

# Data Preprocessing

Jilles Vreeken



IRDM '15/16

22 Oct 2015



UNIVERSITÄT  
DES  
SAARLANDES



**mpi** max planck institut  
informatik



So, how do you pronounce...

Jilles

Vreeken

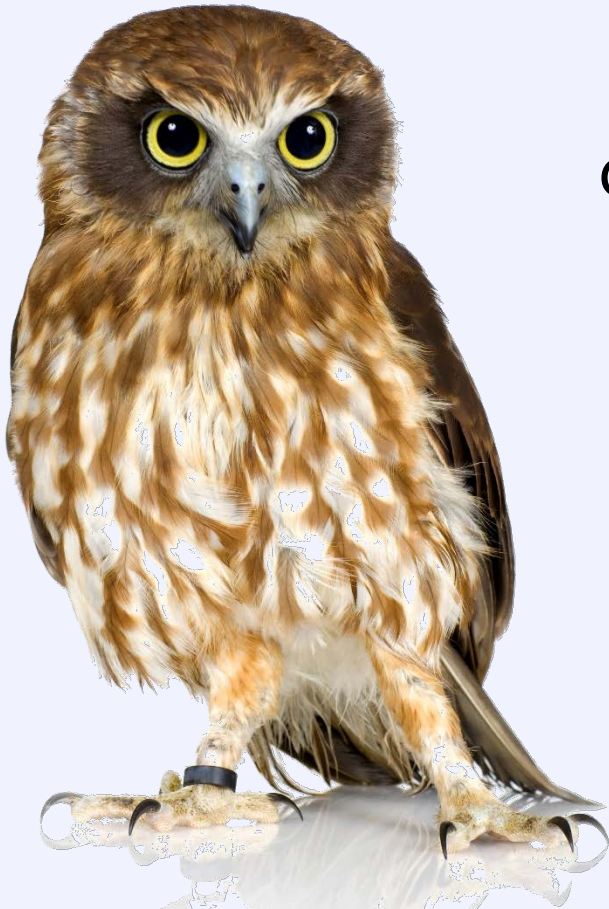
Yill-less

Fray-can



Okay, now we can talk.

# Questions of the day

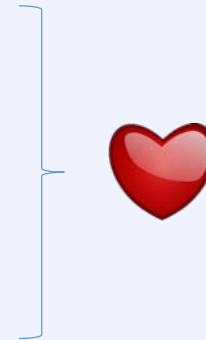


How do we  
**preprocess** data before we  
can extract anything meaningful?

How can we  
convert, normalise, and reduce,  
Big Data into Mineable Data?

# IRDM Chapter 2.1, overview

1. Data type conversion
2. Data normalisation
3. Sampling
4. Dimensionality Reduction



You'll find this covered in  
Aggarwal Chapter 2—2.4.3.2  
Zaki & Meira, Ch. 1, 2.4, 3.5, 6, 7

(Aggarwal, Chapter 2—2.4.3.2)

# We're here to stay

From now on, all lectures will be in HS 002

That is, Tuesday AND Thursday.

# Homework

action speaks louder than words

# Homework Guidelines

Almost every week, we hand out a new homework assignment. You make these **individually** at **home**.

## Homework Timeline:

- week  $n$ : Homework  $i$  handed out
- week  $n + 1$ : Hand in results of Homework  $i$  **at start of lecture**
- week  $n + 2$ : Tutorial session on Homework  $i$

Questions are to be asked during Tutorial sessions.

- in **exceptional** cases of **PANIC** you may email your tutors and **kindly** ask for an appointment – this is **beyond** their official duty.

# Homework Guidelines

Almost every week, we hand out a new homework assignment.

Homework

- we
- we
- we

Please register  
for a tutorial group  
as soon as possible

ture

Questions are to be asked during Tutorial sessions.

- in **exceptional** cases of **PANIC** you may email your tutors and **kindly** ask for an appointment – this is **beyond** their official duty.



# Conversion

From one data type to another

Ch. 2.2.2



# Data

During IRDM we will consider a lot of data.

We will refer to our data as  $\mathbf{D}$ . We will consider many types. Depending on context  $\mathbf{D}$  can be a table, matrix, sequence, etc., with binary, categorical, or real-valued entries.

Most algorithms work **only** for one type of data.

How can we convert  $\mathbf{D}$  from one type into another?

Today we discuss two basic techniques to get us started.

# Numeric to Categorical - Discretisation

## How to go from numeric to categorical?

- by partitioning the domain of numeric attribute  $d_i$  into  $k$  adjacent ranges, and assigning a symbolic label to each.
- e.g. we partition attribute 'age' into ranges of 10 years:  $[0,10)$ ,  $[10,20)$ , etc.

## Standard approaches to choose ranges include

- Equi-width: choose  $[a, b]$  such that  $b - a$  is the same for all ranges
- Equi-log: choose  $[a, b]$  such that  $\log(b) - \log(a)$  is the same for all
- Equi-height: choose ranges such each has the same number of records

**Choosing  $k$  is difficult.** Too low, and you introduce spurious associations. Too high, and your method may not be able to find anything. So, be careful; use domain knowledge, just try, or, use more advanced techniques.

# Categorical to Numeric - Binarisation

How to go from categorical to numeric or binary data?

- by creating a new attribute per categorical attribute-value combination.
- for each categorical attribute  $d$  with  $\phi$  possible values, we create  $\phi$  new attributes, and set their values to 0 or 1 accordingly.

For example, for attribute 'colour' with domain {*red*, *blue*, *green*}, we create 3 new attributes, resp. corresponding to '*colour = red*', '*colour = blue*', and '*colour = green*'.

**Note!** If you are not careful, the downstream method does not know what the 'old' attributes are, and will get lost (or, go wild) with the all the correlations that exist between the 'new' attributes.

# Cleaning

## Missing Values and Normalisation

Ch. 2.3



# Missing Values

Missing values are common in real-world data

- in fact, complete data is the **rare case**
- values are often unobserved, wrongly observed, or lost

Data with missing values needs to be dealt with care

- some methods **are robust to missing values**
  - e.g. naïve Bayes classifiers
- some methods **cannot handle missing values** (natively) (at all)
  - e.g. support vector machines

# Handling missing values

Two common techniques to handle missing values are

- ignoring them
- imputing them

In **imputation**, we **replace** them with 'educated guesses'

- either we use high level statistics, e.g. the mean of the variable
  - perhaps stratified over some class
    - e.g. the mean height vs. the mean height of all professors
- or, we fit a model to the data, and draw values from it
  - e.g. a Bayesian network, a low-rank matrix factorization, etc,
    - **matrix completion** is often used when lots of values are missing

# Some problems

## Imputed values may be wrong!

- this may have a significant effect on results
- especially categorical data is hard
  - the effect of imputation is never 'smooth'

## Ignoring records or variables with missing values may not be possible

- there may not be any data left

## Binary data has the problem of distinguishing non-existent and non-observed data

- did you never see the polar bear living in the Dudweiler forest, or does it not exist?



