Réduction de dimensionnalité

Sergey Kirgizov

ESIREM

Plan

Algèbre linéaire et problèmes d'optimisation

- Matrices
- Multiplication
- Norme, produit scalaire
- Valeurs et vecteurs propres
- Quotient de Rayleigh
- Théorème min-max de Courant-Fischer

Analyse en composantes principales

- Matrice de données
- Algorithmes de calcul

Notations

Nous utilisons :

- des lettres majuscules en gras ($\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$) pour désigner des matrices
- des lettres minuscules en gras ($\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$) pour les vecteurs
- des lettres minuscules ($a, b, c, \cdots, \alpha, \beta, \gamma, \cdots \lambda \cdots$) pour les scalaires.

Vecteurs

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{pmatrix}$$

$$\mathbf{v}^T = \begin{pmatrix} v_1 & v_2 & v_3 & \dots & v_n \end{pmatrix}$$

Matrices

$$\mathbf{A}_{n\times k} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,k} \end{pmatrix} = \mathbf{A}$$

La dimension de A est (n, k)

Matrices

$$\mathrm{A}_{n\times k} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,k} \end{pmatrix} = \mathrm{A}$$

La dimension de A est (n, k)

Operations

Transposition $\mathrm{C} = \mathrm{A}^T \Leftrightarrow c_{j,i} = a_{j,i}$

Addition $\mathrm{A}_{n\times k} = \mathrm{B}_{n\times k} + \mathrm{C}_{n\times k} \Leftrightarrow a_{i,j} = b_{i,j} + c_{i,j}$

Multiplication par un scalaire $\mathrm{B} = c\mathrm{A} \Leftrightarrow b_{i,j} = ca_{i,j}$

Multiplication $\mathrm{C}_{n\times m} = \mathrm{A}_{n\times k}\mathrm{B}_{k\times m} \Leftrightarrow c_{i,j} = \sum_{p=1}^{k} a_{i,p}b_{p,j}$

Attention aux dimensions !

Multiplication

Matrice ligne **L** par une matrice colonne **C**

$$\mathbf{LC} = \begin{pmatrix} x_1 & \dots & x_k \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix} = \sum_{i=1}^{k} x_i y_i$$

Multiplication

Matrice ligne $\mathbf{L}$ par une matrice colonne $\mathbf{C}$

$$\mathbf{LC} = \begin{pmatrix} x_1 & \dots & x_k \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix} = \sum_{i=1}^{k} x_i y_i$$

Cas général

$\mathbf{L}_i$: les lignes de la première matrice

$\mathbf{C}_j$: les colonnes de la deuxième

$$\begin{pmatrix} \boxed{\mathbf{L}_1} \\ \vdots \\ \boxed{\mathbf{L}_m} \end{pmatrix} \begin{pmatrix} \boxed{\mathbf{C}_1} & \dots & \boxed{\mathbf{C}_n} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_1\mathbf{C}_1 & \mathbf{L}_1\mathbf{C}_2 & \dots & \mathbf{L}_1\mathbf{C}_n \\ \mathbf{L}_2\mathbf{C}_1 & \mathbf{L}_2\mathbf{C}_2 & \dots & \mathbf{L}_2\mathbf{C}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{L}_m\mathbf{C}_1 & \mathbf{L}_m\mathbf{C}_2 & \dots & \mathbf{L}_m\mathbf{C}_n \end{pmatrix}$$

