Data Science Training

November 2017

Feature Extraction

Xavier Bresson

Data Science and AI Research Centre
NTU, Singapore

http://data-science-optum17.tk

Outline

- **The Feature Extraction Problem**
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- Robust PCA
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- Conclusion

The Feature Extraction Problem

Q: What are features?

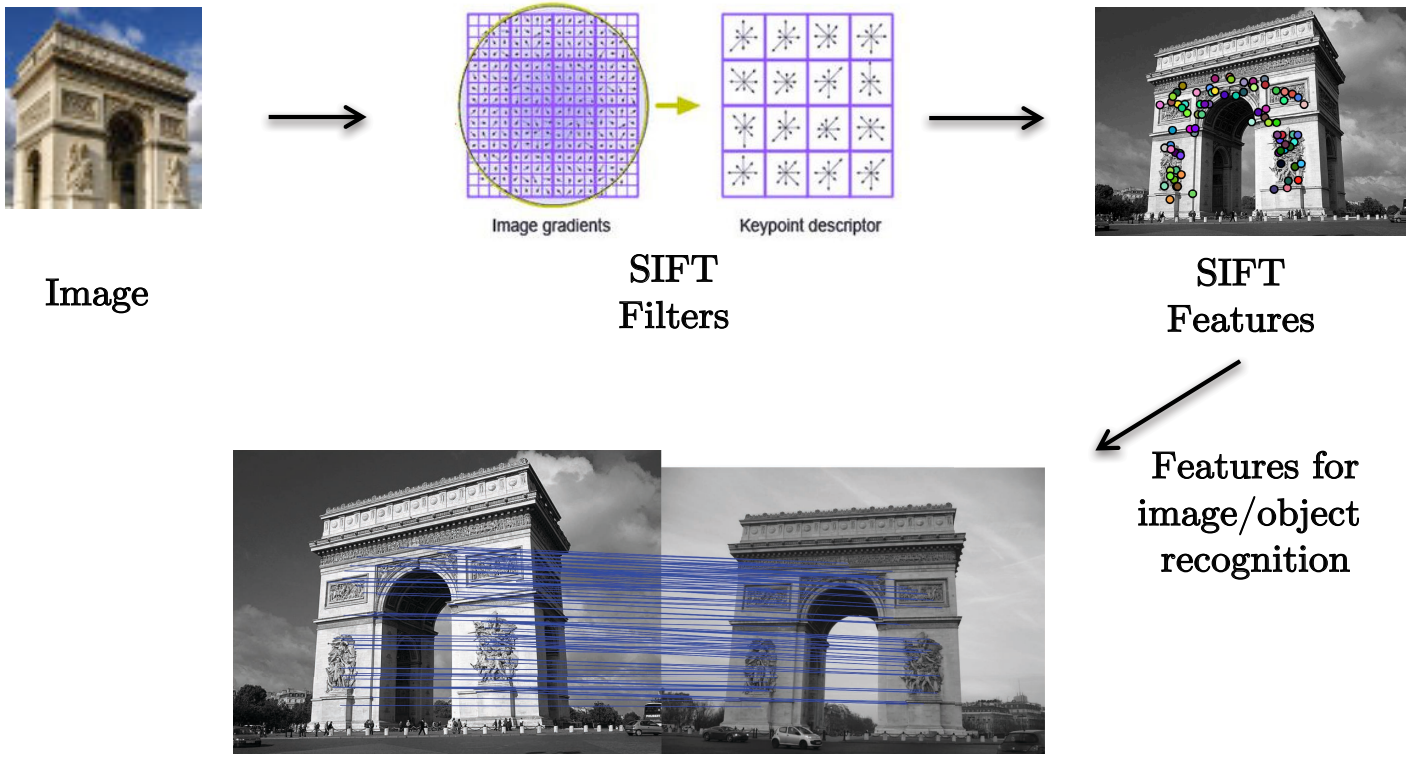
- **Goal:** Find the **best possible representation of data** that reveal special **structures** useful for further applications (like classification, recognition, etc).

- **There are two types of filters for feature extraction:**
  - *(1)* ***Handcrafted*** *features* – from domain expertise.
  - *(2)* ***Learned*** *features from data* – linear and non-linear representations. This approach has become dominant.

Handcrafted Features

- **Domain expertise:** Handcrafted features are domain-dependent, i.e. designed from **experts in specific fields** with years of experience (usually not generalizable to other fields).

- **Popular example: SIFT** - Best image features in Computer Vision, used for many applications such as image recognition. It needed **30 years** of experience (1966-1999) to design good image features!



Learned Features From Data

- **Paradigm shift:** Handcrafted filters/features have been successful for decades but the emergence of big data has made available enough data to *learn the best features from data directly*, without handcrafting anything.