# Centering

Consider  $\mathbf{D}$  of  $n$  observations over  $m$  variables

- if you want, you can see this as an  $n$ -by- $m$  matrix  $\mathbf{D}$

We say  $\mathbf{D}$  is **zero centered**

if  $mean(\mathbf{d}_i) = 0$  for each column  $\mathbf{d}_i$  of  $\mathbf{D}$

We can center *any* dataset

by subtracting the mean from each columns

# Unit variance

An attribute  $\mathbf{d}$  has unit variance if  $var(\mathbf{d}) = 1$

A dataset  $\mathbf{D}$  has unit variance iff  $\forall \mathbf{d}_i var(\mathbf{d}_i) = 1$

We obtain unit variance by dividing every column by its standard deviation.

# Standardisation

Data that is **zero centered** and has **unit variance** is called **standardised**, or the **z-scores**

- many methods (implicitly) assume data is standardised

Of course, we may apply **non-linear transformations** to the data **before standardising**.

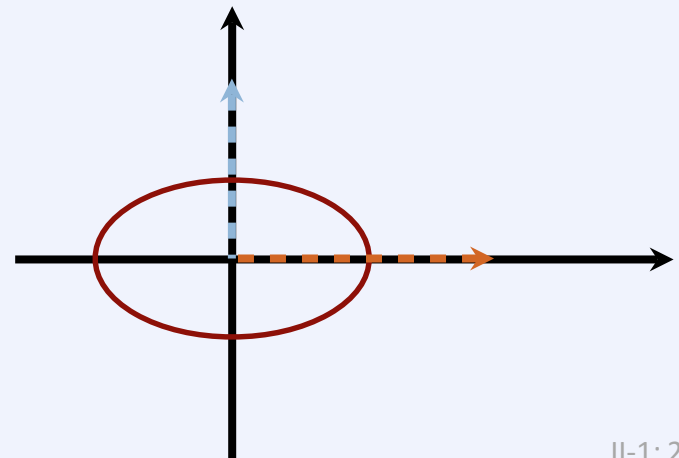
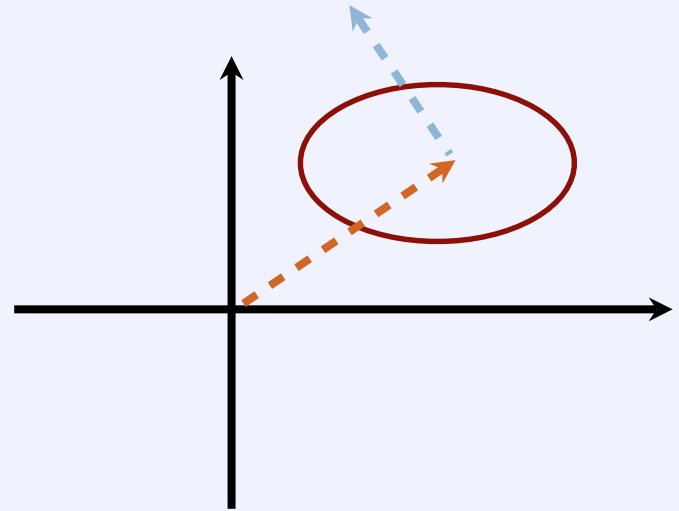
- for example, by taking a logarithm, or the cubic root we can diminish the importance of LARGE values

# Why centering?

Consider the red data ellipse

- the main dimension of variance is from the origin to the data
- the second is orthogonal to the first
- they don't show the variance of the data!

If we **center** the data, however, the directions are correct!



# When not to center?

Centering cannot be applied to all sorts of data

It destroys non-negativity

- e.g. NMF becomes impossible

Centered data won't contain integers

- e.g. no more count or binary data
- can hurt integratability
- itemset mining becomes impossible

Centering destroys sparsity

- bad for algorithmic efficiency
- (we can retain sparsity by only changing non-zero values)

# Why unit variance?

Assume one observation is **height in meters**,  
and one observation is **weight in grams**

- now weight contains much higher values (for humans, at least)
- weight has more weight in calculations

Division by **standard deviation** makes  
all observations **equally important**

- most values now fall between -1 and 1

Question to the audience:  
What's the catch? What's the assumption?

# What's wrong with unit variance?

Dividing by standard deviation is based on the assumption that the values follow a Gaussian distribution

- often this is plausible: Law of Large Numbers, Central Limit Theorem, etc.

Not all data is Gaussian.

- integer counts (especially over small ranges\_
- transaction data

Not all data distributions **have** a mean

- powerlaw distributions

# Dimensionality Reduction

From many to few(er) dimensions

Ch. 2.4.2—2.4.3.2





# Curse of Dimensionality

Life gets harder, **exponentially harder** as dimensionality increases. Not just computationally...

The data volume grows too fast

- 100 evenly-spaced points in a unit interval have a max distance of 0.01
- to get the same distance for adjacent points in a 10-dimensional unit hypercube we need  $10^{20}$  points – that is an increase of factor  $10^{18}$  (!)

And, ten dimensions is really not so many

# Hypercube and Hypersphere

A hypercube is a  $d$ -dimensional cube

- with edge length  $2r$ , its volume is  $vol(H_d(2r)) = (2r)^d$

A Hypersphere is the  $d$ -dimensional ball of radius  $r$

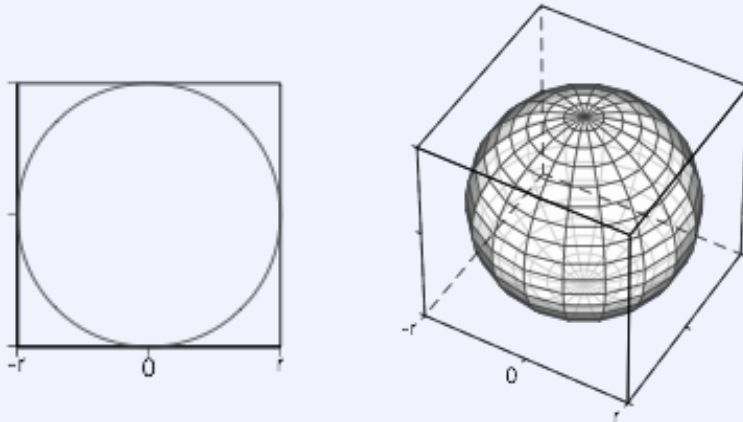
- $vol(S_1(r)) = 2r$
- $vol(S_2(r)) = \pi r^2$
- $vol(S_3(r)) = \frac{4}{3}\pi r^3$
- $vol(S_d(r)) = K_d r^d$  where  $K_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$ 
  - $\Gamma\left(\frac{d}{2} + 1\right) = \left(\frac{d}{2}\right)!$  for even  $d$

# Where is Waldo? (1)

Let's say we consider that **any two points** within the hypersphere are 'close' to each other.

The question then is, how many points are close?  
Or, better, how large is the ball in the box?