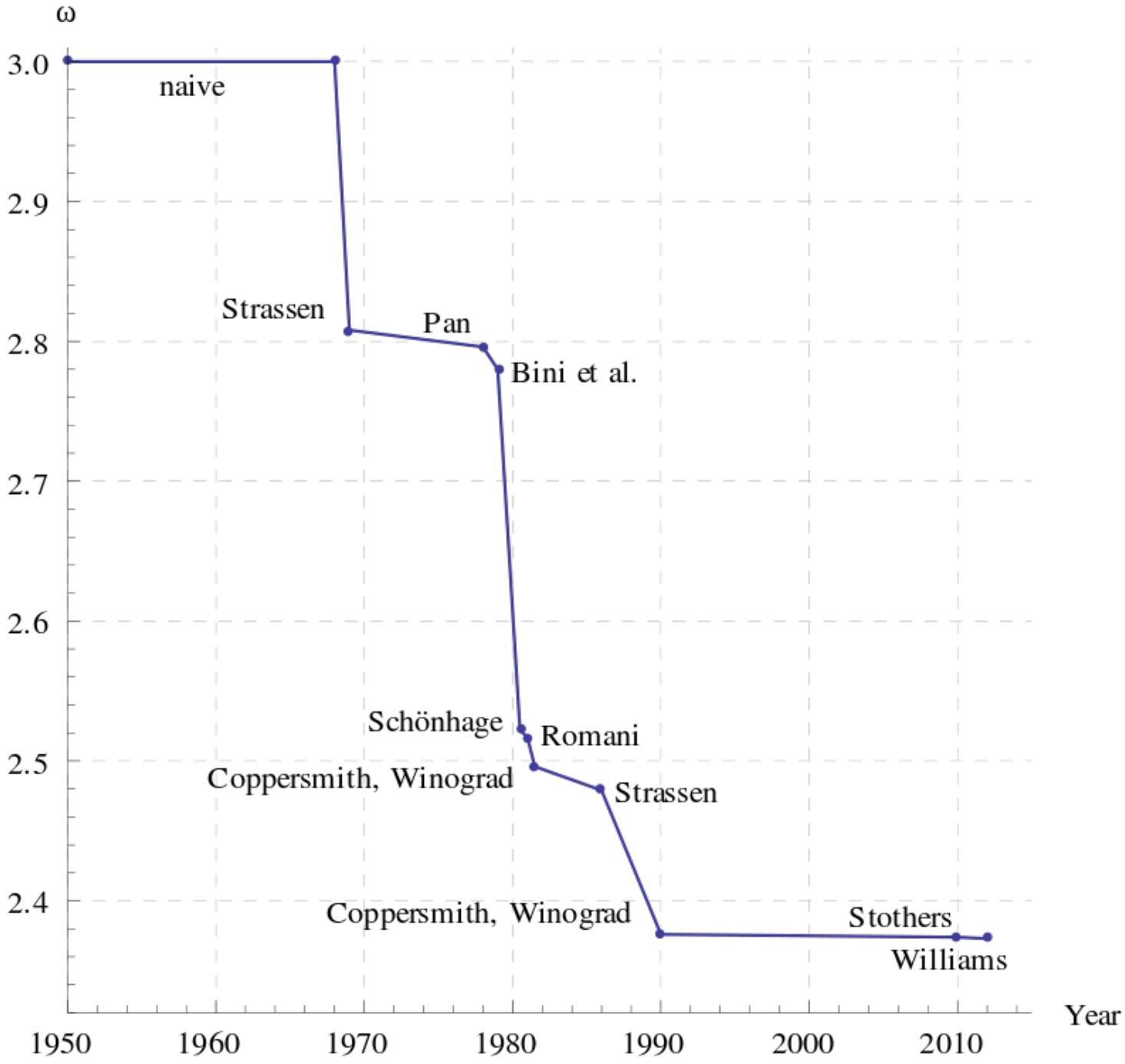
Algorithmes et leurs complexité

Ici nous supposons que l'addition et la multiplictions prennent $O(1)$ secondes. C'est vrai en pratique pour les nombres codés avec un nombre d'octets limité. Sinon, il faut voir les algorithmes de Karatsuba, Toom-Cook, Schönhage-Strassen, Fürer.

$$\mathsf{A}_{n\times k}\mathsf{B}_{k\times m}$$

- **Algorithme classique** $O(nkm)$.
  Dans le cas de matrices carrées $O(n^3)$
- **Strassen algorithm** $O(N^{\log_2 7}) \approx O(N^{2.8074})$
  `http://www.cs.utsa.edu/~wagner/CS3343/strassen/strassen.html`

$O(n^{\omega})$


ω
3.0
2.9
2.8
2.7
2.6
2.5
2.4
naive
Strassen
Pan
Bini et al.
Schönhage
Romani
Coppersmith, Winograd
Strassen
Coppersmith, Winograd
Stothers
Williams
1950
1960
1970
1980
1990
2000
2010
Year

Bah ! ça reste compliqué...

