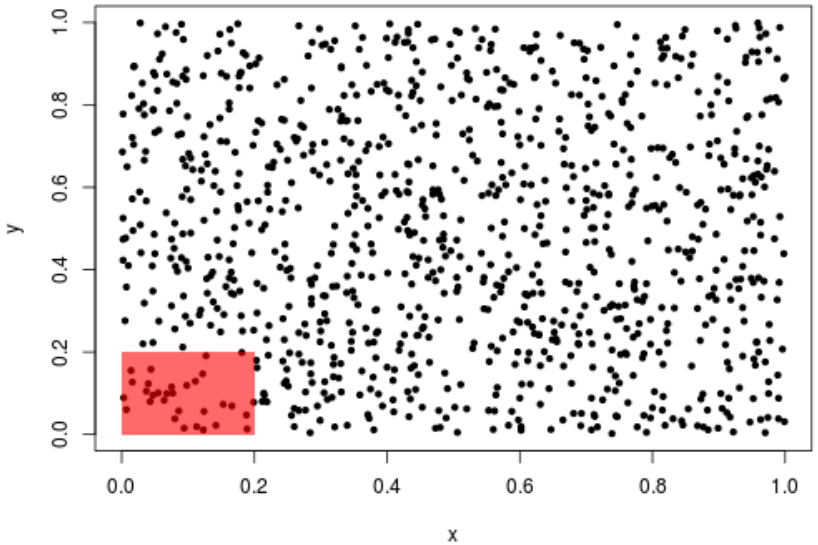


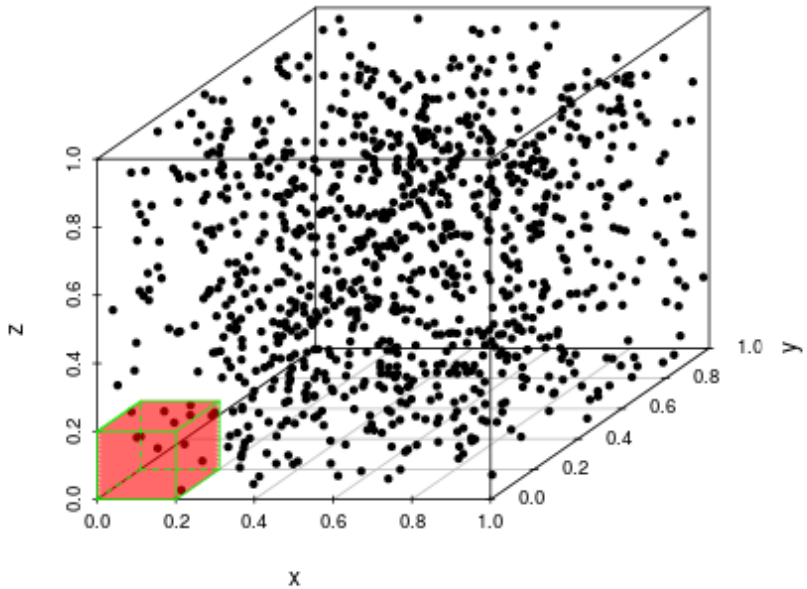
Most data points are closer to the boundary of the sample space.

Sample Sparsity

When the dimensionality d increases, the volume of the space increases so fast that the available data becomes sparse.

Consider uniformly distributed data points. N = 1000
Cover 20% of range of each feather.





Fraction of data points captured:

2D : 3.1%

3D : 0.5%

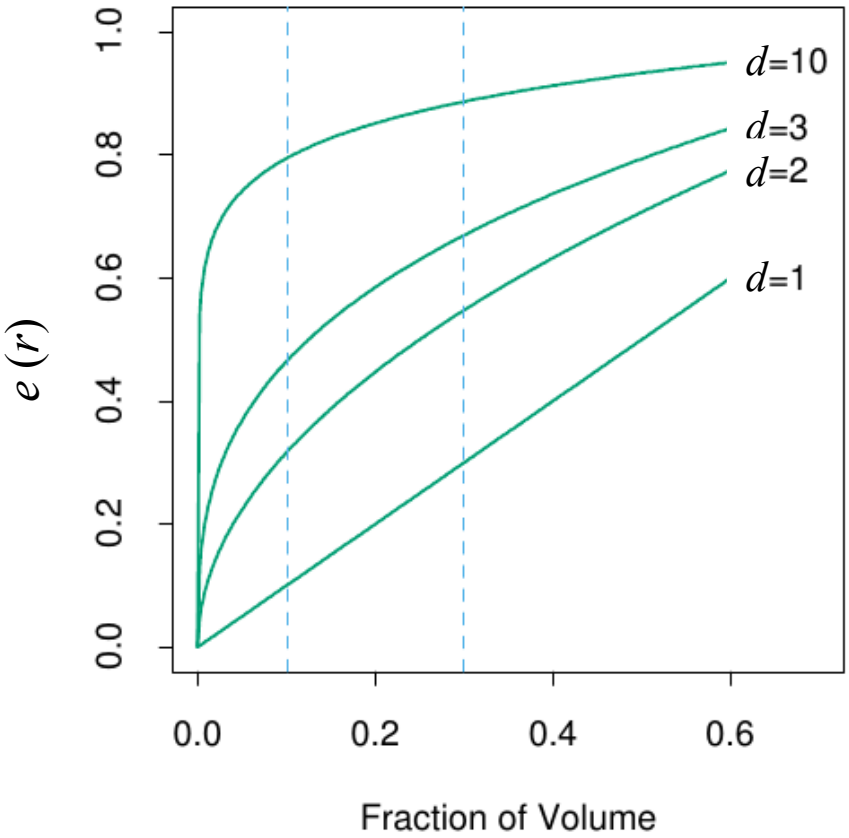
Sample Sparsity

For fraction r of unit volume:

Edge length: $e(r) = r^{1/d}$

In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.