# Hypersphere within a Hypercube

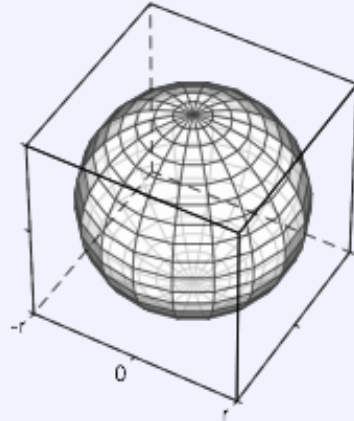
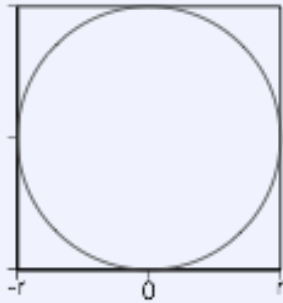


**Mass is in the corners!**

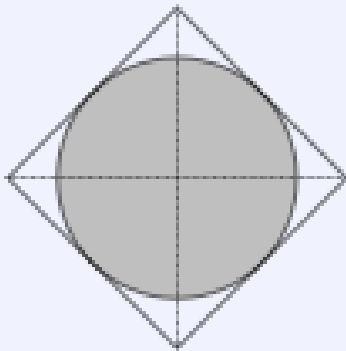
Fraction of volume hypersphere has of surrounding hypercube:

$$\lim_{d \rightarrow \infty} \frac{\text{vol}(S_d(r))}{\text{vol}(H_d(2r))} = \lim_{d \rightarrow \infty} \frac{\pi^{d/2}}{2^d \Gamma\left(\frac{d}{2} + 1\right)} \rightarrow 0$$

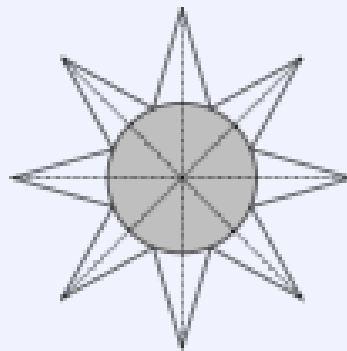
# Hypersphere within a Hypercube



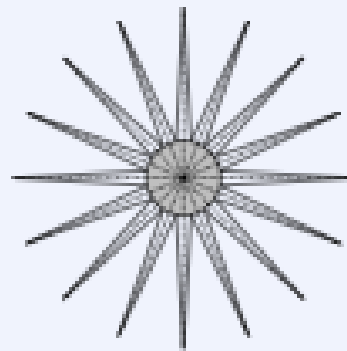
**Mass is in the corners!**



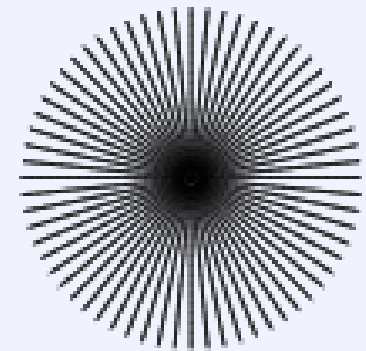
2D



3D



4D



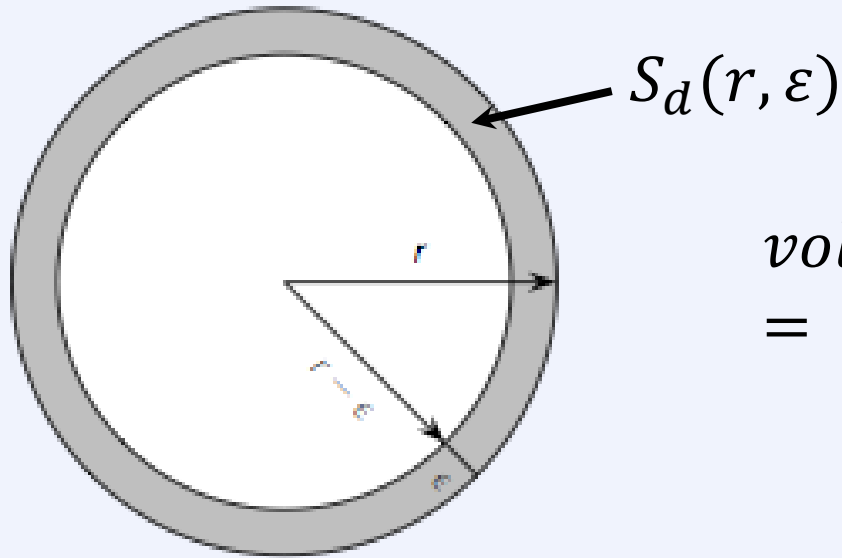
higher dimensions

# Where is Waldo? (2)

Let's now say we consider **any two points** outside the sphere, in the same 'corner' of the box, are 'close' to each other.

Then, the question is, how many points will be close?  
Or, better, how cornered is the data?

# Volume of thin shell of hypersphere



**Mass is in the shell!**

$$\begin{aligned} \text{vol}(S_d(r, \epsilon)) &= \text{vol}(S_d(r)) - \text{vol}(S_d(r - \epsilon)) \\ &= K_d r^d - K_d (r - \epsilon)^d \end{aligned}$$

Fraction of volume in the shell:  $\frac{\text{vol}(S_d(r, \epsilon))}{\text{vol}(S_d(r))} = 1 - \left(1 - \frac{\epsilon}{r}\right)^d$

$$\lim_{d \rightarrow \infty} \frac{\text{vol}(S_d(r, \epsilon))}{\text{vol}(S_d(r))} = \lim_{d \rightarrow \infty} 1 - \left(1 - \frac{\epsilon}{r}\right)^d \rightarrow 1$$

# Back to ~~the Future~~

## Dimensionality Reduction



# Dimensionality Reduction

Aim: reduce the number of dimensions by replacing them with **new** ones

- the new features should capture the “essential part” of the data
- what is considered essential is defined by method you want to use
- using ‘wrong’ dimensionality reduction can lead to useless results

Usually dimensionality reduction methods work on numerical data

- for categorical or binary data, feature **selection** is often more appropriate

# Feature Subset Selection

Data  $D$  may include attributes that are **redundant** and/or **irrelevant** to the task at hand. Removing these will improve efficiency and performance.

That is, given data  $D$  over attributes  $A$ , we want that subset  $S \subseteq A$  of  $k$  attributes such that performance is maximal.

Two big problems:

- 1) how to quantify 'performance'?
- 2) how to search for good subsets?

# Searching for Subsets

How many subsets of  $A$  are there?

- $2^{|A|}$

When do we need to do feature selection?

- when  $A$  is large... oops.

Can we efficiently search for the optimal  $k$ -subset?

