Numerical Linear Algebra and Machine Learning: Low-rank matrix factorizations for Data Analysis

Ángeles Martínez Calomardo
Department of Mathematics and Earth Science
University of Trieste

e-mail:amartinez@units.it
webpage: www.dmi.units.it/~acalomar

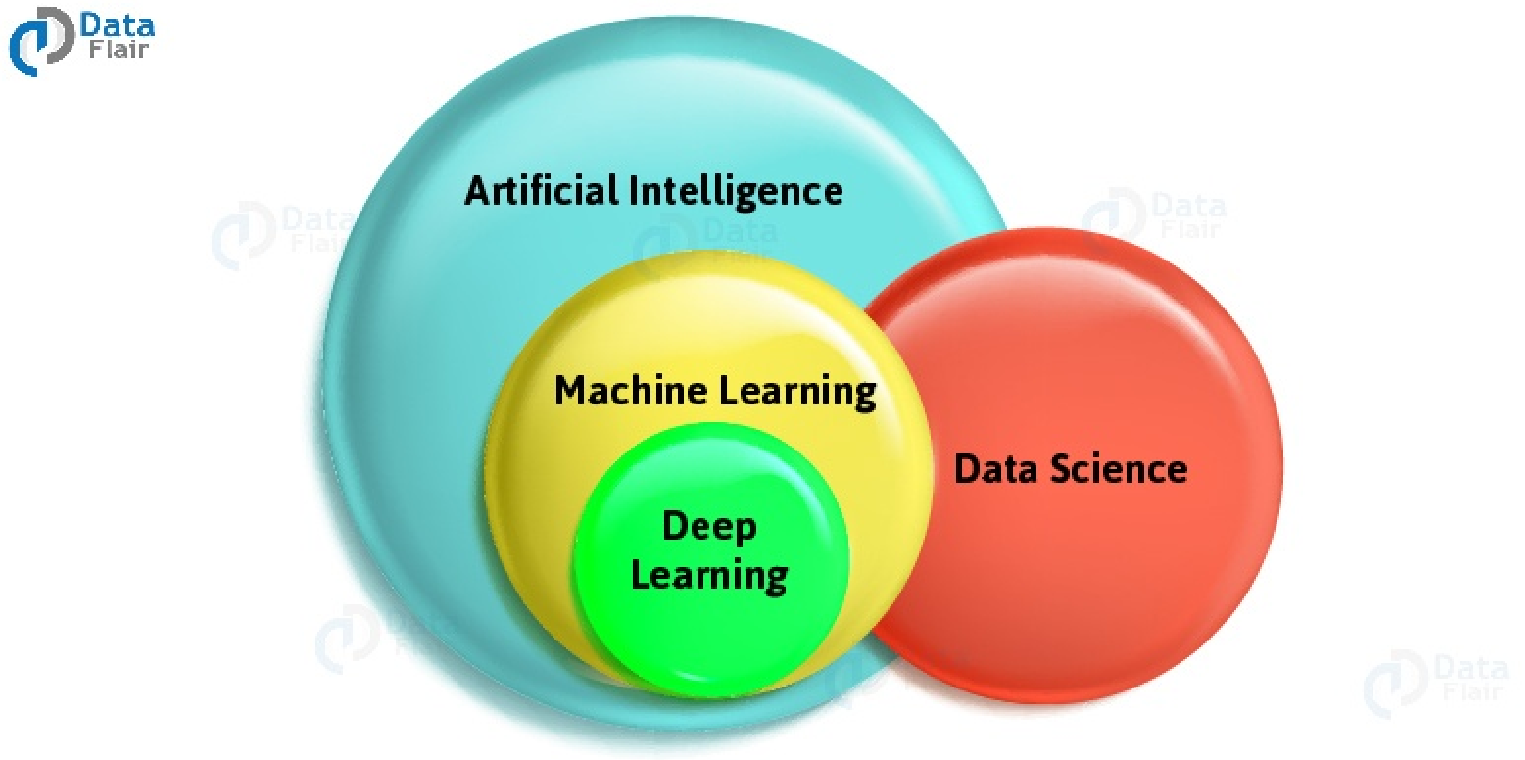
December 9, 2020

Artificial Intelligence vs Machine Learning vs Deep Learning vs Data Science

There is a lot of buzz around the terminology Data Science, Artificial Intelligence, Machine Learning, Deep Learning

Many times people use this interchangeably without actually understanding what it actually means.

The diagram briefly shows how all these concepts are related.



Artificial Intelligence vs Machine Learning vs Deep Learning vs Data Science

Artificial Intelligence (AI) refers to the overall gamut of techniques which enables computer to think. The core objective of AI is to impart human intelligence to machines and enable them to act like humans.

Machine Learning (ML) is a sub area of AI based on (statistical) tools to learn from data. It enables the computer to make a decision based on data rather than on explicitly rule based programs to perform a specific task.

Deep Learning (DL) is a recent area which has taken shape since 2006 and has given a new approach to ML. It uses Multilayered Neural Networks (inspired by human brain).
Deep neural networks obtain impressive results for image, sound and language recognition or to address complex problems in physics. They are partly responsible for the renewal of artificial intelligence.

Some of the most important advances in the last years in AI has happened using Deep Learning!

Data Science It overlaps with Machine Learning and Artificial Intelligence techniques, and (not so much) with Deep Learning. It enables us to analyze and manipulate large volumes of data to find meaning and appropriate information.

What is Data Science?

Data Science is a set of algorithms and techniques to analyze and understand actual phenomena with "data".

The aim of data science is to reveal the features or the hidden structure of complicated natural, human and social phenomena with data.

In other words, data science extracts knowledge from data.

To achieve this, data science draws on elements of machine learning, and mathematical fields such us linear algebra, optimization and statistics.

The study of extracting information from data has become crucial in many many field ranging from business (e-commerce), engineering, medicine (bioinformatics), and many others

In order to obtain information from data it is not necessarily implied that the amount of data is big, **but often it is**!

Learning from data in Data Science and Machine Learning

The task of extracting meaningful information from the collected data vary between areas and two broad classes of techniques are considered:

Supervised learning (classification, regression)

A model is trained from a set of **labeled** data divided into classes.

Afterward the model is applied to predict the class label for incoming unlabeled data.

It is called supervised learning because the training data set supervises the learning process.

Unsupervised learning (clustering, dimensionality reduction)

The data being processed are **unlabeled**.

In the lack of prior knowledge, the algorithm tries to search for a similarity to generate clusters and assign classes.

Matrix method for Data Science

Data is represented as **vectors, matrices or tensors**.

Large $m \times n$ matrices are common in applications since the data often consist of m *objects*, each of which is described by n *features*.

Examples of object–feature pairs include:

- documents and words contained in those documents;
- genomes and environmental conditions under which gene responses are measured;
- MR Images and type of cancer lessions;
- web groups and individual users;
- and many others.

Also in many applications data is represented as tensors, arrays of real numbers with three or more indices i.e. $T = (t_{ijk}) \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

Examples:

- a collection of color images of say 32×32 pixels each and three channels (RGB)
- a collection of pictures of the same person with different expressions for face recognition systems.

Example 1. Term-document matrices are used in information retrieval

Consider the following selection of five documents. **Key words** (in boldface) will be called **terms**.

Document 1	The **Google matrix** P is a model of the **Internet**.
Document 2	P_{ij} is nonzero if there is a **link** from **Web page** j to i.
Document 3	The **Google matrix** is used to **rank** all **Web pages**.
Document 4	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5	**England** dropped out of the top 10 in the **FIFA ranking**.

If we count the frequency of terms in each document we get the following result:

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
Web	0	1	1	0	0

Example 1. Continued.

Each document is a vector in $\mathbb{R}^{10}$ and all documents can be organized into the term-document matrix A.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Now find all documents that all relevant to the query **ranking** of **Web pages**. This is represented by the query vector $q \in \mathbb{R}^{10}$:

$$q = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The query itself is a document.

The information retrieval task can now be formulated as a mathematical problem:

$$\boxed{\textbf{find the columns of } A \textbf{ that are close to the vector } q\textbf{.}}$$

To solve this problem we must use some distance measure in $\mathbb{R}^{10}$.

Example 2. Pattern recognition

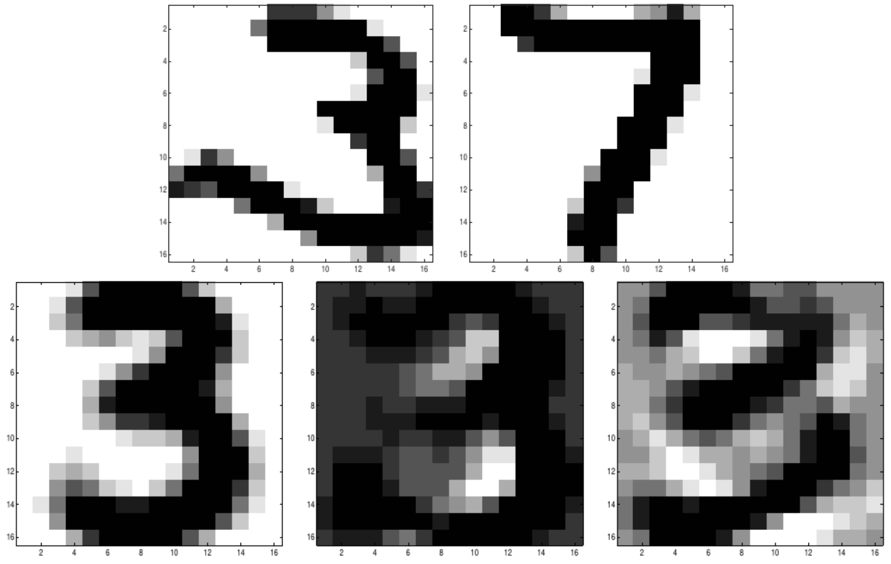
The classification of handwritten digits is a model problem in pattern recognition.

Here vectors are used to represent digits.

The image of one digit is a 16 $\times$ 16 matrix of numbers, representing gray scale. It can also be represented as a vector in $\mathbb{R}^{256}$, by stacking the columns of the matrix.

A set of n digits (handwritten 3s, say) can then be represented by a matrix $A \in \mathbb{R}^{256 \times n}$, and the columns of A span a subspace of $\mathbb{R}^{256}$.

Three basis vectors of the 3-subspace are illustrated in Figure below (bottom).