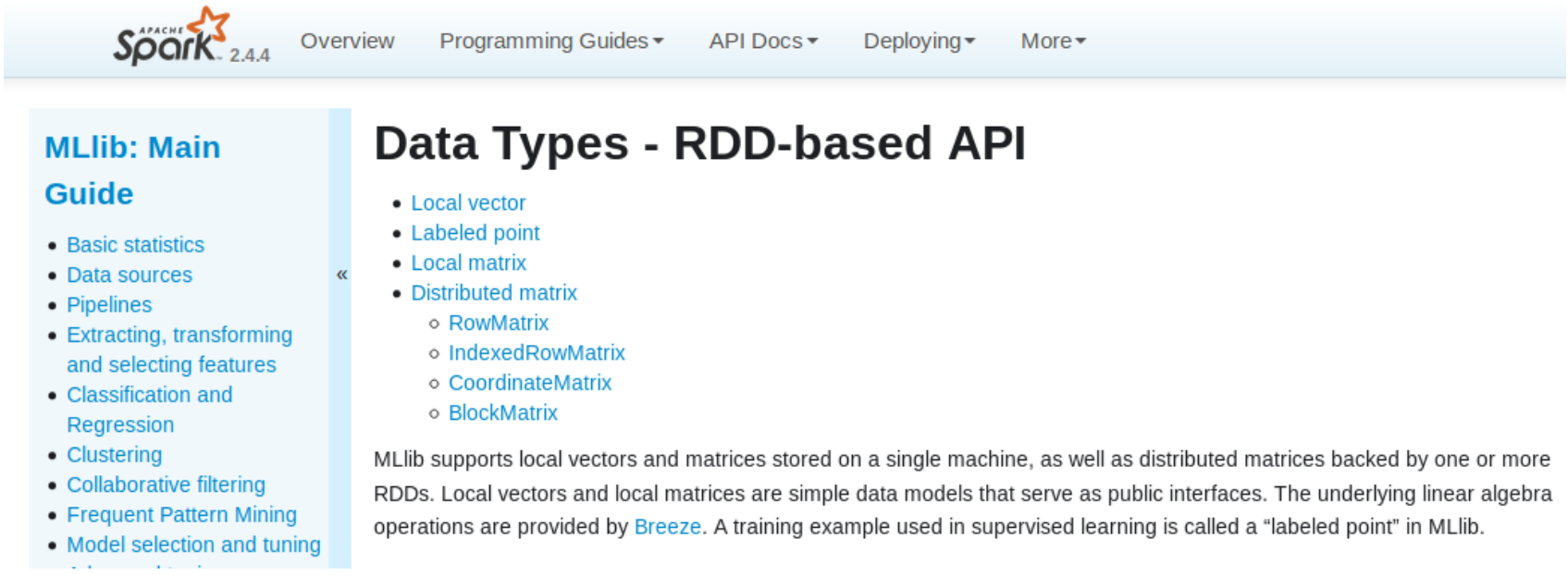
Même la lecture des valeurs prend $O(n^2)$

Ordinateurs quantiques ??

Multipliction rapide pour le big data matrice ?

Deux solutions :

- Algorithmes parallèles et distribués
- Utiliser (si possible) les matrices creuses :
  les matrices contenant beaucoup de zéros.
  Idée : ne pas stocker les zéros. Ne rien multiplier par 0, le résultat est connu d'avance !



https://spark.apache.org/docs/latest/mllib-data-types.html

MLlib: Main Guide

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- **Data types**
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- Dimensionality reduction
- Feature extraction and transformation
- Frequent pattern mining
- Evaluation metrics
- PMML model export

Distributed matrix

A distributed matrix has long-typed row and column indices and double-typed values, stored distributively in one or more RDDs. It is very important to choose the right format to store large and distributed matrices. Converting a distributed matri to a different format may require a global shuffle, which is quite expensive. Four types of distributed matrices have been implemented so far.

The basic type is called `RowMatrix`. A `RowMatrix` is a row-oriented distributed matrix without meaningful row indices, e.g., a collection of feature vectors. It is backed by an RDD of its rows, where each row is a local vector. We assume that the number of columns is not huge for a `RowMatrix` so that a single local vector can be reasonably communicated to the driver and can also be stored / operated on using a single node. An `IndexedRowMatrix` is similar to a `RowMatrix` but with row indices, which can be used for identifying rows and executing joins. A `CoordinateMatrix` is a distributed matrix stored in coordinate list (COO) format, backed by an RDD of its entries. A `BlockMatrix` is a distributed matrix backed by an RDD of `MatrixBlock` which is a tuple of `(Int, Int, Matrix)`.

Note

The underlying RDDs of a distributed matrix must be deterministic, because we cache the matrix size. In general, the use of non-deterministic RDDs can lead to errors.

RowMatrix

A `RowMatrix` is a row-oriented distributed matrix without meaningful row indices, backed by an RDD of its rows, where each row is a local vector. Since each row is represented by a local vector, the number of columns is limited by the integer range but it should be much smaller in practice.