- the theoretical answer: depends on your score
- the practical answer: almost never

# Scoring Feature Subsets

There are two main approaches for scoring feature spaces

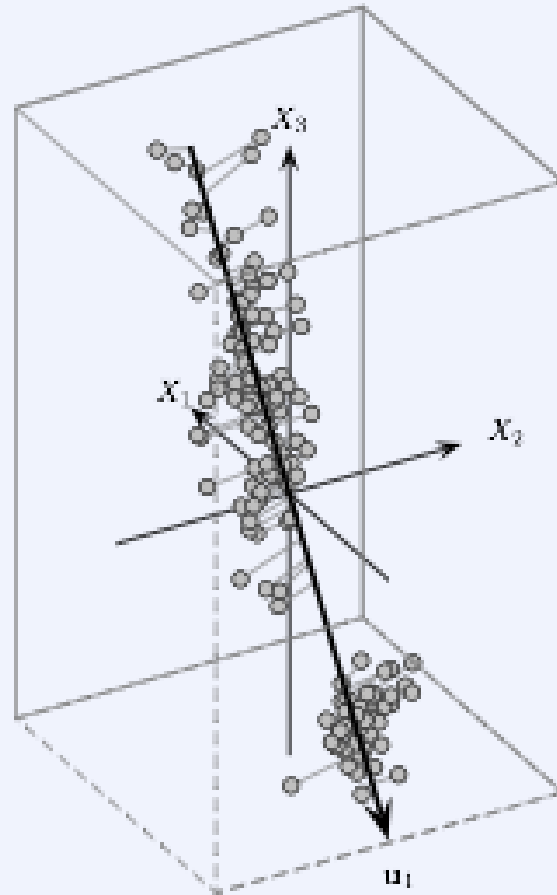
Wrapper methods optimise for **your method** directly

- score each candidate subset by running **your method**
- only makes sense when **your score** is **comparable**
- slow, as it needs to run **your method** for every candidate

Filter methods use an external quality measure

- score each candidate subset using a proxy
- only makes sense when the proxy optimises **your goal**
- can be fast, but may optimise the wrong thing

# Principle component analysis



# Principle component analysis

The goal of **principle component analysis** (PCA) is to project the data onto **linearly uncorrelated variables** in a (possibly) lower-dimensional subspace that **preserves as much of the variance of the original data as possible**

- it is also known as Karhunen–Lòeve transform or Hotelling transform
  - and goes by many other names

In matrix terms, we want to find a column-orthogonal  $n$ -by- $r$  matrix  $\mathbf{U}$  that projects an  $n$ -dimensional data vector  $\mathbf{x}$  into an  $r$ -dimensional vector  $\mathbf{a} = \mathbf{U}^T \mathbf{x}$

# Linear Algebra Recap

# Vectors

A **vector** is

- a 1D array of numbers
- a geometric entity with magnitude and direction

The **norm** of a vector defines its magnitude

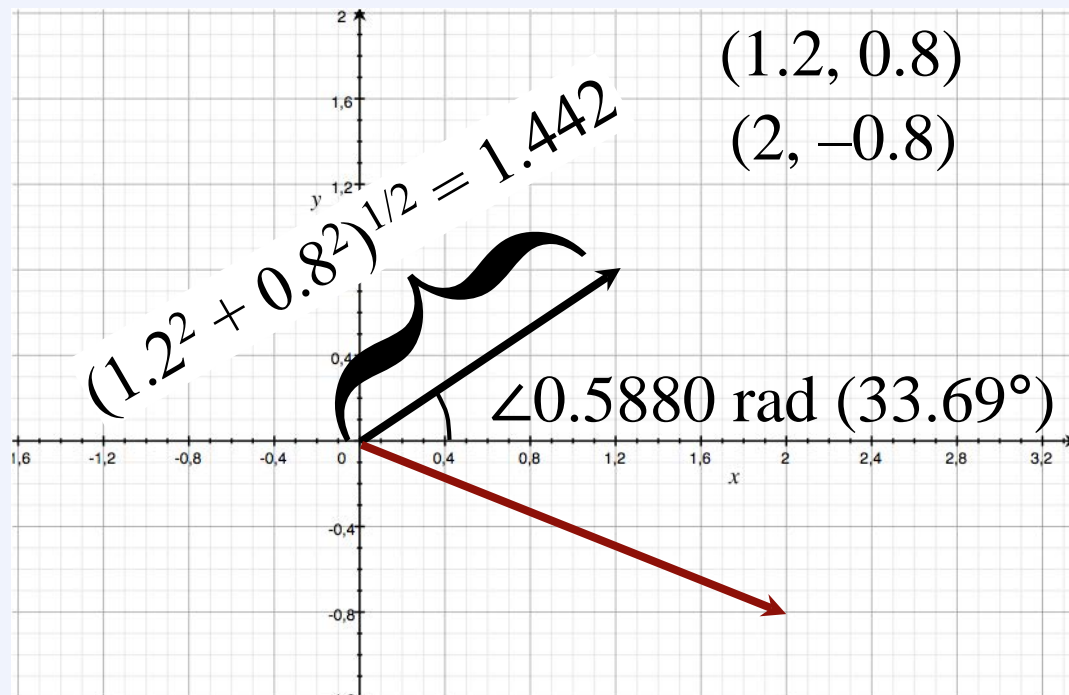
- **Euclidean** ( $L_2$ ) norm:

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$$

- $L_p$  norm ( $1 \leq p \leq \infty$ )

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$$

The direction is the **angle**





# Basic operations on vectors

A **transpose**  $\mathbf{v}^T$  transposes a row vector into a column vector and vice versa

If  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ ,  $\mathbf{v} + \mathbf{w}$  is a vector with  $(\mathbf{v} + \mathbf{w})_i = v_i + w_i$

For vector  $\mathbf{v}$  and scalar  $\alpha$ ,  $(\alpha\mathbf{v})_i = \alpha v_i$

A **dot product** of two vectors  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$  is  $\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^n v_i w_i$

- a.k.a. **scalar product** or **inner product**
- alternative notation:  $\langle \mathbf{v}, \mathbf{w} \rangle$ ,  $\mathbf{v}^T \mathbf{w}$ ,  $\mathbf{v} \mathbf{w}^T$
- in Euclidean space  $\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \cos \theta$

# Basic operations on matrices

Matrix transpose  $\mathbf{A}^T$  has the rows of  $\mathbf{A}$  as its columns

If  $\mathbf{A}$  and  $\mathbf{B}$  are  $n$ -by- $m$  matrices, then  $\mathbf{A} + \mathbf{B}$  is an  $n$ -by- $m$  matrix with  $(\mathbf{A} + \mathbf{B})_{ij} = m_{ij} + n_{ij}$

If  $\mathbf{A}$  is  $n$ -by- $k$  and  $\mathbf{B}$  is  $k$ -by- $m$ , then  $\mathbf{AB}$  is an  $n$ -by- $m$  matrix with

$$(\mathbf{AB})_{ij} = \sum_{\ell=1}^k a_{i\ell} b_{\ell j}$$

- the inner dimension ( $k$ ) must agree (!)
- **vector outer product**  $\mathbf{vw}^T$  (for column vectors) is the matrix product of  $n$ -by-1 and 1-by- $m$  matrices

# Types of matrices

## Diagonal $n$ -by- $n$ matrix

- identity matrix  $I_n$  is a diagonal  $n$ -by- $n$  matrix with 1s in diagonal

$$\begin{pmatrix} x_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & x_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & x_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & x_{n,n} \end{pmatrix}$$

## Upper triangular matrix

- lower triangular is the transpose
- if diagonal is full of 0s, matrix is *strictly triangular*

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,n} \\ 0 & x_{2,2} & x_{2,3} & \cdots & x_{2,n} \\ 0 & 0 & x_{3,3} & \cdots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & x_{n,n} \end{pmatrix}$$

## Permutation matrix

- Each row and column has exactly one 1, rest are 0

Symmetric matrix:  $\mathbf{M} = \mathbf{M}^T$

# Basic concepts

Two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are **orthogonal** if their inner product  $\langle \mathbf{x}, \mathbf{y} \rangle$  is 0