Example 3: Face Recognition (Tensor data representation)

Assume that we have a collection of images of n_p persons, where each image is an $m_{i1} \times m_{i2}$ array with $m_{i1} \cdot m_{i2} = n_i$.

Usually the columns of the images are stacked so that each image is represented by a vector in $\mathbb{R}^{n_i}$.

Each person has been photographed with n_e different facial expressions.

The collection of images is stored as a tensor

$$\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}$$

If, for instance we also had photos of each person with different illumination,viewing angles, etc., then we could represent the image collection by a tensor of higher degree.

The following images (five images of the same person with different expressions) are taken from the Yale Face Database by reducing each image to 112×78 pixels stored in a vector of length 8736.

The key role of Linear Algebra in Data Science and Machine Learning

- Linear Algebra makes the core foundation for Machine Learning algorithms ranging from simple linear regressions to Deep Neural Networks and Data Science.
- The major reason for that is that the set of data can often be represented using a 2-D matrix where the column represents the features and the row represents the different sample data points.
- Many data analysis applications deal with large matrices and matrix computations using all of the values in the data matrix are sometimes redundant or rather computationally expensive.
- Besides, rows and columns of numbers do not have a meaning of their own and **it's almost impossible to spot any relevant correlation or pattern at a first glance**.

Fundamental step

To construct a **compressed representation** of A such that the most important part which is needed for further computations could be extracted easily, and where the hidden structure of the data may be easier to analyze and interpret.

Matrix factorizations and low-rank approximation of matrices

This compressed representation is a low-rank approximation of the data matrix.

Low-rank approximation of matrices has been well studied in literature and have received much attention over the last years.

It consists in approximating a matrix by one whose rank is less than that of the original matrix while retaining most information contained in the data (i.e. **with limited loss of information**):

Low-rank matrix approximation

Given $A \in \mathbb{R}^{m \times n}$, the low-rank approximation (rank k) of A is given by

$$A \approx WH$$

where $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$

Note that only $k(m+n)$ entries must be stored instead of the mn entries of the original matrix.

Low-rank matrix factorizations are a ubiquitous tool in data science.

Example: matrix factorization techniques used for recommender systems[1].

[1] Netflix challenge https://en.wikipedia.org/wiki/Netflix_Prize

Example. The Netflix recommender system

A recommender system, or a recommendation system, is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

The **Netflix Prize** was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films.

i.e. without the users or the films being identified except by numbers assigned for the contest.

Training data provided by Netflix

100M ratings (from 1 to 5) of 17K movies by 500K users
in the form of a triplet of numbers: (User,Movie,Rating),
i.e. one such entry might be (105932,14002,3).

Data are organized as a big sparsely filled matrix A (1.2% is nonzero), with users as rows and movies as columns.

Each nonzero coefficient A_{ij} contains an observed rating (1-5) for movie i by user j.

Now given (User,Movie,?) not in the database, **we want to predict how the given User would rate the given Movie**

Example. The Netflix recommender system

The Data matrix is very big and sparse

It is important to express the information contained in the data matrix in a more compact and descriptive way.

IDEA: Feature extraction
Any given movie can be described in terms of some basic attributes such as **overall quality**, whether it's an **action movie** or a **comedy**, what **stars** are in it, and so on.

And every user's preferences can likewise be roughly described in terms of whether they prefer **action movies or comedies**, **what stars they like**, and so on.

Imagine that we limit it to $k = 40$ aspects, such that each movie is described only by 40 values saying how much that movie exemplifies each aspect, and correspondingly each user is described by 40 values saying how much they prefer each aspect.

What does this mean in matrix terms?

Low-rank data matrix approximation

In matrix terms, the original matrix would be decomposed into two very oblong matrices:

- the 17,000 x 40 movie aspect matrix W,
- the 40 x 500,000 user preference matrix H.

W and H obtained by minimizing some function error like $\|A - WH\|$.

These can combined all together into a rating matrix R in which R_{ij} is the scalar product between the i-th row of W and the j-th column of H:

$$R_{ij} = \sum_{k=1}^{40} W_{ik} H_{kj}$$

Example. Terminator might be (action=1.2,chickflick=-1,...), and user Joe might be (action=3,chickflick=-1,...), and when you combine the two you get Joe likes Terminator with $3 \cdot 1.2 + (-1) \cdot (-1) + \ldots = 4.6 + \ldots$ (negative weights may be OK)

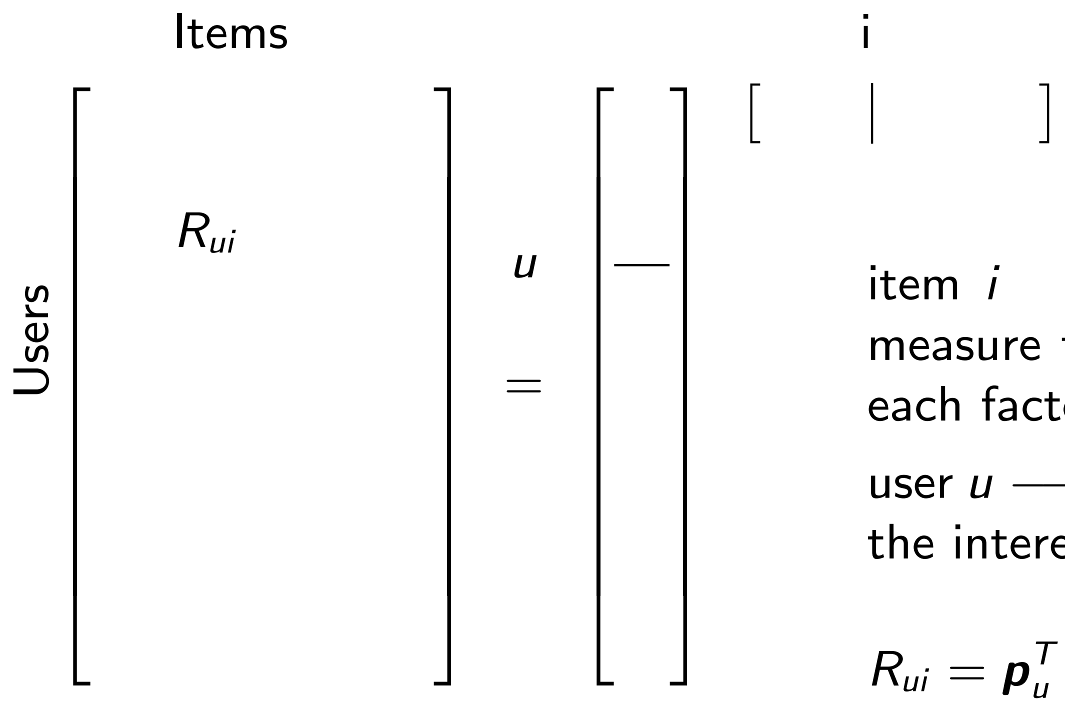
A high R_{ij} value means that User j would probably highly rate Movie i.

A matrix factorization model for recommender systems

Formally:

A matrix factorization model maps both **users** and **items** to a joint latent factor space of dimensionality k.

In that space, user-item interactions are modeled as **inner products.**



item $i \longrightarrow \boldsymbol{q}_i \in \mathbb{R}^k$; $\boldsymbol{q}_i$: Its elements measure the extent to which item i possess each factor.

user $u \longrightarrow \boldsymbol{p}_u \in \mathbb{R}^k$ $\boldsymbol{p}_u$: Its elements measure the interest that user u has on each factor.

$R_{ui} = \boldsymbol{p}_u^T \boldsymbol{q}_i$ User u rating for item i.

High correspondance bewteen item and user factors leads to a **recommendation**

The Singular Value Decomposition (SVD) of a matrix

SVD is the “creme de la creme” of rank-reducing decompositions, the decomposition that all others try to beat.

G. W. Stewart
Matrix Algorithms Vol. 1. Basic decompositions.
SIAM, 1998.

The Singular Value Decomposition (SVD) of a matrix

The Singular Value Decomposition (SVD) is the generalization of the concept of eigendecomposition for general matrices.

Theorem

For every matrix $A \in \mathbb{R}^{m\times n}$ there exist two orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and a matrix $\Sigma \in \mathbb{R}^{m\times n}$ with the following form

$$\Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}, \qquad \Sigma_r = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix},$$

with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0,\ r \leq \min\{m, n\}$ such that

$$A = U\Sigma V^T.$$

The scalars σ_i are called singular values of A. The columns of V and U are called right and left singular vectors of A, respectively.

The SVD of a matrix

Since V and U are orthogonal, the SVD decomposition can be equivalently written as

$$AV = U\Sigma, \qquad A\begin{bmatrix} \boldsymbol{v}_1 & \dots & \boldsymbol{v}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_m \end{bmatrix} \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & \mathbf{0} \\ & & \sigma_r & \\ \hline & \mathbf{0} & & 0 \end{array}\right]$$

Considering matrices U and V columnwise, i.e., $U = \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_m \end{bmatrix}$ and $V = \begin{bmatrix} \boldsymbol{v}_1 & \dots & \boldsymbol{v}_n \end{bmatrix}$ we have that

$$A\boldsymbol{v}_1 = \sigma_1 \boldsymbol{u}_1, \ \dots, \ A\boldsymbol{v}_r = \sigma_r \boldsymbol{u}_r$$

and

$$A\boldsymbol{v}_{r+1} = 0, \ \dots, \ A\boldsymbol{v}_n = 0$$

The reduced form of the SVD

The column-row multiplication of $U\Sigma$ times V^T separates A into rank 1 pieces:

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T.$$

Observing that on the previous sum only the first r columns of U and V are involved we can reduce the equation $AV = U\Sigma$ to

$$AV_r = U_r\Sigma_r,$$

where now $V_r = \begin{bmatrix} \boldsymbol{v}_1 & \dots & \boldsymbol{v}_r \end{bmatrix}$, $U_r = \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_r \end{bmatrix}$, and $\Sigma_r = \operatorname{diag}(\sigma_1, \dots, \sigma_r)$.