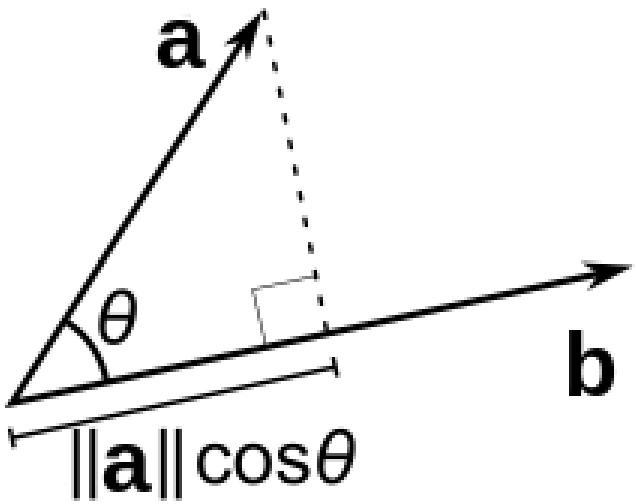
Scala **Java** **Python**

A `RowMatrix` can be created from an `RDD[Vector]` instance. Then we can compute its column summary statistics and decompositions. QR decomposition is of the form A = QR where Q is an orthogonal matrix and R is an upper triangular matrix. For singular value decomposition (SVD) and principal component analysis (PCA), please refer to Dimensionality

Norme de vecteur, produit scalaire

Vecteurs $\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$ et $\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$ de dimension n.

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta = \sum_{i=1}^{n} a_i b_i$$



$$\|\mathbf{a}\|^2 = \langle \mathbf{a}, \mathbf{a} \rangle$$

Valeurs et vecteurs propres

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Valeurs et vecteurs propres

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

- λ valeur propre
- $\mathbf{x}$ vecteur propre

Les vissage propres (Eigenfaces).

An individual face was video recorded and digitized at $2^7 \times 2^7$ pixels with 2^8 gray level by means of an IVS-100 image processor. Faces were lined up by a cross-hair overlay display that appeared on a video monitor. The vertical line passed through the symmetry line of the face and the horizontal line through the pupils of the eyes. Field depth was adjusted so that facial width was the same for images. Since these steps were all adjusted by eye, this contributed to the general error level. The pictures were taken under background-lighting conditions. Since the lighting varied with the time of day, this too was a source of error. To some extent, this error was diminished by a normalization procedure that we now describe.

A face, or for that matter any object, can be regarded as a pointwise map of reflectivities, say, $r(\mathbf{x})$. Under a uniform illumination, say, I, the face is given by

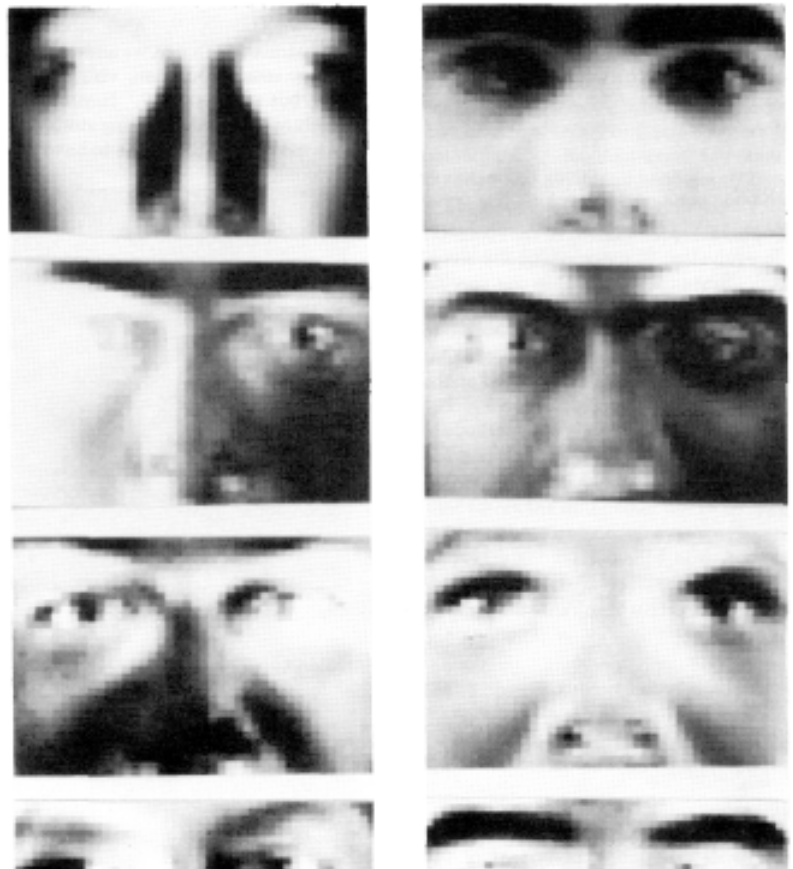
$$\tilde{\varphi}(\mathbf{x}) = Ir(\mathbf{x}). \tag{21}$$

For a variety of reasons, it is important to normalize a picture so that a reference portion of a face is at a standard level of illumination. If we denote a reference point by $\mathbf{x}_0$ and standard light level by I_0, then we take the normalized picture to be

$$\varphi(\mathbf{x}) = \frac{I_0}{\tilde{\varphi}(\mathbf{x}_0)}\tilde{\varphi}(\mathbf{x}). \tag{22}$$

In actual practice we took the reference portion to be small, high cheek areas below each eye and averaged the light level over these two patches. This procedure provides a specific light-level normalization. In addition, it provides the basis for the future identification of a picture.