- vectors are **orthonormal** if they have unit norm,  $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$
- in Euclidean space, this means that  $\|\mathbf{x}\| \|\mathbf{y}\| \cos \theta = 0$  which happens iff  $\cos \theta = 0$  which means  $\mathbf{x}$  and  $\mathbf{y}$  are perpendicular to each other

A square matrix  $\mathbf{X}$  is **orthogonal** if its rows and columns are orthonormal

- an  $n$ -by- $m$  matrix  $\mathbf{X}$  is  
row-orthogonal if  $n < m$  and its rows are orthogonal and  
column-orthogonal if  $n > m$  and its columns are orthogonal

# Linear Independence

Vector  $\mathbf{v} \in \mathbb{R}^n$  is **linearly dependent** from a set of vectors  $W = \{\mathbf{w}_i \in \mathbb{R}^n : i = 1, \dots, m\}$  if there exists a set of coefficients  $\alpha_i$  such that

- if  $\mathbf{v}$  is not linearly dependent, it is **linearly independent**
- that is,  $\mathbf{v}$  cannot be expressed as a linear combination of the vectors in  $W$

A set of vectors  $V = \{\mathbf{v}_i \in \mathbb{R}^n : i = 1, \dots, m\}$  is **linearly independent** if  $\mathbf{v}_i$  is linearly independent from  $V \setminus \{\mathbf{v}_i\}$  for all  $i$

# Matrix rank

The **column rank** of an  $n$ -by- $m$  matrix  $\mathbf{M}$  is the number of linearly independent columns of  $\mathbf{M}$

The **row rank** is the number of linearly independent rows of  $\mathbf{M}$

The **Schein rank** of  $\mathbf{M}$  is the least integer  $k$  such that  $\mathbf{M} = \mathbf{A}\mathbf{B}$  for some  $n$ -by- $k$  matrix  $\mathbf{A}$  and  $k$ -by- $m$  matrix  $\mathbf{B}$

- equivalently, the least  $k$  such that  $\mathbf{M}$  is a sum of  $k$  vector outer products

*All these ranks are equivalent!*

# The matrix inverse

The **inverse** of a matrix  $\mathbf{M}$  is the unique matrix  $\mathbf{N}$  for which  $\mathbf{MN} = \mathbf{NM} = \mathbf{I}$

- the inverse is denoted by  $\mathbf{M}^{-1}$

$\mathbf{M}$  has an inverse (is invertible) iff

- $\mathbf{M}$  is square ( $n$ -by- $n$ )
- the rank of  $\mathbf{M}$  is  $n$  (full rank)

Non-square matrices can have left or right inverses

- $\mathbf{MR} = \mathbf{I}$  or  $\mathbf{LA} = \mathbf{I}$

If  $\mathbf{M}$  is orthogonal, then (and only then)  $\mathbf{M}^{-1} = \mathbf{M}^T$

- that is,  $\mathbf{MM}^T = \mathbf{M}^T\mathbf{M} = \mathbf{I}$

# Fundamental decompositions

A **matrix decomposition** (or **factorization**) presents an  $n$ -by- $m$  matrix  $A$  as a product of two (or more) **factor matrices**

- $A = BC$

For approximate decompositions,  $A \approx BC$

The decomposition **size** is the inner dimension of  $B$  and  $C$

- number of columns in  $B$  and number of rows in  $C$
- for exact decompositions, the size is no less than the rank of the matrix



# Eigenvalues and eigenvectors

If  $\mathbf{X}$  is an  $n$ -by- $n$  matrix and  $\mathbf{v}$  is a vector such that  $\mathbf{X}\mathbf{v} = \lambda\mathbf{v}$  for some scalar  $\lambda$ , then

- $\lambda$  is an **eigenvalue** of  $\mathbf{X}$
- $\mathbf{v}$  is an **eigenvector** of  $\mathbf{X}$  associated to  $\lambda$

That is, eigenvectors are those vectors  $\mathbf{v}$  for which  $\mathbf{X}\mathbf{v}$  **only changes their magnitude, not direction**

- it is possible to exactly reverse the direction
- the change in magnitude is the eigenvalue

If  $\mathbf{v}$  is an eigenvector of  $\mathbf{X}$  and  $\alpha$  is a scalar, then  $\alpha\mathbf{v}$  is also an eigenvector with the same eigenvalue

# Properties of eigenvalues

Multiple linearly independent eigenvectors can be associated with the same eigenvalue

- the **algebraic multiplicity** of the eigenvalue

**Every**  $n$ -by- $n$  matrix has  $n$  eigenvectors and  $n$  eigenvalues (counting multiplicity)

- some of these can be complex numbers

If a matrix is symmetric, then all its eigenvalues are real

Matrix is invertible iff all its eigenvalues are non-zero

# Eigendecomposition

The (real-valued) **eigendecomposition** of an  $n$ -by- $n$  matrix  $\mathbf{X}$  is  $\mathbf{X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$

- $\mathbf{\Lambda}$  is a diagonal matrix with eigenvalues in the diagonal
- columns of  $\mathbf{Q}$  are the eigenvectors associated with the eigenvalues in  $\mathbf{\Lambda}$

Matrix  $\mathbf{X}$  has to be diagonalizable

- $\mathbf{P}\mathbf{X}\mathbf{P}^{-1}$  is a diagonal matrix for some invertible matrix  $\mathbf{P}$

Matrix  $\mathbf{X}$  has to have  $n$  real eigenvalues (counting multiplicity)

# Some useful facts

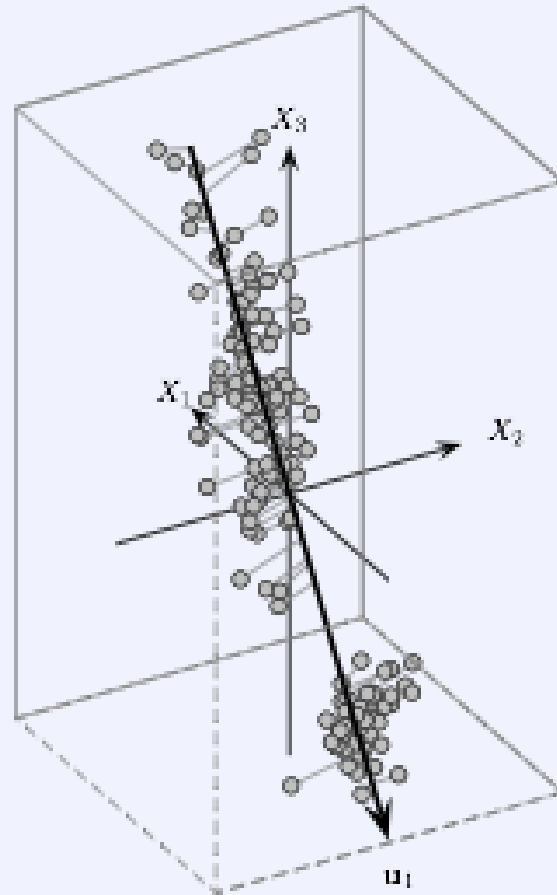
## Not all matrices have eigendecomposition

- not all invertible matrices have eigendecomposition
- not all matrices that have eigendecomposition are invertible
- if  $\mathbf{X}$  is invertible and has eigendecomposition, then  $\mathbf{X}^{-1} = \mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^{-1}$

If  $\mathbf{X}$  is symmetric and invertible (and real), then  $\mathbf{X}$  has eigendecomposition  $\mathbf{X} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$

# Back to PCA

# Example of 1-D PCA (again)



# Principle component analysis, again

The goal of **principle component analysis** (PCA) is to project the data onto **linearly uncorrelated variables** in a (possibly) lower-dimensional subspace that **preserves as much of the variance of the original data as possible**

- it is also known as Karhunen–Lòeve transform or Hotelling transform
  - and goes by many other names

In matrix terms, we want to find a column-orthogonal  $n$ -by- $r$  matrix  $\mathbf{U}$  that projects an  $n$ -dimensional data vector  $\mathbf{x}$  into an  $r$ -dimensional vector  $\mathbf{a} = \mathbf{U}^T \mathbf{x}$

# Deriving PCA: 1-D case (1)

We assume our data is **standardised**. We want to find a unit vector  $\mathbf{u}$  that maximises the variance of the projections  $\mathbf{u}^T \mathbf{x}_i \mathbf{u}$ .