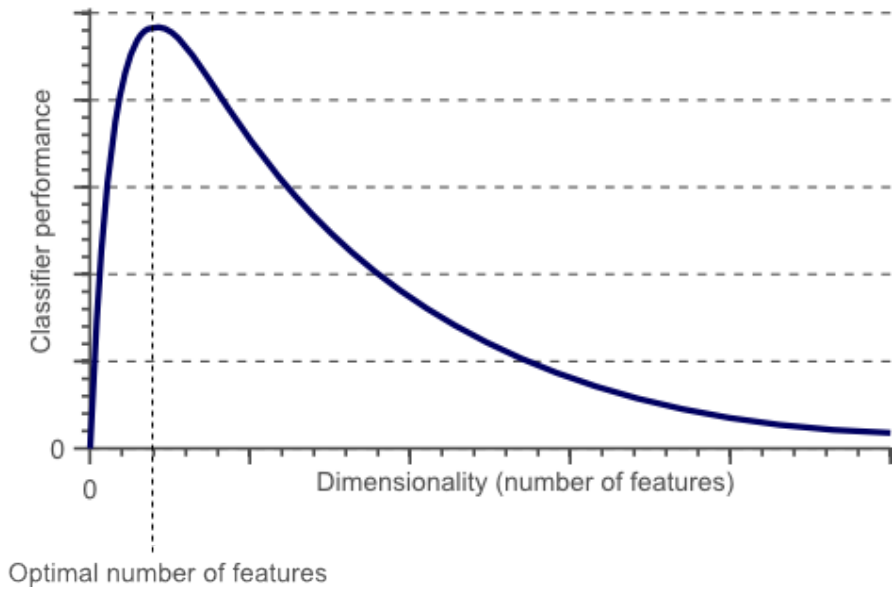
Reducing r gives fewer observations to average - higher variance of fit.



Classifier Performance

As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached.

Increasing the dimensionality further without increasing the number of training samples results in a decrease in classifier performance.



Data Matrix

$$\mathbf{D} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}$$

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

$\mathbf{x}_i^T$ are dimensional row vectors.

$$\mathbf{D} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_d \end{bmatrix}$$

$\mathbf{y}_j$ are column vectors.

	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4
$\mathbf{x}_1$	5.9	3.0	4.2	1.5
$\mathbf{x}_2$	6.9	3.1	4.9	1.5
$\mathbf{x}_3$	6.6	2.9	4.6	1.3
$\mathbf{x}_4$	4.6	3.2	1.4	0.2
$\mathbf{x}_5$	6.0	2.2	4.0	1.0
$\mathbf{x}_6$	4.7	3.2	1.3	0.2
$\mathbf{x}_7$	6.5	3.0	5.8	2.2
$\mathbf{x}_8$	5.8	2.7	5.1	1.9
⋮	⋮	⋮	⋮	⋮
$\mathbf{x}_{149}$	7.7	3.8	6.7	2.2
$\mathbf{x}_{150}$	5.1	3.4	1.5	0.2

Extract from Iris data set.

Rank and Dimensionality

Linear Independence

For any set of n vectors, a linear combination is an expression of the form:

$$c_1\mathbf{y}_1+c_2\mathbf{y}_2+\cdots+c_n\mathbf{y}_n$$

Now consider an equation:

$$c_1\mathbf{y}_1+c_2\mathbf{y}_2+\cdots+c_n\mathbf{y}_n=\mathbf{0}$$

If this equation holds only if all c_i's are zero, then y_i's are *linearly independent* vectors.

If the equation holds with c_i's not equal to zero, then the vectors are *linearly dependent*. This means, we can express at least one of the vectors as linear combination of the others. *e.g.*,

$$\mathbf{y}_1=k_2\mathbf{y}_2+\cdots+k_n\mathbf{y}_n \qquad \text{where } k_j=-c_j/c_1$$

Rank and Dimensionality

The *rank* of a matrix **D**
Maximum number of linearly independent column vectors of **D**
Maximum number of linearly independent row vectors of **D**.

$$\mathbf{D} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_d \end{bmatrix}$$

D is a $[n \times d]$ matrix. Then, the $\text{rank}(\mathbf{D}) = r \leq \min(n, d)$

Rank of data matrix gives the dimensionality of the data.

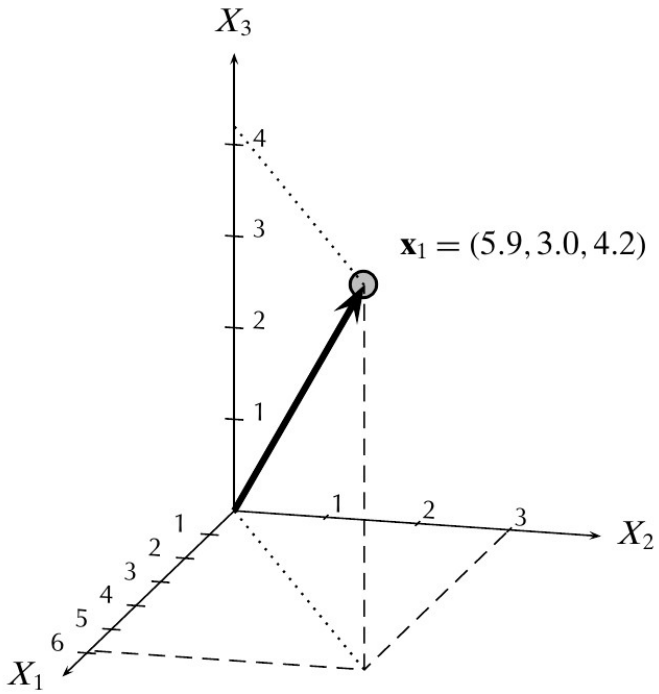
Rank and Dimensionality

The number of linearly independent *basis vectors* needed to represent a data vector, gives the dimensionality of the data. *e.g.*,

$$\mathbf{x} = x_1 \mathbf{e}_1 + x_2 \mathbf{e}_2 + \cdots + x_d \mathbf{e}_d$$

The data points apparently reside in a d-dimensional *attribute space*.