The average face was computed according to Eq. (3) and is

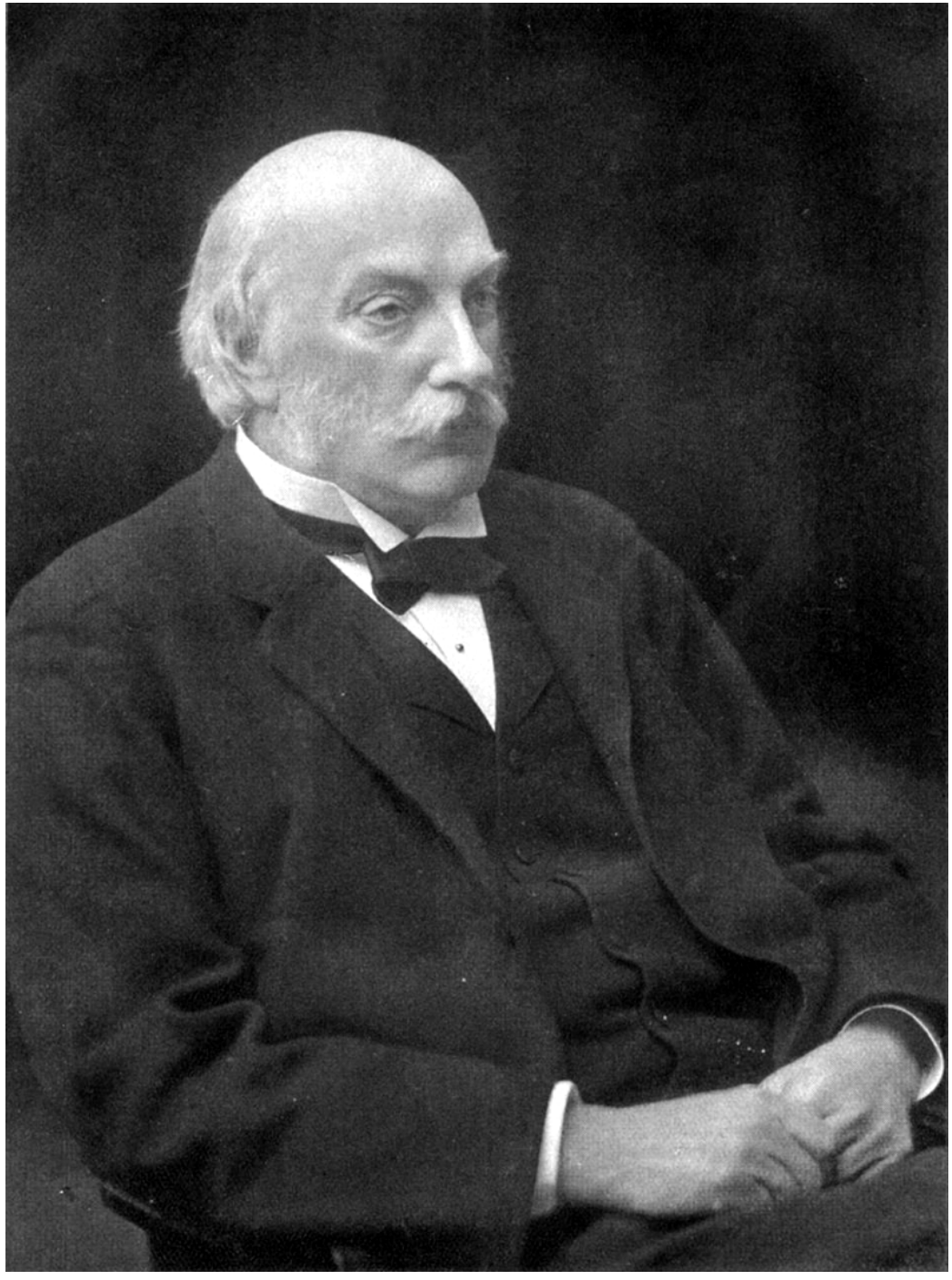


Sirovich and Kirby (1987)
Turk and Pentland (1991)

Caractérisation variationnelle des valeurs propres !

Minima d'une certain fonction !