Scalar  $\mathbf{u}^T \mathbf{x}_i$  gives the coordinate of  $\mathbf{x}_i$  along  $\mathbf{u}$ . As our data is centered the mean is 0, projected to  $\mathbf{u}$  this has coordinate 0.

The variance of the projection is

$$\begin{aligned}\sigma^2 &= \frac{1}{n} \sum_{i=1}^n (\mathbf{u}^T \mathbf{x}_i - \mu_{\mathbf{u}})^2 \\ &= \mathbf{u}^T \boldsymbol{\Sigma} \mathbf{u}\end{aligned}$$

$$\boldsymbol{\Sigma} = \frac{1}{n} \sum_i \mathbf{x}_i \mathbf{x}_i^T$$

The covariance matrix  
for centered data



# Deriving PCA: 1-D case (2)

To maximise the variance  $\sigma^2$ , we maximise

$$J(\mathbf{u}) = \mathbf{u}^T \Sigma \mathbf{u} - \lambda(\mathbf{u}^T \mathbf{u} - 1)$$

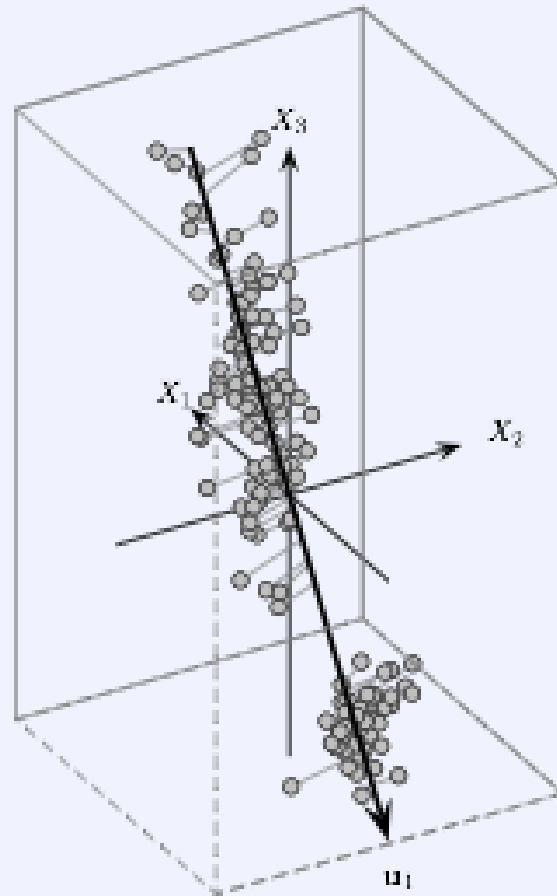
where the second term ensures  $\mathbf{u}$  is a unit vector

Solving the derivative gives  $\Sigma \mathbf{u} = \lambda \mathbf{u}$

- $\mathbf{u}$  is an eigenvector and  $\lambda$  is an eigenvalue
- further,  $\mathbf{u}^T \Sigma \mathbf{u} = \mathbf{u}^T \lambda \mathbf{u}$  implies that  $\sigma^2 = \lambda$
- so, to maximise variance, we need to take the largest eigenvalue

Thus, the **first principal component**  $\mathbf{u}$  is the dominant eigenvector of the covariance matrix  $\Sigma$

# Example of 1-D PCA



# Deriving PCA: $r$ -dimensions

The second principal component should be orthogonal to the first one, and maximise variance

- adding this constraint and doing the derivation shows that the **second principal component** is the **eigenvector** associated with the **second highest eigenvalue**
- in fact, to find  $r$  principal components we simply take the  $r$  eigenvectors of  $\Sigma$  associated to the  $r$  highest eigenvalues
- **the total variance is the sum of the eigenvalues**

It also turns out that maximising the variance minimises the mean squared error

$$\frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{U}^T \mathbf{x}_i \mathbf{U}\|^2$$

# Computing PCA

We have two options. Either

- we compute the covariance matrix, and its top- $k$  eigenvectors, or
- we use singular value decomposition (SVD)
  - because the covariance matrix  $\Sigma = \mathbf{X}\mathbf{X}^T$  and if  $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ , the columns of  $\mathbf{U}$  are the eigenvectors of  $\mathbf{X}\mathbf{X}^T$
  - as computing the covariance matrix can cause numerical stability issues with the eigendecomposition, this approach is preferred