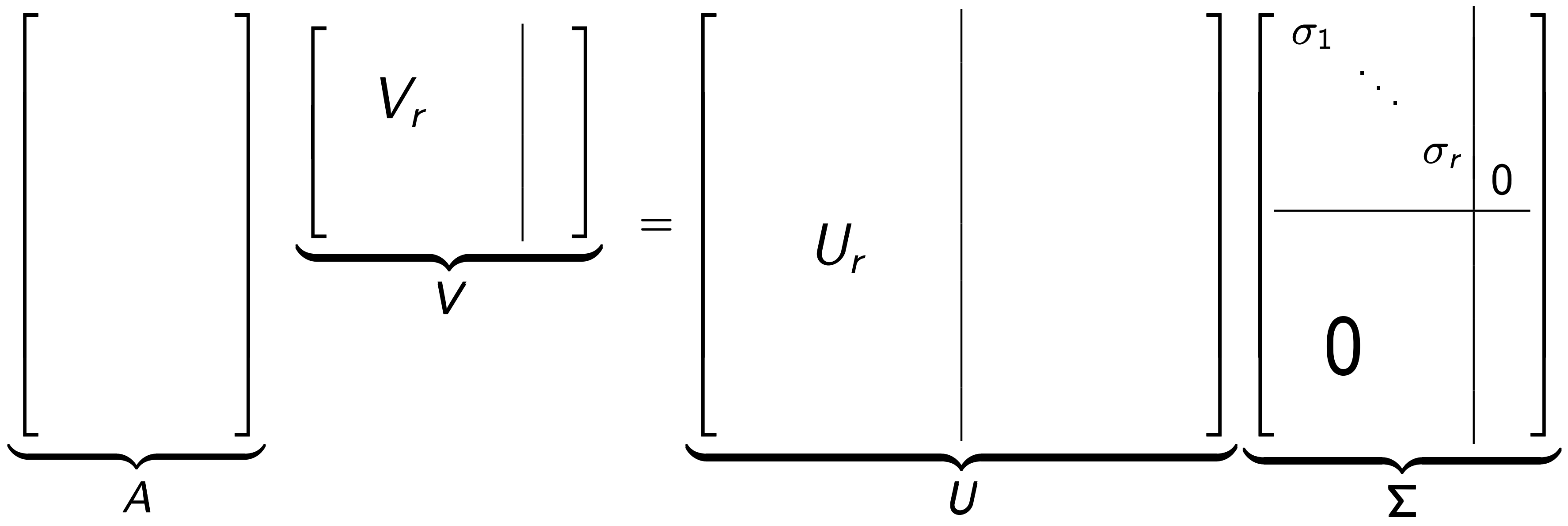
Note: Matrices U_r, V_r are no longer square. They satisfy $U_r^T U_r = I, V_r^T V_r = I$ but, in the common case in which $r < \min\{m, n\}$, $U_r U_r^T \neq I$, $V_r V_r^T \neq I$.

Exercise. Prove that

$$A = U_r \Sigma_r V_r^T.$$

The SVD of a matrix

Pictorial representation of the **sizes** of the matrices involved in the SVD:



The Important Fact in Data Science

Like other common factorizations such that $A = LU$, “LU” factorization, or $A = QR$, “QR” factorization, the SVD separates the matrix into rank one pieces.

A special property of the SVD is that those pieces come in order of importance:

1. $A_1 = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T$ is the rank one matrix closest to A.
2. ... ...
3. $A_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \ldots \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T = \sum_{i=1}^k \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ is the rank k matrix closest to A.

Theorem (Eckart-Young)

If B has rank k then

$$\|A - A_k\| \leq \|A - B\|$$

for every unitarily invariant matrix norm (e.g. the 2-norm or the Frobenius norm).

The Important Fact in Data Science

The Eckart-Young theorem tell us that:

Best rank k approximation

Truncating the Singular Value Decomposition at some number $k \ll \min\{m, n\}$ terms provides the "best" rank-k approximation to A when measured with respect to any unitarily invariant matrix norm.

So we can obtain an optimal approximation of rank k by setting the singular values beyond the $k-$th to zero (zeroing out the $r-k$ trailing singular values).

The error in the approximation can be explicitly computed depending on the norm. Two most common examples:

$$\begin{aligned}
\min_{\mathtt{rank}(B)\le k} \|A-B\|_2 &= \|A-A_k\|_2 = \sigma_{k+1} \\
\min_{\mathtt{rank}(B)\le k} \|A-B\|_F &= \|A-A_k\|_F = \sqrt{\sigma_{k+1}^2+\ldots+\sigma_r^2}.
\end{aligned}$$

Note finally that $A_k = U_k U_k^T A = A V_k V_k^T$.

Computing the SVD

Practical computation of the SVD requires the following steps:

1. Reduce matrix $A \in \mathbb{R}^{m \times n}$ to a bidiagonal form, i.e. find two orthogonal matrices U_B, V_B such that

$$U_B^T A V_B = B, \qquad B = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 \\ 0 & \alpha_2 & \beta_2 & 0 \\ 0 & 0 & \ddots & \ddots \\ 0 & 0 & 0 & \alpha_n \\ 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

2. Compute the SVD of matrix B, namely matrices U_2, V_2 and Σ such that

$$B = U_2 \Sigma V_2^T.$$

Then the final SVD of A will be

$$A = U_B U_D \Sigma V_D^T V_B^T.$$

First Step: reducing A to bidiagonal form

Dense (small) matrices

Bidiagonal matrix B is written as $U_B^T A V_B$ where U_B and V_B are expressed as a product of Householder matrices[2]

$$U_B = U_1 \cdot U_2 \dots U_n, \qquad V_B = V_1 \cdot V_2 \dots V_{n-2},$$

Example.

$$\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \xrightarrow{U_1^T} \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \xrightarrow{V_1} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \dots$$

$$\dots \xrightarrow{U_4^T} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

[2]Multiplying on the left A by a Householder matrix $H = I - 2\boldsymbol{w}\boldsymbol{w}^T$, for suitable $\boldsymbol{w}$, $\|\boldsymbol{w}\| = 1$ annihilates column elements below the diagonal, while multiplication on the right provides zeros in appropriate entries in a single row

Computing the SVD for large matrices

Iterative procedure for bidiagonalization: the Golub-Kahan-Lanczos method.

We need to find two matrices with orthonormal columns U_B, V_B such that $AV_B = U_B B$:

$$A\begin{bmatrix}\mathbf{v}_1 & \dots & \mathbf{v}_n\end{bmatrix} = \begin{bmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_n\end{bmatrix}\begin{bmatrix}\alpha_1 & \beta_1 & 0 & 0\\ 0 & \alpha_2 & \beta_2 & 0\\ & & \ddots & \ddots\\ 0 & 0 & \ddots & \ddots\\ 0 & 0 & 0 & \alpha_n\end{bmatrix} \tag{1}$$

from which

$$A\mathbf{v}_k = \alpha_k\mathbf{u}_k + \beta_{k-1}\mathbf{u}_{k-1} \implies \boxed{\alpha_k\mathbf{u}_k = A\mathbf{v}_k - \beta_{k-1}\mathbf{u}_{k-1}}, \qquad k = 1,\dots,n.$$

The bidiagonalization relation can also be written as $V_B^T A^T U_B = B^T$ or

$$A^T\begin{bmatrix}\mathbf{u}_1 & \dots & \mathbf{u}_n\end{bmatrix} = \begin{bmatrix}\mathbf{v}_1 & \dots & \mathbf{v}_n\end{bmatrix}\begin{bmatrix}\alpha_1 & 0 & 0 & 0\\ \beta_1 & \alpha_2 & \beta_2 & 0\\ & & \ddots & \ddots\\ 0 & 0 & \ddots & \ddots\\ 0 & 0 & \beta_{n-1} & \alpha_n\end{bmatrix} \tag{2}$$

from which

$$A^T\mathbf{u}_k = \alpha_k\mathbf{v}_k + \beta_k\mathbf{v}_{k+1} \implies \boxed{\beta\mathbf{v}_{k+1} = A^T\mathbf{u}_k - \alpha k\mathbf{v}_k}, \qquad k = 1,\dots,n.$$

The Golub-Kahan-Lanczos Algorithm

$$\begin{aligned} \alpha_k \boldsymbol{u}_k &= A\boldsymbol{v}_k - \beta_{k-1}\boldsymbol{u}_{k-1} \\ \beta_k \boldsymbol{v}_{k+1} &= A^T\boldsymbol{u}_k - \alpha k \boldsymbol{v}_k \end{aligned}$$

Imposing that $\|\boldsymbol{u}_k\| = 1$ and $\|\boldsymbol{v}_{k+1}\| = 1$, we have that

$$\alpha_k = \|A\boldsymbol{v}_k - \beta_{k-1}\boldsymbol{u}_{k-1}\| \text{ and } \beta_k = \|A^T\boldsymbol{u}_k - \alpha_k\boldsymbol{v}_k\|.$$

Algorithm 1 The Golub-Kahan-Lanczos Algorithm

```
1: Choose an arbitrary unit norm vector $\boldsymbol{v}_1$, $\beta_0 = 0$.
2: for $k = 1, 2, \dots$ do
3:     $\boldsymbol{u}_k = A\boldsymbol{v}_k - \beta_{k-1}\boldsymbol{u}_{k-1}$
4:     $\alpha_k = \|\boldsymbol{u}_k\|$.
5:     $\boldsymbol{u}_k = \boldsymbol{u}_k/\alpha_k$.
6:     $\boldsymbol{v}_{k+1} = A^T\boldsymbol{u}_k - \alpha_k\boldsymbol{v}_k$
7:     $\beta_k = \|\boldsymbol{v}_{k+1}\|$.
8:     $\boldsymbol{v}_{k+1} = \boldsymbol{v}_{k+1}/\beta_k$.
9: end for
```

Convergence and computational cost

After at most n steps the scalar $\beta_n = 0$ and the algorithm stops with the exact bidiagonalization matrices.

However, the procedure is usually stopped earlier. When?

- At step k the following relations hold

$$\begin{aligned} AV_B^{(k)} &= U_B^{(k)} B^{(k)} \\ A^T U_B^{(k)} &= V_B^{(k)} {B^{(k)}}^T + \beta_k \mathbf{v}_{k+1} \mathbf{e}_1^T. \end{aligned}$$

  so $\beta_k < \varepsilon$ can be used as exit test.
- In that case the largest singular values of $B^{(k)}$ are good approximations to the largest singular values of A.

Computational cost.