The Lord Rayleigh /ˈreɪli/



Quotient de Rayleigh

Une matrice symétrique $\mathbf{A}_{n,n}$, un vecteur $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

$$R(\mathbf{A}, \mathbf{x}) = \frac{\mathbf{x}^T \mathbf{A} \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$$

$$R(\mathbf{A}, \mathbf{x}) = \frac{\sum_{i=1}^{n} a_{i,j} x_i x_j}{\sum_{i=1}^{n} x_i^2}$$

On s'interesse qu'aux vecteurs de la norme 1, c'est-à-dire $\sum_{i=1}^{n} x_i^2 = 1$, alors $R(\mathbf{A}, \mathbf{x}) = \sum_{i=1}^{n} a_{i,j} x_i x_j$

Théorème min-max de Courant-Fischer

Si la matrice $\mathbf{M}$ est symétrique, ces valeurs propres sont réelles :

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

Théorème min-max de Courant-Fischer

Si la matrice $\mathbf{M}$ est symétrique, ces valeurs propres sont réelles :

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

$$\lambda_1 = \max_{\|\mathbf{x}\|=1} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

$$p_1 = \arg\max_{\|\mathbf{x}\|=1} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

$$\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1, \|\mathbf{v}_1\| = 1$$

Théorème min-max de Courant-Fischer

Si la matrice $\mathbf{M}$ est symétrique, ces valeurs propres sont réelles :

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

$$\lambda_1 = \max_{\|\mathbf{x}\|=1} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

$$p_1 = \arg\max_{\|\mathbf{x}\|=1} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

$$\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1, \|\mathbf{v}_1\| = 1$$

$$\lambda_2 = \max_{\|\mathbf{x}\|=1, \mathbf{x} \perp \mathbf{p_1}} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

$$\mathbf{p}_2 = \arg\max_{\|\mathbf{x}\|=1, \mathbf{x} \perp \mathbf{p_1}} \left(\mathbf{x}^T \mathbf{A} \mathbf{x} \right)$$

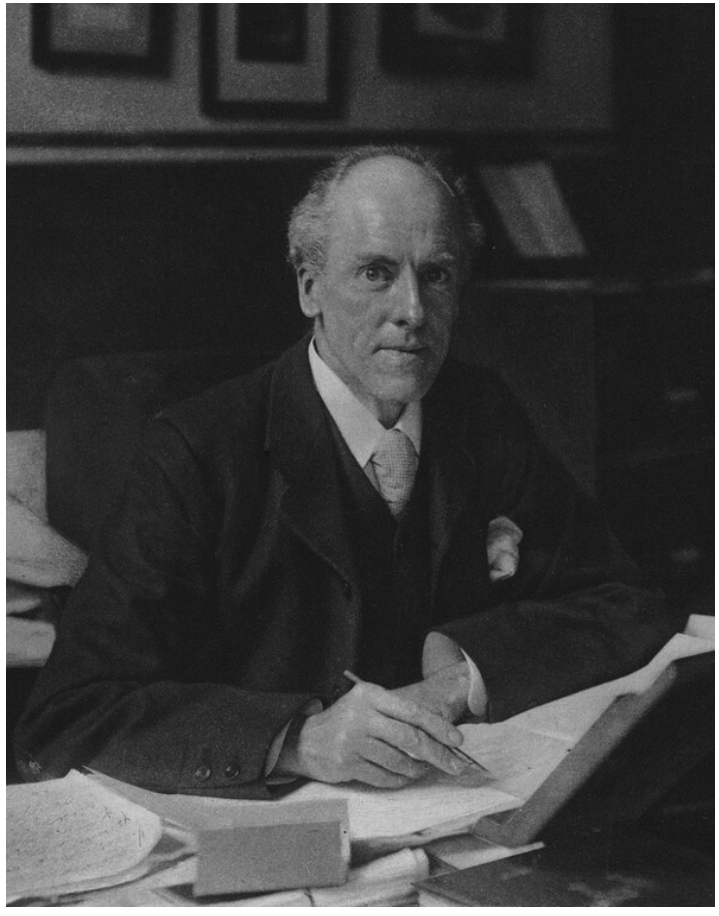
$$\mathbf{A}\mathbf{p}_2 = \lambda_2 \mathbf{p}_2, \|\mathbf{p}_2\| = 1$$

etc...

Analyse en composantes principales

PCA was invented in 1901 by Karl Pearson, as an analogue of the **principal axis theorem** *in mechanics; it was later independently developed and named by Harold Hotelling in the 1930s.*

— Wikipedia



Matrice de données

Matrice de données

$$\mathbf{D}_{n\times k} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,k} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,k} \end{pmatrix}$$

n nombre des instances de données.
k nombre de caractéristiques (features).