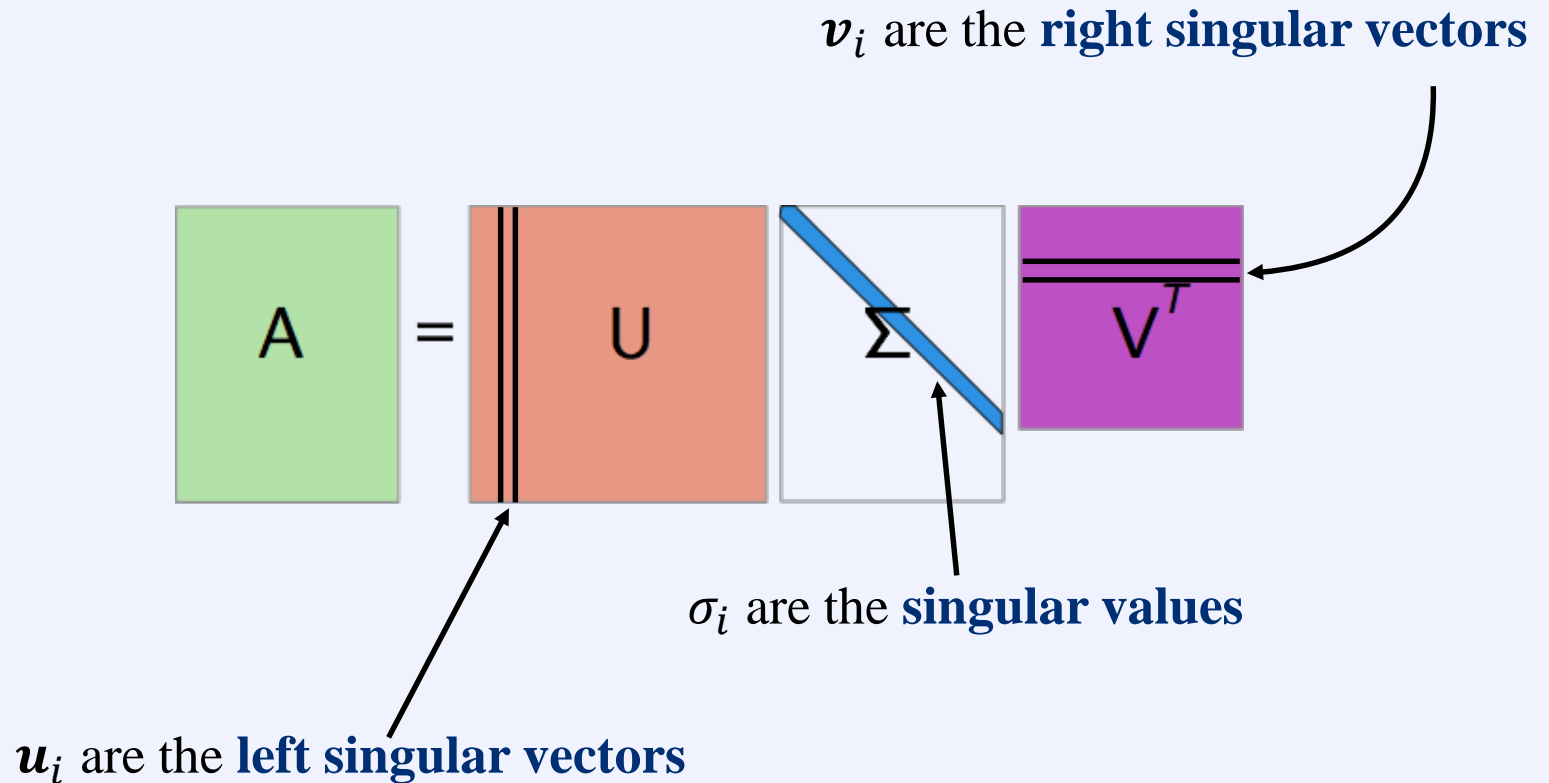
# SVD: The Definition

**Theorem.** For every  $\mathbf{A} \in \mathbb{R}^{n \times m}$   
there exists an  $n$ -by- $n$  *orthogonal* matrix  $\mathbf{U}$   
and an  $m$ -by- $m$  *orthogonal* matrix  $\mathbf{V}$   
such that  $\mathbf{U}^T \mathbf{A} \mathbf{V}$  is an  $n$ -by- $m$  *diagonal* matrix  $\mathbf{\Sigma}$  with  
values  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min\{n,m\}} \geq 0$  in its diagonal

That is, *every*  $\mathbf{A}$  has decomposition  $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$

It is called the **singular value decomposition** of  $\mathbf{A}$

# SVD in a Picture



# SVD, some useful equations

- $A = U\Sigma V^T = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ 
  - SVD expresses  $A$  as a *sum* of rank-1 matrices
- $A^{-1} = (U\Sigma V^T)^{-1} = V\Sigma^{-1}U^T$ 
  - if  $A$  is invertible, so is its SVD
- $A^T A \mathbf{v}_i = \sigma_i^2 \mathbf{v}_i$  (for any  $A$ )
- $AA^T \mathbf{u}_i = \sigma_i^2 \mathbf{u}_i$  (for any  $A$ )

# Truncated SVD

The **rank** of the matrix is the number of its non-zero singular values (write  $\mathbf{A} = \sum_i \sigma_i \mathbf{u}_i \mathbf{v}_i^T$ )

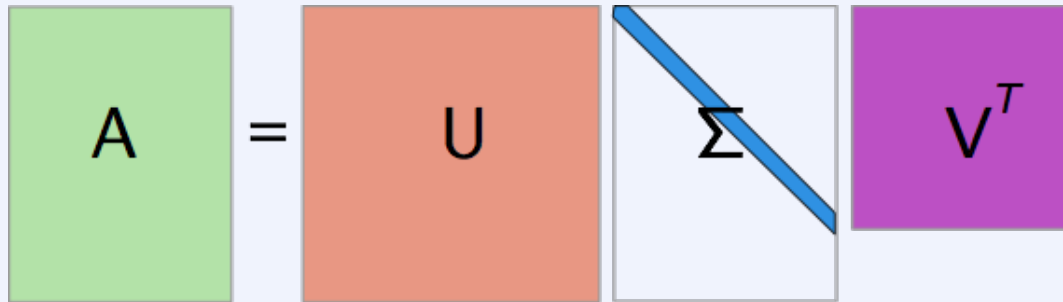
The **truncated SVD** takes the first  $k$  columns of  $\mathbf{U}$  and  $\mathbf{V}$  and the main  $k$ -by- $k$  submatrix of  $\mathbf{\Sigma}$

- $\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$
- $\mathbf{U}_k$  and  $\mathbf{V}_k$  are column-orthogonal

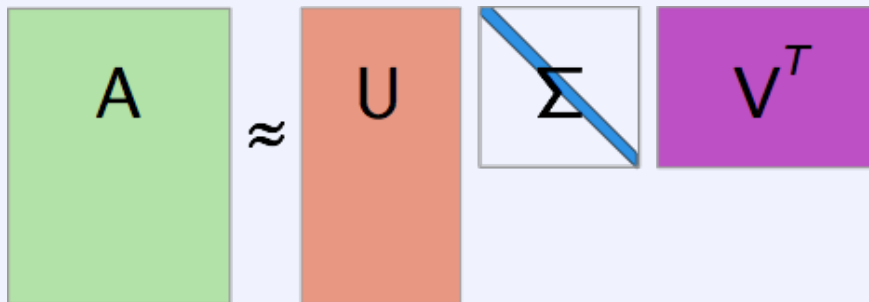


# Truncated SVD

Full



Truncated



# Why is SVD important?

It lets us compute **various norms**

It tells about the **sensitivity of linear systems**

It shows the **dimensions of the fundamental subspaces**

It gives optimal solutions to **least-square linear systems**

It gives the **least-error rank-k-decomposition**

**Every matrix has one**

# SVD as Low-Rank Approximation

## Theorem:

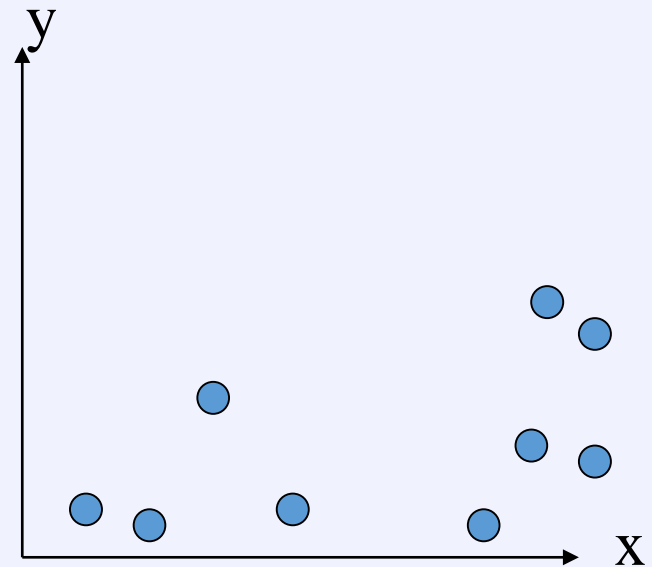
Let  $\mathbf{A}$  be an  $m$ -by- $n$  matrix with rank  $r$ , and let  $\mathbf{A}_k = \mathbf{U}_k \times \mathbf{\Sigma}_k \times \mathbf{V}_k^T$ , where the  $k \times k$  diagonal matrix  $\mathbf{\Sigma}_k$  contains the  $k$  largest singular values of  $\mathbf{A}$  and the  $m \times k$  matrix  $\mathbf{U}_k$  and the  $n \times k$  matrix  $\mathbf{V}_k$  contain the corresponding Eigenvectors from the SVD of  $\mathbf{A}$ .