	Case	Cost
Dense matrix	Full bidiagonalization	$\mathcal{O}(mn^2)$
Dense matrix	Incomplete bidiagonalization	$\mathcal{O}(kmn)$
Sparse matrix[3]	Incomplete bidiagonalization	$\mathcal{O}(kcn)$

G. Golub and C. Van Loan
Matrix Computations, 4th Edition
Johns Hopkins University Press, 2013

[3]We assume that the nonzeros of A are cn with $c \ll m$.

Second step. Computing the SVD of B

The most known method is the Golub-Reinsch which employs implicit shift-QR method

However a simpler approach is based on the following idea:

Assume B is square $k \times k$. The relation $B = U\Sigma V^T$ can be exploited to obtain the following eigenproblem:

$$\underbrace{\begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix}}_{H} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

So the problems simplifies to computing the eigenvalues of H (which is better conditioned than $B^T B$) through the following steps:

- Permute rows and columns of H to obtain a tridiagonal matrix $T = P^T HP$ with the same eigenvalues of H.
- Compute the eigenvalues and eigenvectors of the $2k \times 2k$ by decomposing T as

  $$T = FDF^T,$$

  where the first k entries of D are the k singular values while the singular vector of B can be extracted from F. In fact from $D = F^T P^T HPF$ we have that the eigenvectors of H are the column of PF. Finally, setting $Z = PF$:

  $$V = Z(1\text{:k},1\text{:k}), \qquad U = Z(\text{k+1:2k, 1:k}).$$

Computing the SVD from the Polar Decomposition

Represents a new line of research that exploits the availability of efficient algorithms recently proposed to compute the Polar decomposition of a matrix.

Polar decomposition

Any rectangular matrix $A \in \mathbb{C}^{m\times n}(m \geq n)$ has a polar decomposition (PD)

$$A = Q_p H,$$

where $Q_p \in \mathbb{C}^{m\times n}$ is a (tall) matrix with orthogonal columns and $H \in \mathbb{C}^{n\times n}$ is Hermitian positive semidefinite.

The SVD of A can be obtained by further computing the eigendecomposition of H, .i.e,

$$A = Q_p(V\Lambda V^*) = (Q_p V)\Lambda V^* := U\Lambda V^*.$$

See for instance:

N. J. Higham and P. Papadimitriou
A parallel algorithm for computing the singular value decomposition
In J. G. Lewis, editor, The Fifth SIAM Conference on Applied Linear Algebra, pages 8084, Philadelphia, 1994. SIAM.

Computing the SVD from the Polar Decomposition

Advantages:

The SVD problem is reduced to an eigenvalue problem via the polar decomposition, therefore it can take benefit of well-developed scalable eigensolvers.

Main drawback:

It requires much more floating point operations than the bidiagonal reduction approach.

The remaining problem is how to compute the polar decomposition in a scalable and efficient way. Very important recent works in this direction:

Y. Nakatsukasa and R. W. Freund.
Computing fundamental matrix decompositions accurately via the matrix sign function in two iterations: The power of Zolotarev's functions.
SIAM Review, 2016.

Ltaief, Hatem and Sukkari, Dalal and Esposito, Aniello and Nakatsukasa, Yuji and Keyes, David.
Massively Parallel Polar Decomposition on Distributed-Memory Systems,
ACM Trans. Parallel Comput., 6(1), 2019.
https://doi.org/10.1145/3328723,doi 10.1145/3328723.

Principal Component Analysis (PCA)

PCA is an important technique frequently used for **data analysis** and **dimensionality reduction** in many applications:

- Quantitative finance
- Image Compression
- Facial Recognition
- Medical Data correlation
- Neuroscience
- and others

It is strictly related with the SVD.

Principal component analysis (PCA) is a statistical procedure that uses an **orthogonal transformation** to convert a set of observations of possibly correlated variables into a set of values of **linearly uncorrelated variables called principal components**.

This is done by finding the directions (principal components) in which the projected data display **maximum variance**.

Principal Component Analysis (PCA)

Let us consider our matrix $A \in \mathbb{R}^{m \times n}$ columnwise. Each column $\boldsymbol{a}^i, i = 1, \dots n$ is a **sample** of m variables. Usually $m < n$ (sometimes $m \ll n$).

Example. Consider samples describing height (in cms) and weight (in Kgs) of 10 men stored in a matrix A as

$$\begin{array}{c c c}
 & \textit{SAMPLES} & & \\
A = & \begin{bmatrix} 178 & 190 & 169 & 162 & 177 & 178 & 193 & 190 & 188 & 184 \\ 78 & 84 & 65 & 68 & 78 & 85 & 80 & 88 & 77 & 76 \end{bmatrix} & \begin{matrix} x_1 \\ x_2 \end{matrix} & \text{variables}
\end{array}$$

First compute the means of the n rows and collect them as a column vector:

$$\boldsymbol{\mu}_A = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{a}^i,$$

then define a new matrix B whose columns are $\boldsymbol{b}^i = \boldsymbol{a}^i - \boldsymbol{\mu}_A, \qquad i = 1 \dots, n.$

Each row of the *centered* matrix B has zero mean.

In our example

$$B = \begin{bmatrix} -2.9 & 9.1 & -11.9 & -18.9 & -3.9 & -2.9 & 12.1 & 9.1 & 7.1 & 3.1 \\ 0.1 & 6.1 & -12.9 & -9.9 & 0.1 & 7.1 & 2.1 & 10.1 & -0.9 & -1.9 \end{bmatrix}$$

Principal Component Analysis

We now define the **covariance matrix** which measures the correlation among the samples as

$$S = \frac{1}{n-1} B B^T = \left(\frac{1}{\sqrt{n-1}} B \right) \left(\frac{1}{\sqrt{n-1}} B \right)^T \equiv \hat{B} \hat{B}^T$$

The meaning of S is that S_{ii} represents the **variance** of **variable** i while $S_{ij}, i \neq j$ has the meaning of the **covariance** of the i-th and the j-th variables.

The **total variance** of the dataset is $\tau = \text{tr}(S) = \sum_{i=1}^{m} s_{ii}$.

τ can also be expressed using the Frobenius norm of $\hat{B}$ as $\tau = \text{tr}(\hat{B}\hat{B}^T) = \|\hat{B}\|_F^2$.

In our example:

$$S = \begin{bmatrix} 100.322 & 53.322 \\ 53.322 & 51.433 \end{bmatrix} \qquad \tau = 151.7556.$$

Principal Component Analysis

We denote as $\sigma_{\boldsymbol{v}}^2$ the variance along a vector $\boldsymbol{v}$ defined as:

$$\sigma_{\boldsymbol{v}}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{b}^{i\,T}\boldsymbol{v})^2$$

The main idea of the PCA is to find the vector which **maximize the variance $\sigma_{\boldsymbol{v}}^2$.**

$$\begin{aligned}\sigma_{\boldsymbol{v}}^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{b}^{i\,T}\boldsymbol{v})^2 = \frac{1}{n-1}\sum_{i=1}^{n}\boldsymbol{v}^T(\boldsymbol{b}^i\boldsymbol{b}^{i\,T})\boldsymbol{v} = \\ &= \frac{1}{n-1}\boldsymbol{v}^T\left(\sum_{i=1}^{n}(\boldsymbol{b}^i\boldsymbol{b}^{i\,T})\right)\boldsymbol{v} = \boldsymbol{v}^T S\boldsymbol{v}.\end{aligned}$$

Since S is symmetric then the maximum value of $\boldsymbol{v}^T S\boldsymbol{v}$ is the largest eigenvalue of S (if we choose $\boldsymbol{v}$ of unit Euclidean norm) and $\boldsymbol{v} = \boldsymbol{v}_1$ is the corresponding eigenvector.

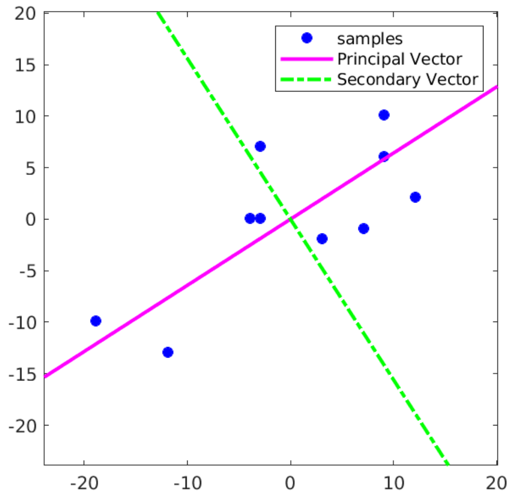
The fraction of the variance along this vector is $\dfrac{\sigma_1^2}{\tau}$.

Regarding the previous example we find that

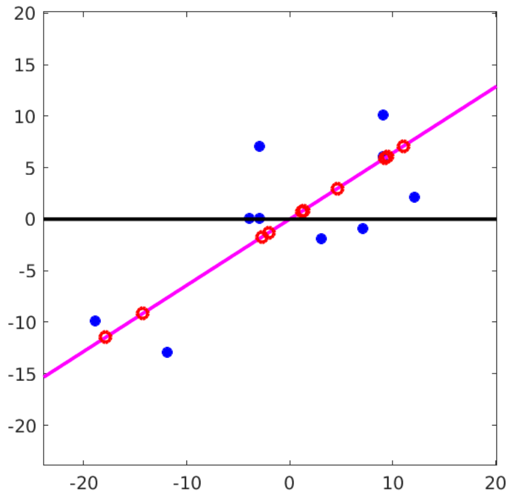
$$\boldsymbol{v}_1 = \begin{bmatrix} -0.8416 \\ -0.54 \end{bmatrix} \qquad {\sigma_1}^2 = 134.5356, \qquad \frac{\sigma_1^2}{\tau} = 0.8865.$$

Principal Component Analysis

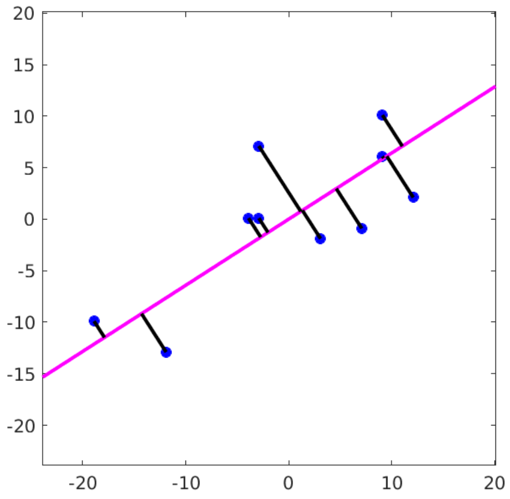
Geometrically



Samples (blue stars) and Principal components



Samples and their projections onto the Principal components



Samples and their distances from the Principal components

The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data.