PCA. Minimisation des erreurs

Matrice de données

$$\mathsf{D}_{n\times k} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,k} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,k} \end{pmatrix} = \begin{pmatrix} \boxed{\boldsymbol{\delta}_1} \\ \boxed{\boldsymbol{\delta}_2} \\ \vdots \\ \boxed{\boldsymbol{\delta}_n} \end{pmatrix}$$

n nombre des instances de données
k nombre de caractéristiques (features)
$\boldsymbol{\delta}_i$ ième instance.

On cherche un vecteur x, $\|\mathsf{x}\| = 1$, t.q.

$$\min \sum_{i=1}^{n} \|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathsf{x}\rangle \mathsf{x}\|^2$$

On minimise la somme des différences entre les données et leurs projections sur x !

$$\langle \boldsymbol{\delta}_i, \mathsf{x}\rangle = \|\boldsymbol{\delta}_i\| \, \|\mathsf{x}\| \cos\theta = \|\boldsymbol{\delta}_i\| \cos\theta$$

$\langle \boldsymbol{\delta}_i, \mathsf{x}\rangle \in \mathbb{R}$**, mais** $\langle \boldsymbol{\delta}_i, \mathsf{x}\rangle \mathsf{x}$ **est un vecteur !**

Minimisation des erreurs = Maximisation de variance !

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\min \sum_{i=1}^{n} \|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle \mathbf{x}\|^2$$

Minimisation des erreurs = Maximisation de variance !

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\min \sum_{i=1}^{n} \|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle \mathbf{x}\|^2$$

Théorème de Pythagore

$$\|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle \mathbf{x}\|^2 = \|\boldsymbol{\delta}_i\|^2 - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle^2$$

Minimisation des erreurs = Maximisation de variance !

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\min \sum_{i=1}^{n} \|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle \mathbf{x}\|^2$$

Théorème de Pythagore

$$\|\boldsymbol{\delta}_i - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle \mathbf{x}\|^2 = \|\boldsymbol{\delta}_i\|^2 - \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle^2$$

Alors,

$$\max \sum_{i=1}^{n} \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle^2$$

Maximisation de variance

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\max \sum_{i=1}^{n} \langle \boldsymbol{\delta}_i, \mathbf{x} \rangle^2 = \max \|\mathbf{D}\mathbf{x}\|^2 =$$

$$\|\mathbf{a}\|^2 = \langle \mathbf{a}, \mathbf{a} \rangle$$

$$= \max \left\langle \mathbf{D}\mathbf{x}, \mathbf{D}\mathbf{x} \right\rangle$$

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b}$$

$$= \max \left((\mathbf{D}\mathbf{x})^T \mathbf{D}\mathbf{x} \right)$$

$$(\mathbf{A}\mathbf{B})^T = \mathbf{B}^T \mathbf{A}^T$$

$$= \max \left(\mathbf{x}^T \mathbf{D}^T \mathbf{D}\mathbf{x} \right)$$

$$\text{Associativité } (\mathbf{A}\mathbf{B})\mathbf{C} = \mathbf{A}(\mathbf{B}\mathbf{C})$$

$$= \max \left(\mathbf{x}^T (\mathbf{D}^T \mathbf{D}) \mathbf{x} \right)$$

Covariance

$\mathbb{E}[Z]$ désigne l'espérance mathématique de la variable aléatoire Z. Pour deux variables aléatoires réelles X et Y on définit **la covariance**

$$\mathrm{Cov}(X, Y) = \mathbb{E}\Big[\big(X - \mathbb{E}[X]\big)\,\big(Y - \mathbb{E}[Y]\big)\Big]$$

La variance de X est donc $\mathrm{Var}(X) = \mathrm{Cov}(X, X)$.

L'écart type de X est $\sigma(X) = \sqrt{\mathbb{E}\left[\big(X - \mathbb{E}[X]\big)^2\right]}$.

Matrice de n-covariance

Considérons chaque colonne comme un échantillon d'une variable aléatoire.