Among all  $m$ -by- $n$  matrices  $\mathbf{C}$  with rank at most  $k$   $\mathbf{A}_k$  is the matrix that minimizes the Frobenius norm

$$\|\mathbf{A} - \mathbf{C}\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n (\mathbf{A}_{ij} - \mathbf{C}_{ij})^2$$

## Example:

$m = 2, n = 8, k = 1$   
projection onto  $x'$  axis  
minimizes „error“ or  
maximizes „variance“  
in  $k$ -dimensional space



# SVD as Low-Rank Approximation

## Theorem:

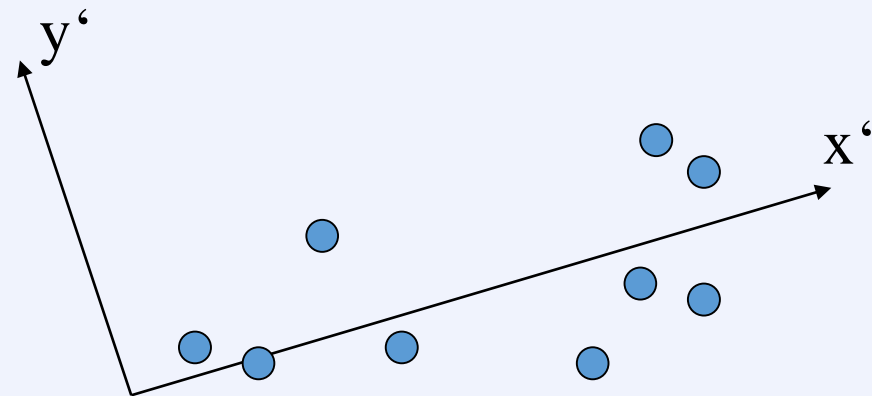
Let  $\mathbf{A}$  be an  $m$ -by- $n$  matrix with rank  $r$ , and let  $\mathbf{A}_k = \mathbf{U}_k \times \mathbf{\Sigma}_k \times \mathbf{V}_k^T$ , where the  $k \times k$  diagonal matrix  $\mathbf{\Sigma}_k$  contains the  $k$  largest singular values of  $\mathbf{A}$  and the  $m \times k$  matrix  $\mathbf{U}_k$  and the  $n \times k$  matrix  $\mathbf{V}_k$  contain the corresponding Eigenvectors from the SVD of  $\mathbf{A}$ .

Among all  $m$ -by- $n$  matrices  $\mathbf{C}$  with rank at most  $k$   $\mathbf{A}_k$  is the matrix that minimizes the Frobenius norm

$$\|\mathbf{A} - \mathbf{C}\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n (\mathbf{A}_{ij} - \mathbf{C}_{ij})^2$$

## Example:

$m = 2, n = 8, k = 1$   
projection onto  $x'$  axis  
minimizes „error“ or  
maximizes „variance“  
in  $k$ -dimensional space



# SVD as Low-Rank Approximation

## Theorem:

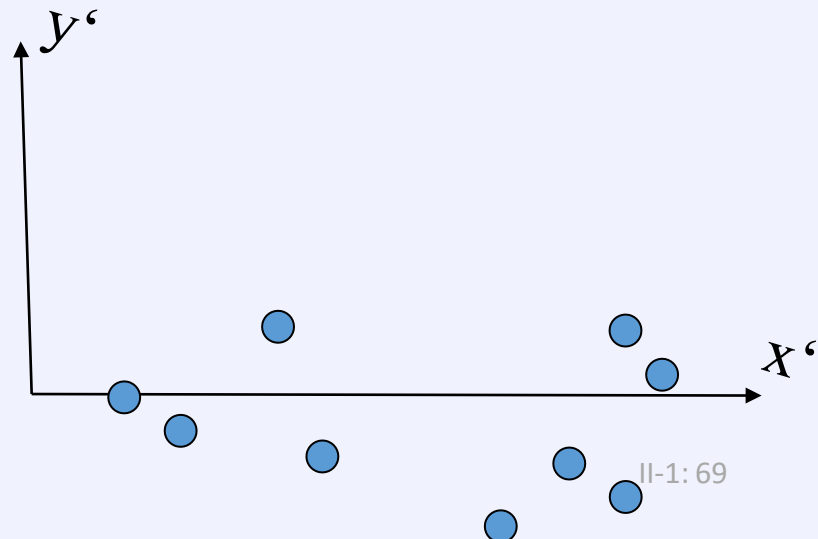
Let  $\mathbf{A}$  be an  $m$ -by- $n$  matrix with rank  $r$ , and let  $\mathbf{A}_k = \mathbf{U}_k \times \mathbf{\Sigma}_k \times \mathbf{V}_k^T$ , where the  $k \times k$  diagonal matrix  $\mathbf{\Sigma}_k$  contains the  $k$  largest singular values of  $\mathbf{A}$  and the  $m \times k$  matrix  $\mathbf{U}_k$  and the  $n \times k$  matrix  $\mathbf{V}_k$  contain the corresponding Eigenvectors from the SVD of  $\mathbf{A}$ .

Among all  $m$ -by- $n$  matrices  $\mathbf{C}$  with rank at most  $k$   
 $\mathbf{A}_k$  is the matrix that minimizes the Frobenius norm

$$\|\mathbf{A} - \mathbf{C}\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n (\mathbf{A}_{ij} - \mathbf{C}_{ij})^2$$

## Example:

$m = 2, n = 8, k = 1$   
projection onto  $x'$  axis  
minimizes „error“ or  
maximizes „variance“  
in  $k$ -dimensional space




# Problems with PCA and SVD

Many characteristics of the input data are lost

- non-negativity
- integrality
- sparsity
- ...

Also, computation is costly for big matrices

- approximate methods exist that can do SVD in a single sweep 

# Conclusions

## Preprocessing your data is crucial

- Conversion, normalisation, and dealing with missing values

## Too much data is a problem

- computational complexity
- can 'solved' by sampling

## Too high dimensional data is a problem

- everything is evenly far from everything
- feature selections retains important features of the data
- PCA and SVD reduce dimensionality with global guarantees

# *Thank you!*

## Preprocessing your data is crucial

- Conversion, normalisation, and dealing with missing values

## Too much data is a problem

- computational complexity
- can 'solved' by sampling

## Too high dimensional data is a problem

- everything is evenly far from everything
- feature selections retains important features of the data
- PCA and SVD reduce dimensionality with global guarantees