The i-th principal component can be seen as a direction orthogonal to the first $i-1$ principal components that maximizes the variance of the projected data.

Principal component analysis and SVD

The **principal components** are the eigenvectors of the covariance matrix S related to the largest eigenvalues.

These are precisely the **right singular vectors** of $\hat{B}$ corresponding to the largest singular values $\sigma_1, \dots, \sigma_k$, which can be obtained from the SVD of **matrix** $\hat{B}$ (since $S = \hat{B}^T \hat{B}$):

$$\hat{B} = U \Sigma V^T, \qquad \text{which is the same as} \qquad S = V \Sigma^2 V^T,$$

where the first columns of V (V_k) are the k **principal components**.

If a small number (say k) of the largest eigenvalues are significantly larger than the others then the size of the original problem can be reduced to a problem of dimension k.

After forming the matrix $V_k = \begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_k \end{bmatrix}$ of the first k principal components we could obtain the PCA projection of the initial data using the truncated SVD

$$\tilde{A}_k = U_k U_k^T A = A V_k V_k^T$$

We can finally relate the measure of the efficiency of the approximation, i.e. the fraction of the total variance, using the singular values of $\hat{B}$:

$$\frac{\sum_{i=1}^k \sigma_i^2}{\tau} = \frac{\sum_{i=1}^k \sigma_i^2}{\sum_{i=1}^n \sigma_i^2} = 1 - \frac{\sum_{i=k+1}^n \sigma_i^2}{\sum_{i=1}^n \sigma_i^2}.$$

Trying it out in practice

See: example_svd.m

Limits of the SVD for data science

- Principal Component Analysis (PCA) and more generally the SVD are fundamental data analysis tools that express a data matrix in terms of a sequence of orthogonal (or uncorrelated) vectors of decreasing importance.

- Unfortunately, being linear combinations of almost all the data points, these vectors are notoriously difficult to interpret in terms of the data and processes generating the data.

  As an example, if all columns of A are non negative, the singular vectors can have negative components.

- The desire for interpretability in data analysis is sufficiently strong so as to argue for Interpretable low-rank matrix decompositions

  Decompositions belonging to this class are:

  - The CUR decomposition
  - The nonnegative matrix factorization (NMF)

The CUR decomposition

Is an approximate low-rank factorization which is **explicitly expressed in terms of a small number of columns and rows of the original matrix**.

Given $A \in \mathbb{R}^{m \times n}$ of rank r we can reconstruct the matrix as:

$$A = CUR$$

by choosing $C \in \mathbb{R}^{m \times r}$ with r columns of A, $R \in \mathbb{R}^{r \times n}$ with r rows of A, selected such that the intersection matrix $W \in \mathbb{R}^{r \times r}$ is nonsingular and $U = W^{-1}$.

When the number of selected columns k *is less than the actual rank* this decomposition provides a low-rank approximation of A.

The extent to which $A \approx CUR$ will depend sensitively on the choice of C and R (Column Subset Selection Problem – CSSP) as well as on the construction of U.

The low-rank approximation provided by CUR is often nearly as good as that provided by the SVD, but it is directly interpretable.

CUR matrices C and R inherit the sparsity of the original data matrix (much less memory is needed than for storing the SVD orthogonal matrices).

Column subset selection problem (CSSP)

Definition (CSSP): Given a matrix $A_{m\times n}$ and a positive integer k as the number of columns of A forming a matrix $C \in \mathbb{R}^{m\times k}$ such that the residual $\|A - P_C A\|_\xi$ is minimized over all possible $\begin{pmatrix} n \\ k \end{pmatrix}$ choices for the matrix C.

Here $P_C = CC^+$ (C^+ is Moore-Penrose pseudoinverse of C) denotes the projection onto the k dimensional space spanned by the columns of C and $\xi = 2 \; or \; F$.

Finding k columns out of n columns such that $\|A - P_C A\|_\xi$ is minimum is a hard optimization problem, requiring $O(n^k)$ time (prohibitively high if the data size is large).

The NP-hardness of the CSSP (assuming k is a function of n) is an open problem.

Christos Boutsidis, Michael W Mahoney, Petros Drineas
An improved approximation algorithm for the column subset selection problem
Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms, 2007

Research is focused on computing approximate solutions to CSSP.

Error. Let A_k be the best low rank k approximation. The most common algorithms in the literature select k columns of A such that matrix C satisfies, for some function $p(k, n)$,

$$\|A - A_k\|_\xi \le \|A - P_C A\|_\xi \le p(k,n)\,\|A - A_k\|_\xi \,.$$

Historical Evolution of the CUR matrix decomposition

1. G. W. Stewart developed the quasi-Gram-Schmidt method and applied it to a matrix and its transpose to obtain a CUR decomposition.

   G. W. Stewart
   Four algorithms for the the efficient computation of truncated pivoted QR approximations to a sparse matrix
   Numerische Mathematik, 1999

   Berry, M. W. and Pulatova, S. A. and Stewart, G. W.
   Algorithm 844: Computing Sparse Reduced-Rank Approximations to Sparse Matrices,
   ACM Transaction on Numerical Software, 2005.

2. Goreinov, Tyrtyshnikov and Zamarashkin developed a CUR matrix decomposition (*pseudoskeleton approximation*) and related the choices of columns and rows to a "maximum uncorrelatedness" concept.

   Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L.
   A theory of pseudoskeleton approximations
   Linear Algebra and its Applications, 1997

3. Drineas, Kannan and Mahoney constructed a CUR decomposition by choosing columns and rows simultaneously.

   Drineas, P., Kannan, R., Mahoney, M.W.
   Fast Monte Carlo Algorithms for Matrices III: Computing a Compressed Approximate Matrix Decomposition
   SIAM Journal on Computing, 2006

A practical CUR algorithm

We now describe a possible CUR implementation from

Mahoney, M.W. and Drineas, P.
CUR matrix decompositions for improved data analysis
Proceedings of the National Academy of Sciences, 2009

Leverage Score CUR (LSCUR)

- For all columns of A (and A^T) an *importance score* is computed which aims at capturing the influence of a given column in the best low-rank fit of the data matrix.
- It is proved that the obtained CUR factorization is nearly good as A_k.

How to compute the score?

Recall, from the SVD theorem, that $A = U\Sigma V^T$, the j-th column of A (A^j) can be written as a linear combination of columns of U_k in terms of the j-th row of V as

$$A^j = U\Sigma \mathbf{v}^j = \sum_{i=1}^{r} (\sigma_i \mathbf{u}_i) v_i^j \approx \sum_{i=1}^{k} (\sigma_i \mathbf{u}_i) v_i^j,$$

with $k \ll r$ (truncated SVD).

From this relation we see that each $\sigma_i \mathbf{u}_i$ is weighted by the corresponding component v_i^j of the j-th right singular vector.

Interpretation of the LSCUR

Recall that the rank-k truncated SVD has the following repesentation

$$\underbrace{\left[\begin{array}{c|c|c} \quad & a_1^j & \qquad \\ & a_2^j & \\ & a_3^j & \\ & \vdots & \\ & \vdots & \\ & a_m^j & \end{array}\right]}_{A} \approx \underbrace{\left[\begin{array}{c} \quad \\ \\ \\ \\ \\ \\ \end{array}\right]}_{U_k} \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}}_{\Sigma_k} \underbrace{\left[\begin{array}{c|c|c} \quad & v_1^j & \qquad \\ & \cdots & \\ & v_k^j & \end{array}\right]}_{V_k^T}$$

$$A^j = U\Sigma \boldsymbol{v}^j = \sum_{i=1}^{r} (\sigma_i \boldsymbol{u}_i) v_i^j \approx \sum_{i=1}^{k} (\sigma_i \boldsymbol{u}_i) v_i^j,$$

Leverage Score CUR (LSCUR)

The *Normalized statistical leverage score* is then computed as

$$\pi_j = \frac{1}{k}\sum_{i=1}^{k}(v_i^j)^2, \qquad j = 1, \dots, n \tag{3}$$

The set $\{\pi_j\}_{j=1}^n$ forms a probability distribution over columns of A as $\pi_j \geq 0$ and $\sum_{i=1}^n \pi_j = 1$.

The number c of columns in C must be chosen as $c = \mathcal{O}(k \log k/\varepsilon^2)$. In this way it can be proved that

$$\|A - CUR\|_F \leq (2+\varepsilon)\|A - A_k\|_F, \qquad \textit{holds with high probability}.$$

The strategy to select the c columns is described in the following

Algorithm 2 ColumnSelect(A)

```
1: Input: $A \in \mathbb{R}^{m\times n}, c, k$
2: Output: $C \in \mathbb{R}^{m\times c}$
3: Compute the truncated SVD of $A = U\Sigma V^T$ and define $\pi_j, j = 1, \dots, k$ using (3)
4: for $i = 1 : c$ do
5:     Pick $j \in \{1, \dots, n\}$ with probability $\min(1, c\pi_j)$
6:     Set $C(:, i) = A(:, j)/\sqrt{c^2\pi_j}$
7: end for
```

Leverage Score CUR (LSCUR)

The previous algorithm must be embedded within the construction of C, R and U as described below:

Algorithm 3 LeverageScore CUR

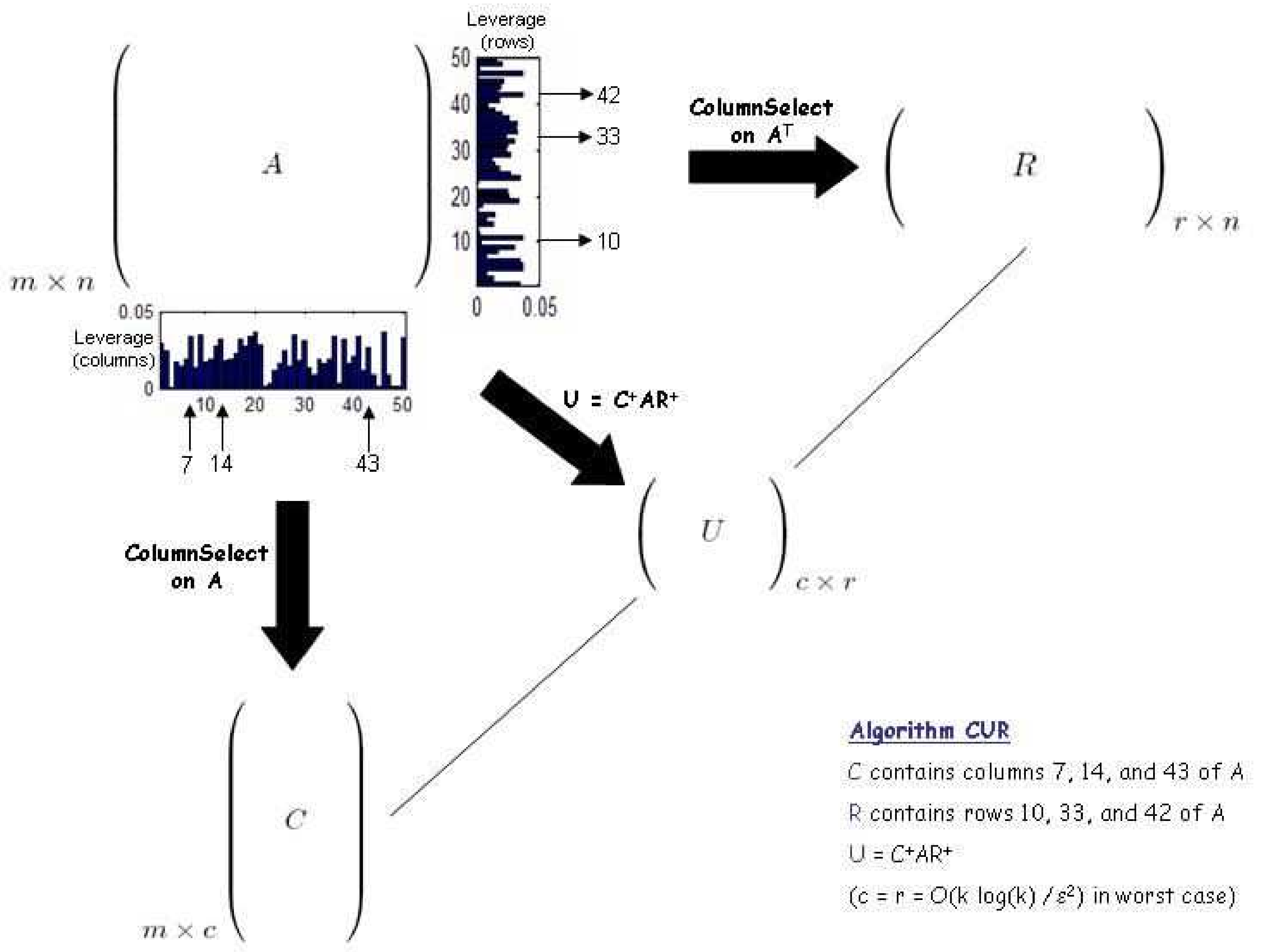
```
1: **Input**: $A \in \mathbb{R}^{m\times n}, c, r, k$
2: **Output**: $C \in \mathbb{R}^{m\times c}, U \in \mathbb{R}^{c\times r}, R \in \mathbb{R}^{r\times n}$,
3: Invoke function ColumnSelect($A$) with $c = \mathcal{O}(k \log k/\varepsilon^2)$ to construct matrix $C$.
4: Invoke function ColumnSelect($A^T$) with $r = \mathcal{O}(k \log k/\varepsilon^2)$ to construct matrix $R$.
5: Set matrix $U$ as $C^+AR^+$
```

Recall that the pseudoinverse of a rectangular full rank matrix $A \in \mathbb{R}^{m\times n}$ is defined as

$$\begin{array}{lclcl} A^+ & = & (A^TA)^{-1}A^T, & \text{if} \quad m > n & (A^+A = I_m) \\ A^+ & = & A^T(AA^T)^{-1}, & \text{if} \quad n > m & (AA^+ = I_n) \end{array}$$

which satisfies $A^+A = I_n$.

Schematic view of algorithm LSCUR



Algorithm CUR

C contains columns 7, 14, and 43 of A

R contains rows 10, 33, and 42 of A

$U = C^+AR^+$

($c = r = O(k \log(k) / \varepsilon^2)$ in worst case)

Nyström Method

Nyström methods are used for approximating symmetric positive semidefinite matrices (SPSD)

The Nyström method approximates a matrix using only a subset of its columns, selected by different sampling techniques.

Let $A \in \mathbb{R}^{n\times n}$ be an SPSD matrix. Let $C_{n\times m}$ be a matrix consisting of m $(\ll n)$ randomly selected columns of A as columns.
Now, the matrix A can be rearranged such that C and A are written as

$$C = \left[\begin{array}{c} W \\ S \end{array}\right] \; and \;\; A = \left[\begin{array}{cc} W & S^T \\ S & B \end{array}\right],$$

where $W \in \mathbb{R}^{m\times m}$, $S \in \mathbb{R}^{(n-m)\times m}$ and $B \in \mathbb{R}^{(n-m)\times(n-m)}$.

Computing a truncated SVD (partial spectral decomposition) of W, $W_k = U_k \Sigma_k U_k^T$ for k $(k \le m)$, the rank$-k$ Nyström approximation is defined by

$$\widetilde{A}_k = C W_k^\dagger C^T, \qquad W_k^\dagger = \sum_{i=1}^{k} \sigma_i^{-1} \boldsymbol{u}_i \boldsymbol{u}_i^T,$$

where $(\boldsymbol{u}_i, \sigma_i)$ are the first k singular vectors and values of W $(V \equiv U$ as W is symmetric positive semidefinite as $A)$.

Nyström Method

The approximation quality of the Nyström approximation highly depends on the sampling strategy used for column selection.

The simplest Nyström-based procedure selects columns from the original data set uniformly at random and then uses those columns to construct a low-rank SPSD approximation.

However, nonuniform sampling strategies can lead to lower reconstruction error for a fixed number of column samples, both in theory and in practice.
When $\Omega(k\epsilon^{-4} \ln \delta^{-1})$ columns are sampled with an importance sampling distribution that is proportional to the square of the diagonal entries of A, then

$$\left\| A - CW_k^{\dagger} C^T \right\|_{\xi} \leq \| A - A_k \|_{\xi} + \epsilon \sum_{i=1}^{n} A_{ii}^2 .$$

holds with probability $1 - \delta$, where $\xi = 2, F$ represents the Frobenius or spectral norm.

Petros Drinneas and Michael W. Mahoney
On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning
Journal of Machine Learning Research 6 (2005) 2153–2175

Cross Approximation

A recent approach for constructing a low-rank approximation of a given matrix A is connected with the problem of finding a submatrix of A of a given dimension k having **maximum volume** i.e. which **maximizes the absolute value of the determinant**.

Let us denote a submatrix of A as $A(I, J)$ with $I, J \subset \{1, \dots, n\}$.

The rank-k Cross Approximation of A is

$$A(:, J)A(I, J)^{-1}A(I, :)$$

where I and J should maximize $|\det A(I, J)|$ and $|I|, |J| = k$ ($|.|$ denotes cardinality of a set).

If $A(I, J)$ has maximum volume then

$$\|A - A(:, J)A(I, J)^{-1}A(I, :)\|_{\max} \leq (k + 1)\sigma_{k+1}(A), \tag{4}$$

where $\| \cdot \|_{\max}$ denotes the maximum absolute value of the entries of a matrix.

S. A. Goreinov and E. E. Tyrtyshnikov.
The maximal-volume concept in approximation by low-rank matrices.
In *Structured matrices in mathematics, CS and Engineering*. Contemporary Mathematics, *2001*

Unfortunately, finding the submatrix of A of maximum volume is NP hard.

Cross Approximation: an important connection

An approximation of the form $A(:,J)A(I,J)^{-1}A(I,:)$ is closely connected to an incomplete LU decomposition of A.

To see this, suppose that A has been permuted such that $I = J = \{1,\dots,k\}$ and partition

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad A_{11} \in \mathbb{R}^{k\times k}.$$

Assume that A_{11} is invertible and admits an LU decomposition $A_{11} = L_{11}U_{11}$, where L_{11} is lower triangular and U_{11} is upper triangular with ones on the diagonal. By setting $L_{21} = A_{21}U_{11}^{-1}$ and $U_{12} = L_{11}^{-1}A_{12}$, we obtain

$$\begin{aligned} A &= A(:,J)A(I,J)^{-1}A(I,:) + \begin{bmatrix} 0 & 0 \\ 0 & A^{(k)} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & A^{(k)} \end{bmatrix} && (5) \\ &= \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & A^{(k)} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I \end{bmatrix}, && (6) \end{aligned}$$

with the Schur complement

$$A^{(k)} := A_{22} - A_{21}A_{11}^{-1}A_{12}.$$

This shows that the approximation error is governed by $A^{(k)}$. The factorized form (5) corresponds exactly to what is obtained after applying k steps of the LU factorization to A.

Greedy algorithm for Cross Approximation

Given index sets I and J, one step of a greedy method for volume maximization consists of choosing indices such that

$$(i_{k+1}, j_{k+1}) = \arg\max \left\{ \left| \det\left(A(I \cup \{i\}, J \cup \{j\})\right) \right| : i \notin I, j \notin J \right\}.$$

Again, let us assume that $I = J = \{1, \ldots, k\}$ and set $\tilde{I} = I \cup \{k + \tilde{i}\}$, $\tilde{J} = J \cup \{k + \tilde{j}\}$ for some $\tilde{i}, \tilde{j} \in \{1, \ldots, n-k\}$. Then (6) implies

$$\det\left(A(\tilde{I}, \tilde{J})\right) = \det\left(A(I, J)\right) \cdot A^{(k)}(\tilde{i}, \tilde{j}).$$

So the local optimization problem is solved by searching the entry of $A^{(k)}$ that has maximum modulus.

This choice leads to an Algorithm which is equivalent to applying LU factorization with complete pivoting to A.