But, if $r < d$, then the data points actually reside in a lower r-dimensional space.



3 dimensional space

Dimensionality Reduction

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

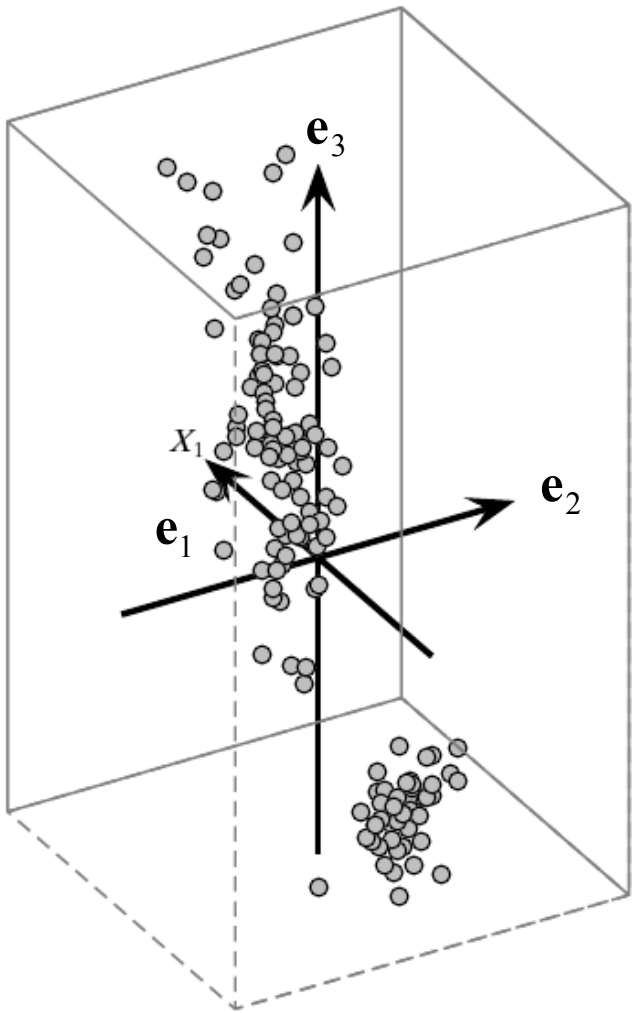
Each point $\mathbf{x}_i^T = (x_1, x_2, \ldots x_d)^T$ is a vector in d-dimensional vector space.

We can write $\mathbf{x}$ as

$$\mathbf{x} = \sum_{i=1}^{d} x_i \mathbf{e}_i$$

where $\mathbf{e}_i$ are orthonormal basis vectors:

$$\mathbf{e}_i^T \mathbf{e}_j = 1 \quad \text{if } i = j$$
$$= 0 \quad \text{if } i \neq j$$



Dimensionality Reduction

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

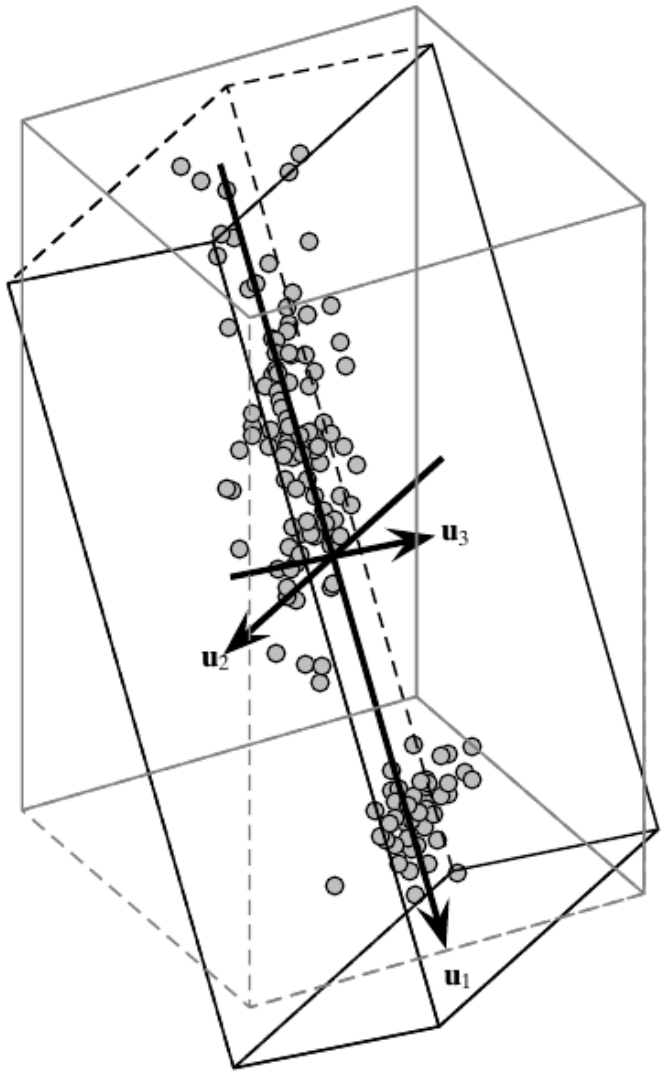
Each point $\mathbf{x}_i^T = (x_{1,} x_{2,} \ldots x_d)^T$ is a vector in d-dimensional vector space.