Matrice de n-covariance

Considérons chaque colonne comme un échantillon d'une variable aléatoire.

$$\mathsf{D}_{n\times k} = \begin{pmatrix} \mathsf{V}_1 & \mathsf{V}_2 & \cdots & \mathsf{V}_k \end{pmatrix}$$

Matrice de n-covariance

Considérons chaque colonne comme un échantillon d'une variable aléatoire.

$$\mathbf{D}_{n\times k} = \begin{pmatrix} \mathbf{V}_1 & \mathbf{V}_2 & \cdots & \mathbf{V}_k \end{pmatrix}$$

La matrice de n-covariance est la matrice $\mathbf{C}$, t.q $c_{i,j}$ est égal à la covariance entre $\mathbf{V}_i$ et $\mathbf{V}_j$ fois n.

$$\mathbf{B} = \begin{pmatrix} \mathbf{V}_1 - \mathbb{E}[\mathbf{V}_1] & \mathbf{V}_2 - \mathbb{E}[\mathbf{V}_2] & \cdots & \mathbf{V}_k - \mathbb{E}[\mathbf{V}_k] \end{pmatrix}$$

Matrice

$$\mathbf{C} = \mathbf{B}^T\mathbf{B}$$

est symétrique semi-définie positive, ces valeurs propres sont réelles et positives :

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq 0$$

Application du Théorème min-max de Courant-Fischer

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\max \left(\mathbf{x}^T (\mathbf{D}^T \mathbf{D}) \mathbf{x} \right)$$

Si on change la matrice $\mathbf{D}$ en soustrayant la moyenne de chaque colonne... on pourra interpreter $\mathbf{D}^T \mathbf{D}$ comme la matrice de n-covariance $\mathbf{C}$.

Application du Théorème min-max de Courant-Fischer

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\max \left(\mathbf{x}^T (\mathbf{D}^T \mathbf{D}) \mathbf{x} \right)$$

Si on change la matrice $\mathbf{D}$ en soustrayant la moyenne de chaque colonne... on pourra interpreter $\mathbf{D}^T \mathbf{D}$ comme la matrice de n-covariance $\mathbf{C}$.

$$\max \left(\mathbf{x}^T \mathbf{C} \mathbf{x} \right)$$

Application du Théorème min-max de Courant-Fischer

On cherche un vecteur $\mathbf{x}$, $\|\mathbf{x}\| = 1$, t.q.

$$\max \left(\mathbf{x}^T (\mathbf{D}^T \mathbf{D}) \mathbf{x} \right)$$

Si on change la matrice $\mathbf{D}$ en soustrayant la moyenne de chaque colonne... on pourra interpreter $\mathbf{D}^T \mathbf{D}$ comme la matrice de n-covariance $\mathbf{C}$.

$$\max \left(\mathbf{x}^T \mathbf{C} \mathbf{x} \right)$$

et par le théorème min-max de Courant-Fischer on sait que c'est la plus grande valeur propre qui maximise ce quotient de Rayleigh.

Schéma d'algorithme PCA

- Soustraire des moyennes des colonnes
- Calculer la matrice de n-covariance C
- Trouver les plus grandes valeurs et vecteurs propres de la matrice C
- Projetez les données sur le sous-espace engendré par ces vecteurs. Plus il y a des vecteurs, mieux on explique les données.

Algorithmes de calcul

On n'a pas besoin de calculer toutes les vecteurs propres.

Nous ne sommes intéressés que par ceux qui sont liés aux plus grandes valeurs propres.

Et encore... on peut les approcher sans calculer avec une précision absolue par des méthodes itératives :

- Méthode de la puissance itérée
- Itération du quotient de Rayleigh

MLlib: Main Guide

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- Data types
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- **Dimensionality reduction**
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and

Principal component analysis (PCA)

Principal component analysis (PCA) is a statistical method to find a rotation such that the first coordinate has the largest variance possible, and each succeeding coordinate, in turn, has the largest variance possible. The columns of the rotation matrix are called principal components. PCA is used widely in dimensionality reduction.

`spark.mllib` supports PCA for tall-and-skinny matrices stored in row-oriented format and any Vectors.

Scala **Java** **Python**

The following code demonstrates how to compute principal components on a `RowMatrix` and use them to project the vectors into a low-dimensional space.

Refer to the `RowMatrix` Python docs for details on the API.

```
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.linalg.distributed import RowMatrix

rows = sc.parallelize([
    Vectors.sparse(5, {1: 1.0, 3: 7.0}),
    Vectors.dense(2.0, 0.0, 3.0, 4.0, 5.0),
    Vectors.dense(4.0, 0.0, 0.0, 6.0, 7.0)
])

mat = RowMatrix(rows)
# Compute the top 4 principal components.
# Principal components are stored in a local dense matrix.
pc = mat.computePrincipalComponents(4)

# Project the rows to the linear space spanned by the top 4 principal components.
projected = mat.multiply(pc)
```

Autres méthodes de réduction de dimensionnalité :

- Non-negative matrix factorization (NMF)
- Kernel PCA
- Graph-based kernel PCA
- Linear discriminant analysis (LDA)
- Generalized discriminant analysis (GDA)
- Autoencoder

Questions ?