Alice Cortinovis, Daniel Kressner and Stefano Massei,
On maximum volume submatrices and cross approximation for symmetric semidefinite and diagonally dominant matrices,
Linear Algebra and its Applications, 2020

Cross Approximation: Algorithm

The algorithm to compute the subsets I and J yielding an m-rank approximation of A is given below:

Algorithm 4 Cross approximation with complete pivoting

```
1: Initialize $R_0 := A$, $I := \{\}$, $J := \{\}$.
2: for $k = 0, \dots, m-1$ do
3:     $(i_{k+1}, j_{k+1}) := \arg\max_{i,j} |R_k(i,j)|$
4:     $I \leftarrow I \cup \{i_{k+1}\}$, $J \leftarrow J \cup \{j_{k+1}\}$
5:     $p_{k+1} := R_k(i_{k+1}, j_{k+1})$
6:     $R_{k+1} := R_k - \frac{1}{p_{k+1}} R_k(:, j_{k+1}) R_k(i_{k+1}, :)$
7: end for
```

The algorithm returns the sets I and J.

Matrix R_k is the remainder i.e.

$$R_k = A - A(:,J)A(I,J)^{-1}A(I,:) = \begin{bmatrix} 0 & 0 \\ 0 & A^{(k)} \end{bmatrix}$$

after a suitable permutation of the indices.

Error bounds

If the sets I, J are given by Algorithm 4 some error results are proved for special matrices:

- **Diagonally dominant matrices**

$$\|A - A(:,J)A(I,J)^{-1}A(I,:)\|_{\max} \leq (m+1)\cdot 2^{m+1}\cdot \sigma_{m+1}(A).$$

- **Symmetric Positive Semidefinite Matrices**

$$\|A - A(:,J)A(I,J)^{-1}A(I,:)\|_{\max} \leq 4^{m}\cdot \sigma_{m+1}(A).$$

Remark: the bound of the error for general matrices involves also the growth factor of the Gaussian Elimination algorithm with complete pivoting.

More recent results for SPSD matrices in:

Stefano Massei
Some algorithms for maximum volume and cross approximation of symmetric semidefinite matrices,
arxiv: 2007.04858, 2020

Nonnegative matrix factorization

- Nonnegative matrix factorization (NMF) has become a widely used tool for the analysis of high-dimensional data.
- It automatically extracts **sparse and meaningful features** from a set of nonnegative data vectors since it computes a rank-k approximation of a given data matrix $A \in \mathbb{R}^{m\times n}$ that is constrained to have nonnegative factors.
- Unfortunately, the problem of solving NMF is **NP-hard** in general.
- Standard NMF algorithms approximate this factorization **numerically**.
- Due to the many important applications of nonnegative matrix factorizations, algorithm development is an active area of research.

Nonnegative matrix factorization

Given a data matrix $A \in \mathbb{R}^{m \times n}$ NMF aims at decomposing it as

$$A \approx WH$$

where $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$ and $W \geq 0$, $H \geq 0$ (meaning that W and H are component-wise nonnegative).

NMF was first introduced in 1994 by Paatero and Tapper in:

Paatero, P., Tapper, U.:
Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values.
Environmetrics **5**, 111–126 (1994)

but it gathered more and more interest after this article by Lee and Seung in 1999:

Lee, D., Seung, H.:
Learning the Parts of Objects by Nonnegative Matrix Factorization.
Nature **401**, 788–791 (1999)

Since then, the number of publications referencing the technique has grown rapidly.

Nonnegative matrix factorization

Topic Recovery and Document Classification

The reason why NMF has become so popular is **because of its ability to automatically extract sparse and easily interpretable factors**.

Let us illustrate this property of NMF through an application in **text mining**:

Let each column of the nonnegative data matrix $A \in \mathbb{R}^{m \times n}$ correspond to a document and each row to a word.

The $(i, j)-$th entry of the matrix A indicates the number of times the $i-$th word appears in the $j-$th document[4]

Note that matrix A is, in general, rather sparse (most documents only use a small subset of the dictionary).

Given such a matrix A and a factorization rank r, NMF generates two factors (W, H) such that, for all $1 \leq j \leq n$, we have

$$\underbrace{A(:,j)}_{j\text{th document}} \approx \sum_{k=1}^{r} \underbrace{W(:,k)}_{k\text{th topic}} \underbrace{H(k,j)}_{\substack{\text{importance of } k\text{th topic} \\ \text{in } j\text{th document}}}, \quad \text{with } W \geq 0 \text{ and } H \geq 0.$$

[4]This is the so-called **bag-of-words model**: each document is associated with a set of words with different weights, while the ordering of the words in the documents is not taken into account.

Nonnegative matrix factorization

Topic Recovery and Document Classification

The **basis vectors** (columns of W) can be interpreted as *topics*, that is, set of words found simultaneously in different documents, while the weights in the linear combinations (that is, the matrix H) assign the documents to the different topics, that is, identify which document discusses which topic.

Therefore, given a set of documents, *NMF identifies topics and simultaneously classifies the documents among these different topics.*

Note: NMF is closely related to probabilistic latent semantic analysis and indexing (PLSA and PLSI)

Gaussier, E., Goutte, C.:
Relation between PLSA and NMF and implications.
Proc. 28th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005

Ding, C., Li, T., Peng, W.:
On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing.
Computational Statistics & Data Analysis **52**(8), 3913–3927 (2008)

NMF. Some bibliography and applications

- **image processing**

  Lee, D., Seung, H.:
  Learning the Parts of Objects by Nonnegative Matrix Factorization.
  Nature **401**, 788–791 (1999)

- **computational biology**

  Devarajan, K.:
  Nonnegative Matrix Factorization: An Analytical and Interpretive Tool in Computational Biology.
  PLoS Computational Biology **4**(7), e1000029 (2008)

- **clustering**

  Ding, C., He, X., Simon, H.:
  On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering.
  In: SIAM Int. Conf. Data Mining (SDM'05), pp. 606–610 (2005)

- **collaborative filtering**

  Melville, P., Sindhwani, V.:
  Recommender systems.
  Encyclopedia of machine learning **1**, 829–838 (2010)

- **community detection**

  Wang, F., Li, T., Wang, X., Zhu, S., Ding, C.:
  Community discovery using nonnegative matrix factorization.
  Data Min. Knowl. Disc. **22**(3), 493–521 (2011)

Implementing NMF

To obtain a rank r nonnegative matrix factorization, we have to solve the following optimisation problem:

$$\min_{\substack{W \in \mathbb{R}^{m \times r} \\ H \in \mathbb{R}^{r \times n}}} \|A - WH\|_F^2 \quad \text{subject to} \quad W \geq 0 \text{ and } H \geq 0. \tag{7}$$

There are many issues when using NMF in practice.

In particular, as opposed to the unconstrained problem which can be solved efficiently, **NMF is NP-hard** in general.

Hence, in practice, most algorithms are applications of standard nonlinear optimization methods and may only be guaranteed to converge to stationary points;

However, these heuristics have been proved to be successful in many applications.

Implementing NMF

Another important issue is that **NMF is ill-posed**:

Given an NMF (W, H) of A, there usually exist equivalent NMF's (W', H') with $W'H' = WH$.

In particular, any matrix Q satisfying $WQ \geq 0$ and $Q^{-1}H \geq 0$ generates such an equivalent factorization.

The matrix Q can always be chosen as the permutation of a diagonal matrix with positive diagonal elements (that is, as a monomial matrix) and this amounts to the scaling and permutation of the rank-one factors $W(:,k)H(k,:)$ for $1 \leq k \leq r$; this is not an issue in practice.
The issue is when there exist non-monomial matrices Q satisfying the above conditions:

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

In that case, such equivalent factorizations generate different interpretations!

For example, in text mining, they would lead to different topics and classifications.

In practice, this issue is tackled using other priors on the factors W and H and adding proper regularization terms in the objective function.

Nonnegative Matrix Factorizations. Practical Algorithms.

The minimization problem

$$\min \|A - WH\|_F^2 = \min \sum_{i=1}^{n} \|A^{(i)} - WH^{(i)}\|_2^2, \quad \text{subject to} \quad W, H \geq 0,$$

can be split into n independent problems

$$\sum_{i=1}^{n} \min_{\boldsymbol{h} \geq 0} \|\boldsymbol{a} - W\boldsymbol{h}\|_2,$$

involving columns of A and of H and assuming W known.

After obtaining an approximation of columns of H the same idea is applied by taking now H fixed to compute the rows of W.

This gives raise e.g. to the **Alternating Least Square (ALS)** method.

The Alternating Least Square method

Algorithm 5 Alternating Least Squares

```
1: Initialize $W$ and $H$
2: repeat
3:     Solve $\min_{H \geq 0} \frac{1}{2}\|A - WH\|_F^2$
4:     Solve $\min_{W \geq 0} \frac{1}{2}\|A - WH\|_F^2$
5: until stopping criterion
```

This algorithm requires excessive computing time to solve the subproblems.

It is usually replaced by an inexact LS solution by solving the unconstrained problem and projecting it in the space of nonnegative components.

Once again the matrix minimization problem is solved column by columns by solving a classical LS problem:

$$\begin{aligned} \min \|\mathbf{a} - W\mathbf{h}\|_2 \quad &\Longrightarrow \quad W^T W\mathbf{h} = W^T\mathbf{a} \\ &\Longrightarrow \quad W^T WH = W^T A \end{aligned}$$

This yields the algorithm on the next slide.

Inexact Alternating Least Squares Algorithm

Algorithm 6 Inexact Alternating Least Squares

```
1: Initialize $W$ and $H$
2: repeat
3:     Solve for $H$ the equation: $W^T WH = W^T A$;
4:     Set to zero all negative elements of $H$.
5:     Solve for $W$ the equation: $WHH^T = AH^T$;
6:     Set to zero all negative elements of $W$.
7: until stopping criterion
```

One of the possible alternative methods is represented by the **Gradient Descent** (see the second part of the course) which in short, given the multivariate function

$$F(\boldsymbol{h}) = \|\boldsymbol{a} - W\boldsymbol{h}\|_2^2$$

constructs a sequence $\boldsymbol{x}_0, \dots, \boldsymbol{x}_k, \dots$ defined as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \nabla F(\boldsymbol{x}_k), \qquad (\boldsymbol{x}_{k+1})_i = \max\{(\boldsymbol{x}_{k+1}), 0\},$$

where ∇F is the gradient of F and α_k is a scalar to be determined.

Algorithm development is an active area of research!

Example. NMF of the term-document matrix

Recall the term-document example.