Given any other set of d orthonormal vectors:

$$\mathbf{x} = \sum_{i=1}^{d} a_i \mathbf{u}_i$$

where $\mathbf{u}_i$ are orthonormal basis vectors, and

$$\mathbf{u}_i^T \mathbf{u}_j = 1 \quad \text{if } i = j$$
$$= 0 \quad \text{if } i \neq j$$



Dimensionality Reduction

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix}$$

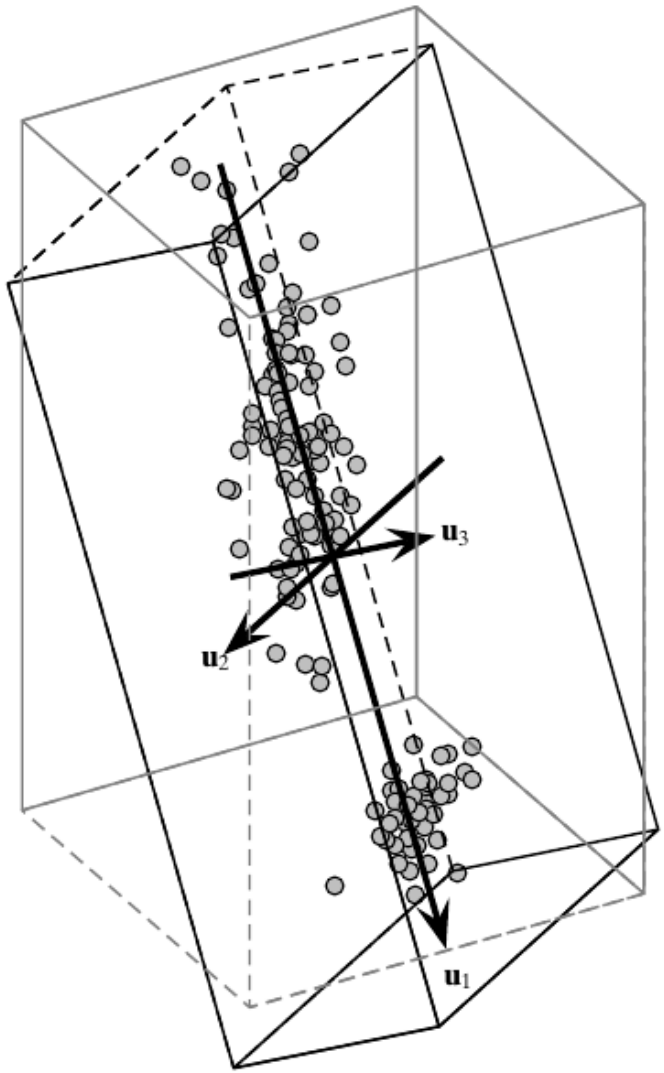
Each point $\mathbf{a}_i^T = (a_{1,} a_{2,} \ldots a_d)^T$ is a vector in d-dimensional vector space.

$$a_j = \mathbf{u}_j^T \mathbf{x}$$

In vector form:

$$\mathbf{a} = \mathbf{U}^T \mathbf{x}$$

$$\mathbf{U} = (\mathbf{u}_1 \; \mathbf{u}_2 \; \ldots \; \mathbf{u}_d)$$



Dimensionality Reduction

Because there are potentially infinite choices for the set of orthonormal basis vectors, one natural question is whether there exists an *optimal* basis, for a suitable notion of optimality.

Can we find a reduced dimensionality subspace that still preserves the essential characteristics of the data.

Basic idea is to project data points from a d dimensional space to an r dimensional space where $r < d$.

$$\mathbf{x}' = \sum_{i=1}^{r} a_i \mathbf{u}_i$$

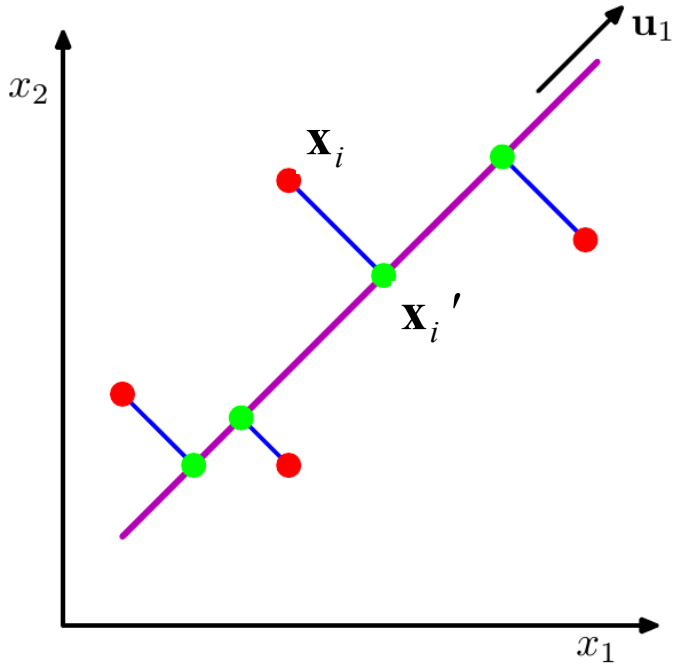
$$\boldsymbol{\epsilon} = \sum_{i=r+1}^{d} a_i \mathbf{u}_i = \mathbf{x} - \mathbf{x}' \qquad \textit{error vector}$$

Principle Component Analysis

Principal Component Analysis (PCA) is a technique that seeks a r-dimensional basis that best captures the variance in the data.

The direction with the largest projected variance is called the first principal component.

The orthogonal direction that captures the second largest projected variance is called the second principal component, and so on.



Principle Component Analysis

First Principle Component: We have to choose the direction $\mathbf{u}$ such that the variance of the projected points is maximized.

The projected variance along $\mathbf{u}$ is:

$$\sigma_{\mathbf{u}}^2 = \frac{1}{n}\sum_{i=1}^{n}(a_i - \mu_{\mathbf{u}})^2$$

For centered data:

$$\sigma_{\mathbf{u}}^2 = \mathbf{u}^T \Sigma \mathbf{u}$$

$\mathbf{\Sigma}$ is covariance matrix of centered $\mathbf{D}$.

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

Principle Component Analysis

Maximizing σ (with constraint $\mathbf{u}^T\mathbf{u} = 1$) gives:

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

$$\sigma_{\mathbf{u}}^2 = \lambda$$

To maximize projected variance, we maximize the eigenvalue of Σ

Eigenvector $\mathbf{u}$ with maximum λ specifies the direction of most variance, also called the *first principal component.*

Principle Component Analysis

To find the best r-dim approximation to $\mathbf{D}$, we compute the eigenvalues of the covariance matrix Σ.

Eigenvalues of Σ are non-negative, and we can sort them in decreasing order:

$$\lambda_1 \geq \lambda_2 \geq \ldots \lambda_r \geq \lambda_{r+1} \ldots \geq \lambda_d \geq 0$$

Eigenvector corresponding to λ_1 gives first principle component, eigenvector corresponding to λ_2 gives the second principle component, and so on.

Principle Component Analysis

Reduced r-dimensional data matrix is then:

$$\mathbf{A} = [\boldsymbol{\alpha}_1 \quad \boldsymbol{\alpha}_2 \quad \ldots \quad \boldsymbol{\alpha}_r]$$

Total variance of $\mathbf{A}$:

$$var(\mathbf{A}) = \sum_{i=1}^{r} \lambda_i$$

The first r-principal components maximize the projected variance $var(\mathbf{A})$ and thus they also minimize the MSE.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\epsilon}_i^T \boldsymbol{\epsilon}_i \qquad \boldsymbol{\epsilon}_i = \mathbf{x}_i - \mathbf{x}_i'$$

$$= var(\mathbf{D}) - var(\mathbf{A})$$

Principle Component Analysis

Choosing the dimensionality: how many dimensions, r, to use for a good approximation.

Compute the fraction of the total variance captured by the first r principal components:

$$f(r) = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_r}{\lambda_1 + \lambda_2 + \ldots + \lambda_d} = \frac{var(\mathbf{A})}{var(\mathbf{D})}$$

Starting from the first principal component, then keep on adding additional components, and stop at the smallest value r, for which $f(r) \geq \alpha$.

In practice, α is usually set to 0.9 or higher, so that the reduced dataset captures at least 90% of the total variance.

PCA for $n < d$

The covariance matrix is a $d \times d$ matrix.

Typical algorithms for finding eigenvectors of $d \times d$ matrix have a computational cost that scales like $O(d^3)$.

For $n<d$, there are only n non-zero eigenvalues of covariance matrix.

Starting from the eigenvalue equation:

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

We can write an eigenvalue equation:

$$\frac{1}{n}\mathbf{D}\mathbf{D}^{\mathrm{T}}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

where $\mathbf{v}_i = \mathbf{D}\mathbf{u}_i$.
This is an eigenvalue equation of $n \times n$ matrix!

$$\mathbf{u}_i = \frac{1}{\sqrt{n\lambda_i}}\mathbf{D}^T\mathbf{v}_i$$

In summary, to apply this approach we first evaluate $\mathbf{D}\mathbf{D}^{\mathrm{T}}$ and then find its eigenvectors and eigenvalues, and then compute the eigenvectors in the original data space.

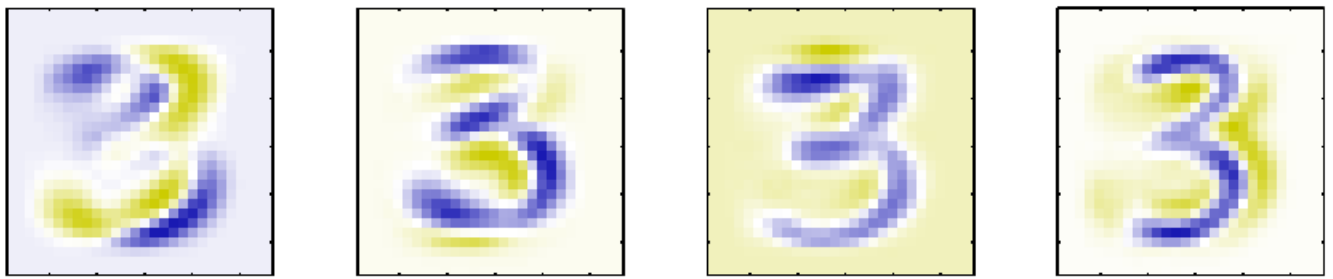
Application of PCA

PCA for Image Compression: Handwritten digits from MNIST data set.

Each digit is centered in 28×28 pixel box.
So,each digit can be represented as a vector in 784-dimensional vector space.

Each eigenvector of the covariance matrix is a vector in in the original d-dimensional space, we can represent the eigenvectors as images of the same size as the data points.