Term	D1	D2	D3	D4	D5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
Web	0	1	1	0	0

$$\longrightarrow \qquad A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

A nonnegative factorization $A \approx WH$ can be used for clustering: the data vector $\boldsymbol{a}_j$ is assigned to cluster i if h_{ij} is the largest element in column j of H.

To obtain a NM Factorization we can invoke the Matlab function

```
[W,H] = nnmf(A,k)
```

which uses by default the Inexact ALS method. The second parameter is the rank of the factorization.

Example. Continued

```
>> help nnmf

   nnmf Non-negative matrix factorization.

   [W,H] = nnmf(A,K) factors the N-by-M matrix A into non-negative factors
  W (N-by-K) and H (K-by-M).

   The result is not an exact factorization, but W*H is a lower-rank
   approximation to the original matrix A.

   The W and H matrices are chosen to minimize the objective function that
   is defined as the root mean squared residual between A and the
         approximation W*H.
   This is equivalent to

            D = norm(A-W*H,'fro')/sqrt(N*M)

   The factorization uses an iterative method starting with random initial
   values for W and H.

   Because the objective function often has local
   minima, repeated factorizations may yield different W and H values.

   Sometimes the algorithm converges to solutions of lower rank than K,
   and this is often an indication that the result is not optimal.

   [W,H,D] = nnmf(...) also returns D, the root mean square residual.
```

Example. Continued

Using $k = 2$ we obtain:

$$W = \begin{bmatrix} 0.6415 & 0 \\ 0.3654 & 0 \\ 0.3654 & 0 \\ 0.9818 & 0.2521 \\ 0.5507 & 0 \\ 0 & 0.9162 \\ 1.6232 & 0.0331 \\ 0.1183 & 1.3563 \\ 1.4380 & 0.0963 \\ 0.1183 & 1.3563 \end{bmatrix} \qquad H = \begin{bmatrix} 0.4865 & 0 & 0.5813 & 0.5667 & 0.3228 \\ 0 & 0.8094 & 0.5873 & 0 & 0 \end{bmatrix}$$

It is now possible to interpret the decomposition. The first four documents are well represented by the basis vectors, which have large components for Google-related keywords.

In contrast, the 5-th document is represented by the first basis vector only, but its coordinates are smaller than those of the first four Google-oriented documents.

In this way, the rank-2 approximation accentuates the Google-related contents, while the football-document is de-emphasized.

Example. Continued

Choosing instead $k = 3$ yields

$$W = \begin{bmatrix} 0.3655 & 0 & 0.3074 \\ 0 & 0 & 0.9674 \\ 0 & 0 & 0.9674 \\ 1.2572 & 0.2061 & 0 \\ 0.8765 & 0 & 0 \\ 0 & 0.8957 & 0 \\ 1.6228 & 0.0507 & 0.1546 \\ 0.0926 & 1.3693 & 0.0093 \\ 0.5129 & 0.2855 & 1.3130 \\ 0.0926 & 1.3693 & 0.0093 \end{bmatrix} \qquad H = \begin{bmatrix} 0.7401 & 0 & 0.5466 & 0.3918 & 0 \\ 0 & 0.7974 & 0.6035 & 0 & 0 \\ 0 & 0 & 0.1879 & 0.3821 & 0.9048 \end{bmatrix}$$

We see that now the third vector in W is essentially a *football* basis vector, while the other two represent the Google-related documents.

Query matching in Text mining

Query matching is the process of finding the documents that are relevant to a particular query q.

Using a rank-k approximation the term-document matrix is represented by $A_k = W_k H_k$. Query matching is performed in this k-dimensional space.

We make use of the **cosine distance measure** (the cosines of the angles that the query form with the columns of the rank-k approximation of the data):
$\boldsymbol{q}^T A_k = \boldsymbol{q}^T W_k H_k = (W_k^T \boldsymbol{q})^T H_k = \boldsymbol{q}_k^T H_k$:

$$\cos(\theta_j) = \frac{\boldsymbol{q}_k^T \boldsymbol{h}_j}{||\boldsymbol{q}_k||_2 ||\boldsymbol{h}_j||_2}$$

A document a_j is deemed relevant if the angle between the query $\boldsymbol{q}$ and $\boldsymbol{a}_j$ is small enough.

For the example, for the query *eigenvalue* ($\boldsymbol{q} = (1,0,0,0,0,0,0,0,0,0)^T$) we obtained the following vector of cosine distances:

$$\begin{bmatrix} 0.7653 & 0 & 0.6453 & 0.9973 & 0.6437 \end{bmatrix}$$

indicating the document number 4 as the most relevant one.

Bibliography including some references on the extension to Tensors

N. K. Kumar and J. Schneider
Literature survey on low rank approximation of matrices
Linear and Multilinear Algebra, 65, 11, 2212–2244, 2017

Y. Koren, R. Bell and C. Volinsky,
Matrix Factorization Techniques for Recommender Systems,
Computer, vol. 42, no. 8, pp. 30-37, 2009

C. Boutsidis and D. P. Woodruff
Optimal CUR Matrix Decompositions
SIAM Journal on Computing 46(2) 543–589, 2017

E. C. Chi and T. Kolda
On tensors, sparsity and nonnegative factorizations,
SIAM Journal on Matrix Analisys and Applications 33, 2012

T. Kolda and B. W. Bader
Tensor decomposition and applications.
SIAM Review 51, 2009

A. Cichocki N. Lee I. Oseledets A. Phan Q. Zhao D. P. Mandi
Tensor Networks for Dimensionality Reduction and Large-Scale Optimization: Part 1 Low-Rank Tensor Decompositions
Foundations and Trends® in Machine Learning, 9, 249–429, 2016

Concluding remarks

We have seen:

- How information extraction from data and pattern finding relies on efficient techniques from numerical linear algebra that not only enable computations (reducing the dimensionality) but also reveal hidden information.
- Recent trends in data analysis via matrix factorizations. Beyond the use of the classical SVD we have introduced also interpretable factorizations such as the non-negative matrix factorization (NMF) or CUR decompositions.

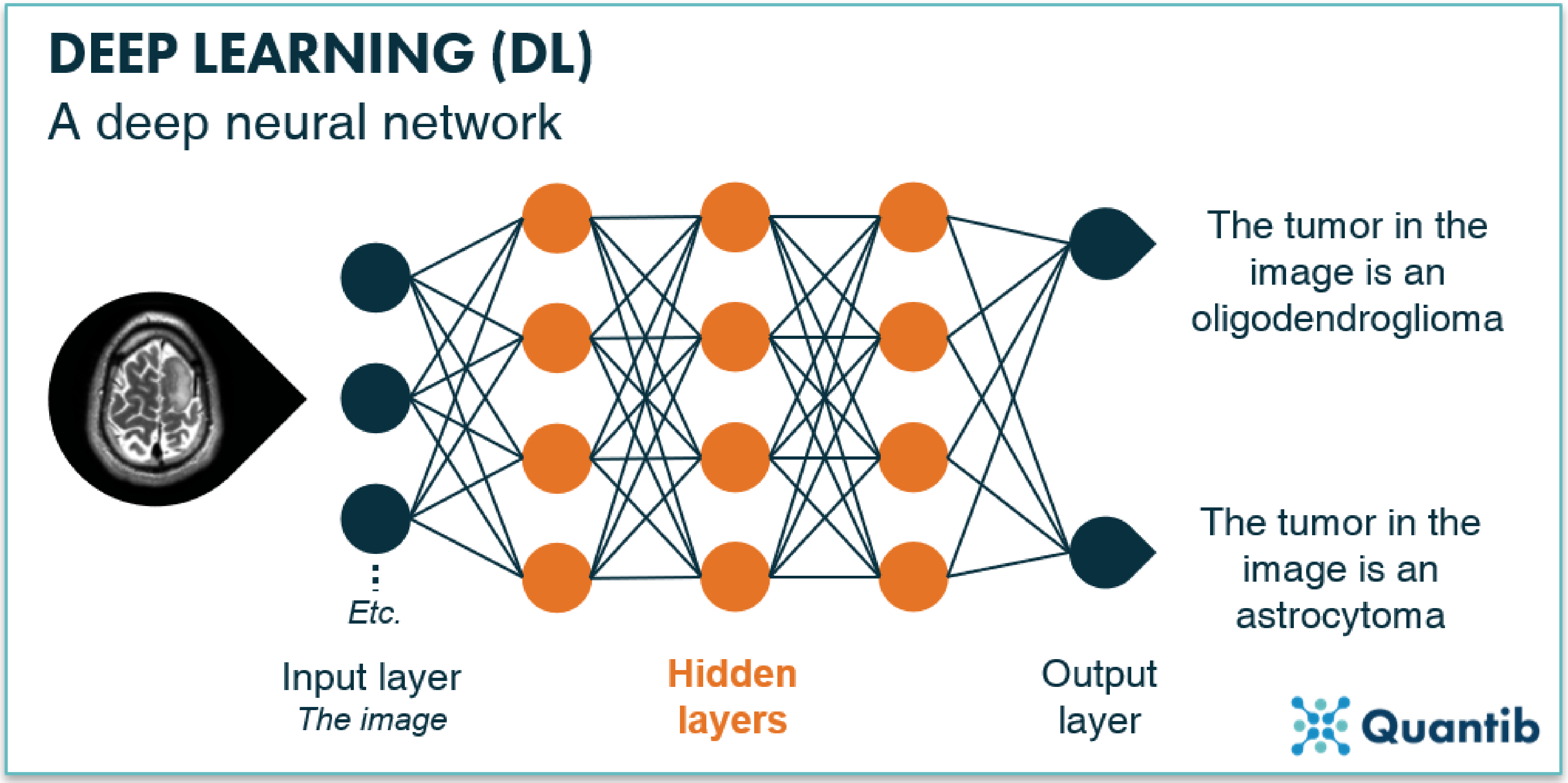
For a very vast bibliographical survey on NLA for Data Science see the recent work:

Martin Stoll
A literature survey of matrix methods for data science
GAMM Mitteilungen, 43(3) Special Issue:Applied and Numerical Linear Algebra Part I, 2020.

Beautiful books that provide general introductions to linear algebra for data science applications are:

Lars Eldén	Matrix Methods in Data Mining and Pattern Recognition. Philadelphia: SIAM, 2019.
Gilbert Strang	Linear Algebra and learning from data, Wellesley-Cambridge Press, 2019.

Next lecture



We will dive into the fascinating world of Deep Learning!