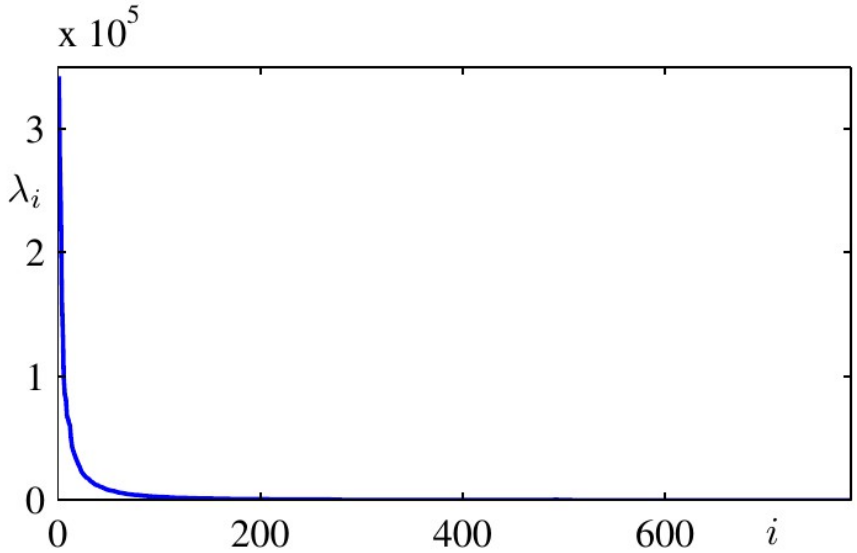
$\lambda_1 = 3.4 \cdot 10^5$ $\lambda_2 = 2.8 \cdot 10^5$ $\lambda_3 = 2.4 \cdot 10^5$ $\lambda_4 = 1.6 \cdot 10^5$



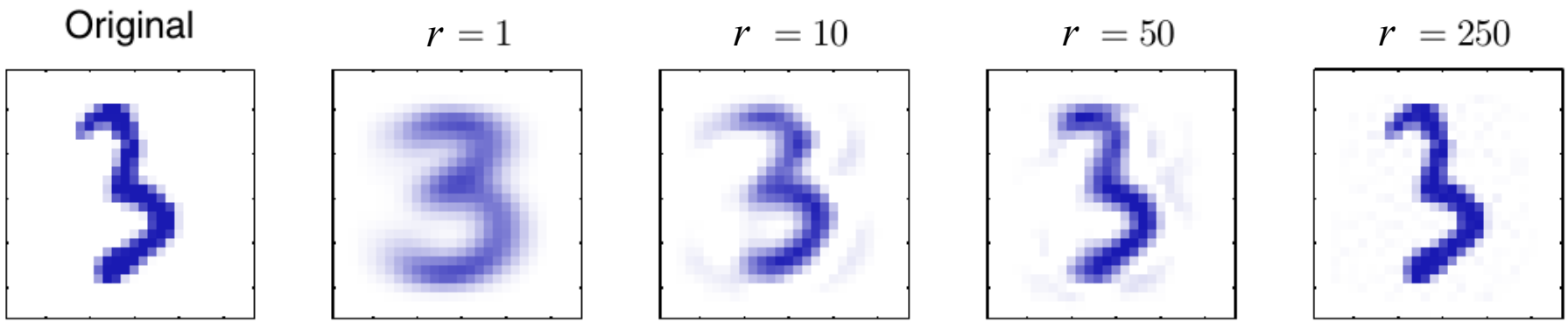
$\mathbf{u}_1$ $\mathbf{u}_2$ $\mathbf{u}_3$ $\mathbf{u}_4$

Application of PCA

Plot of the complete spectrum of eigenvalues, sorted into decreasing order.



An original example from the digits data set together with its PCA reconstructions obtained by retaining r principal components for various values of r.



Singular Value Decomposition

PCA yields the following decomposition of covariance matrix:

$$\mathbf{\Sigma} = \mathbf{U}\,\mathbf{\Lambda}\,\mathbf{U}^T$$

$$\sigma_{\mathbf{i}}^2 = \lambda_i = \mathbf{u}_i^T\,\Sigma\,\mathbf{u}_i$$

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix}$$

$\mathbf{\Lambda}$ is the covariance matrix in new basis.

Singular Value Decomposition

PCA is a special case of more general matrix decomposition:
Singular Value Decomposition (*SVD).*

An $n \times d$ data matrix $\mathbf{D}$ can be factorized as:

$$\mathbf{D} = \mathbf{L}\,\mathbf{\Delta}\,\mathbf{R}^T$$

$$\mathbf{\Delta}(i,j) = \begin{cases} \delta_i & \text{If } i = j \\ 0 & \text{If } i \neq j \end{cases}$$

$\mathbf{L}$: $n \times n$ *left singular matrix*
$\mathbf{R}$: $d \times d$ *right singular matrix*
$\mathbf{\Delta}$: $n \times d$ *diagonal matrix*

$i = 1, \ldots. n$
$j = 1, \ldots. d$
δ_i : *singular values*

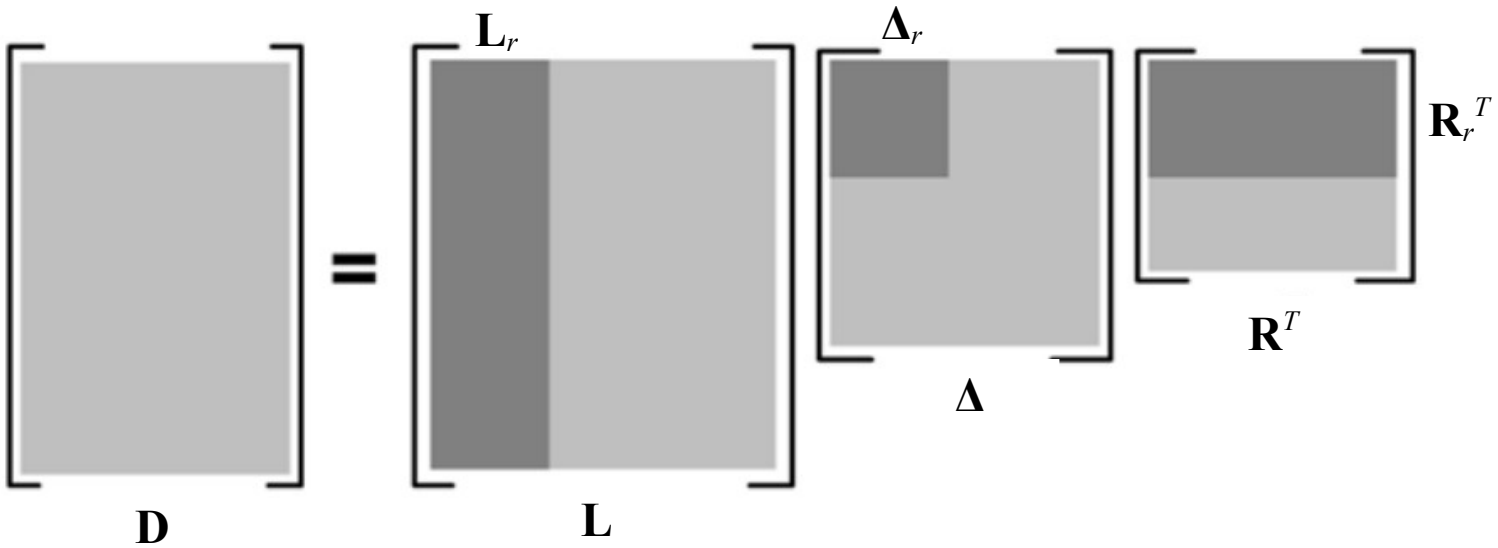
Singular Value Decomposition

If the rank of **D** is $r \leq \min(n, d)$, then there will be only r nonzero singular values:

$$\delta_1 \geq \delta_2 \geq \ldots \geq \delta_r \geq 0$$

One can discard those *left* and *right* singular vectors that correspond to zero singular values, to obtain the reduced *SVD* as:

$$\mathbf{D}_r = \mathbf{L}_r \boldsymbol{\Delta}_r \mathbf{R}_r^T$$



Connection Between PCA and SVD

PCA gives:

$$\mathbf{D}^T\mathbf{D} = n\boldsymbol{\Sigma}$$

$$= \mathbf{U}(n\boldsymbol{\Lambda})\mathbf{U}^T$$

SVD gives:

$$\mathbf{D}^T\mathbf{D} = (\mathbf{R}\boldsymbol{\Delta}\mathbf{L}^T)^T(\mathbf{R}\boldsymbol{\Delta}\mathbf{L}^T)$$

$$= \mathbf{R}\boldsymbol{\Delta}_d^2\mathbf{R}^T$$

where $\boldsymbol{\Delta}_d^2$ is the $d \times d$ diagonal matrix defined as

$$\boldsymbol{\Delta}_d^2(i,i) = \delta_i^2$$

Comparing both: $\delta_i^2 = n\lambda_i$

$$\mathbf{R} = \mathbf{U}$$

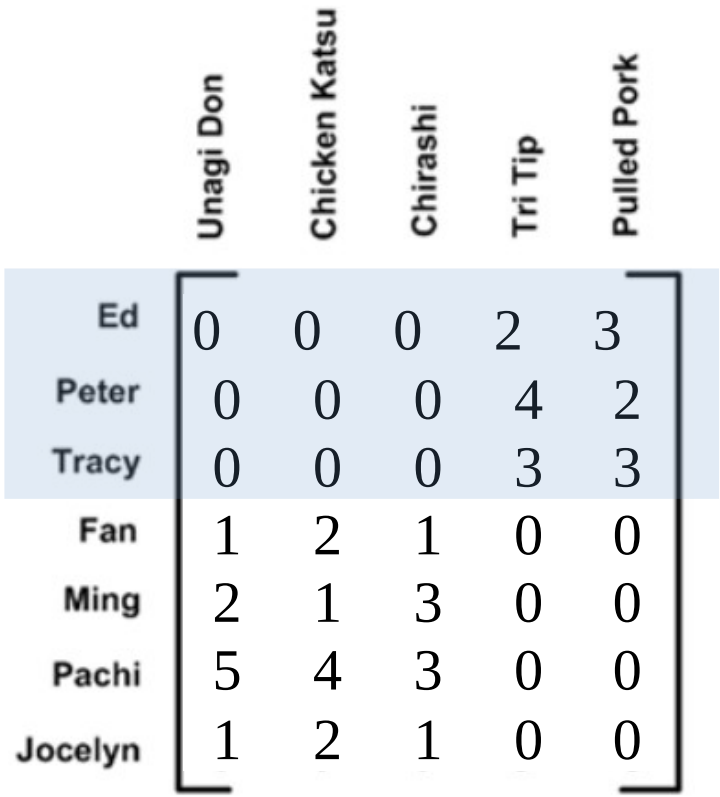
The right singular vectors in $\mathbf{R}$ are the same as eigenvectors of $\boldsymbol{\Sigma}$.

The left singular vectors in $\mathbf{L}$ are the eigenvectors of the matrix $n \times n$ matrix $\mathbf{D}\mathbf{D}^T$, and the corresponding eigenvalues are given as δ_i^2.

Finding Topics/Concepts

SVD can be used to find latent topics in the data.

	Unagi Don	Chicken Katsu	Chirashi	Tri Tip	Pulled Pork
Ed	0	0	0	2	3
Peter	0	0	0	4	2
Tracy	0	0	0	3	3
Fan	1	2	1	0	0
Ming	2	1	3	0	0
Pachi	5	4	3	0	0
Jocelyn	1	2	1	0	0



Finding Topics/Concepts

$\Delta =$

```
array([[ 8.45,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  6.98,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  1.83,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  1.5 ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  1.1 ],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  0.  ,  0.  ]])
```

	Unagi Don	Chicken Katsu	Chirashi	Tri Tip	Pulled Pork
Ed	0	0	0	2	3
Peter	0	0	0	4	2
Tracy	0	0	0	3	3
Fan	1	2	1	0	0
Ming	2	1	3	0	0
Pachi	5	4	3	0	0
Jocelyn	1	2	1	0	0

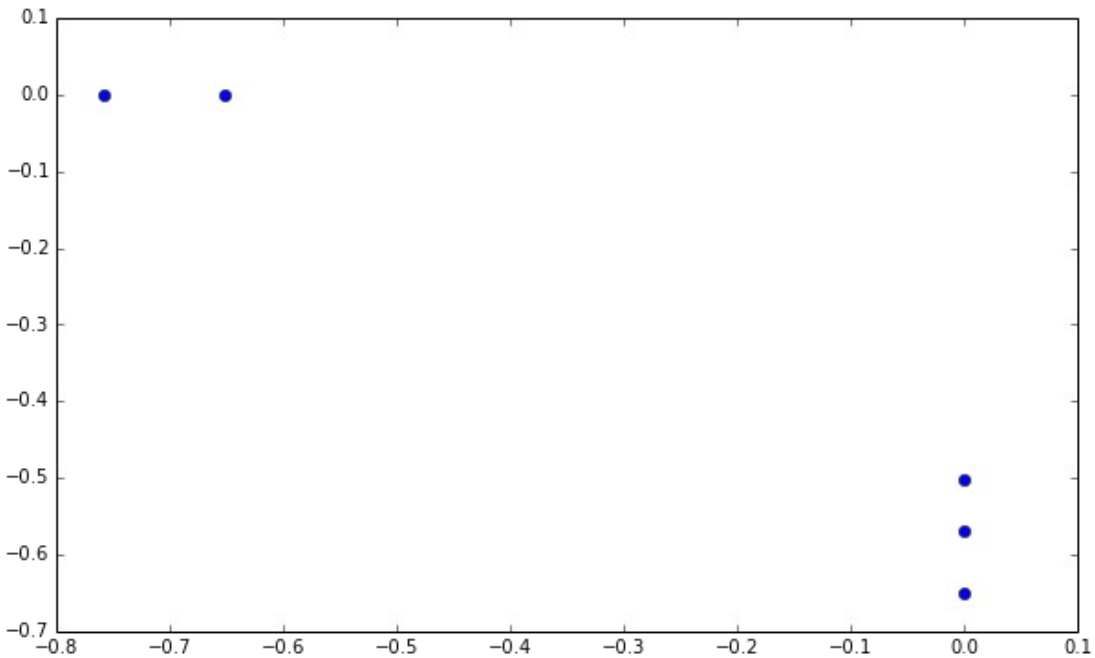
$\mathbf{R}^T =$

```
array([[ -6.51e-01,  -5.70e-01,  -5.01e-01,  -3.55e-17,  -2.47e-17],
       [  2.14e-17,   4.32e-16,  -4.33e-16,  -7.58e-01,  -6.52e-01],
       [  3.91e-02,  -6.85e-01,   7.27e-01,  -8.00e-16,   5.97e-19],
       [ -1.55e-16,  -2.57e-16,   4.84e-16,   6.52e-01,  -7.58e-01],
       [ -7.58e-01,   4.54e-01,   4.68e-01,  -1.66e-16,   1.58e-16]])
```

Finding Topics/Concepts

$\mathbf{R}_2 =$

```
array([[ -6.51e-01,   2.14e-17],
       [ -5.70e-01,   4.32e-16],
       [ -5.01e-01,  -4.33e-16],
       [ -3.55e-17,  -7.58e-01],
       [ -2.47e-17,  -6.52e-01]])
```



	Unagi Don	Chicken Katsu	Chirashi	Tri Tip	Pulled Pork
Ed	0	0	0	2	3
Peter	0	0	0	4	2
Tracy	0	0	0	3	3
Fan	1	2	1	0	0
Ming	2	1	3	0	0
Pachi	5	4	3	0	0
Jocelyn	1	2	1	0	0

Multidimensional Scaling

Distance Matrix or Proximity Matrix

$$\Delta = \begin{bmatrix} \delta_{11} & \delta_{12} & \cdots & \delta_{1n} \\ \delta_{21} & \delta_{22} & \cdots & \delta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \cdots & \delta_{nn} \end{bmatrix}$$

$$\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$$

Euclidean distance between two vectors

MDS is a process used to find lower dimensional representation that give the same distance between points.

Multidimensional Scaling

The following steps summarize the algorithm of MDS

Calculate the matrix of squared proximities: $\boldsymbol{\Delta}^2$

Calculate the matrix $\mathbf{B} = \frac{1}{2}\mathbf{J}\boldsymbol{\Delta}^2\mathbf{J}$ $\qquad$ $\mathbf{J} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$

Obtain SVD of $\mathbf{B}$

$$\mathbf{B} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$$

For a 2-*dim* representation, keep two eigenvectors corresponding to the largest eigenvalues.