  Besides, deep learning has showed us how to *extract highly meaningful features from data.*

- **Learned filters:** There exist **two classes** of learned filters depending on the **mathematical data representations**:

  (1) ***Linear*** *representation*

  (2) ***Non-linear*** *representation*

Linear Representation

- **Formulation:**

$$z = Dx$$

Features or coefficients of x in the dictionary D

Dictionary of filters (or basis functions)

High-dim data

$$z = Dx = \begin{bmatrix} \langle D_{1,.}, x \rangle \\ \vdots \\ \langle D_{K,.}, x \rangle \end{bmatrix} \quad \rightarrow \quad z_i = \langle D_{i,.}, x \rangle$$

i^{th} coefficient

i^{th} filter

Linear Representation

- **How to learn D and z?**

  $\Rightarrow$ Techniques available: *PCA, ICA, NMF, Sparse Coding, etc.*

- **Which technique to choose?**

  Each technique assumes different **assumptions on data**. Pick the one that follows your data properties (later discussed).

Non-Linear Representation

- **Non-linear mapping φ:**

$$\textit{Linear representation:} \quad x \quad \rightarrow \quad z = Dx$$

$$\textit{Non-linear representation:} \quad x \quad \rightarrow \quad z = \varphi(x) \quad (z \neq Dx)$$

- These techniques are called **non-linear dimensionality reduction** techniques, and they are used for feature extraction, classification, visualization, etc.

- **Examples:**

  (1) Non-linear PCA, Locally Linear Embedding (LLE), Laplacian Eigenmaps, t-Distributed Stochastic Neighbor Embedding (t-SNE)

  (2) Deep Learning

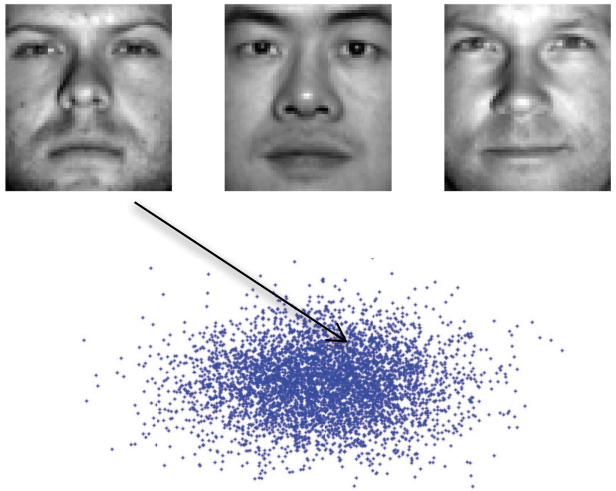
Linear vs. Non-Linear Representations

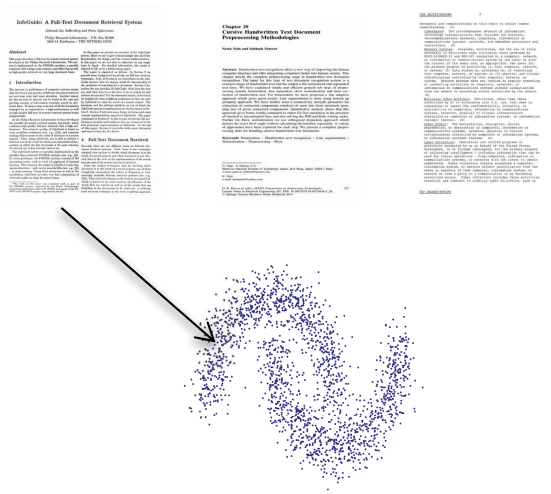
- **Which representation to use?** The answer depends on the type of data distributions:

  *If data follow a **Gaussian Mixture Model** (GMM) like human faces, it is then enough to use **linear** data representation (and useless to use non-linear techniques).*

  *However, if data follow **complex distributions** like text documents then they require **non-linear** techniques.*

Q: What is the shape of Gaussian Model?





Outline

- The Feature Extraction Problem
- **Standard Principal Component Analysis (PCA)**
- Sparse PCA
- Robust PCA
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- Conclusion

Principal Component Analysis

Q: Who does *not* know PCA?

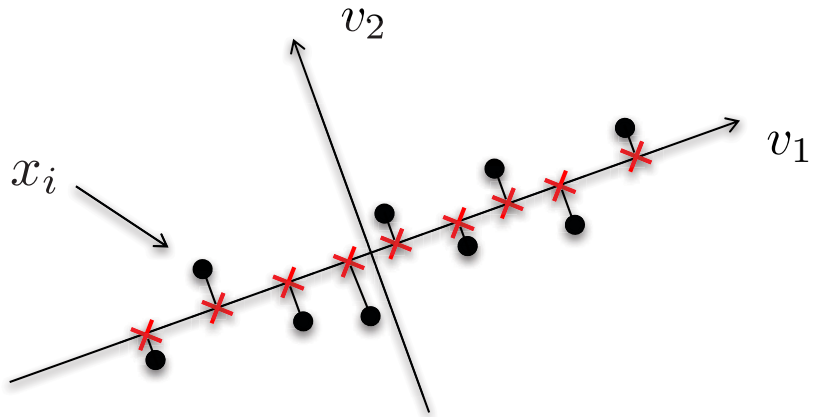
- **History:** PCA introduced by Pearson in 1901 in physics.

  PCA is the most popular technique for linear representation (also one of the top 10 techniques in data science).

- **Formulation:** Given a set of data, PCA aims at projecting data onto an orthogonal **basis that best captures the data variance**.

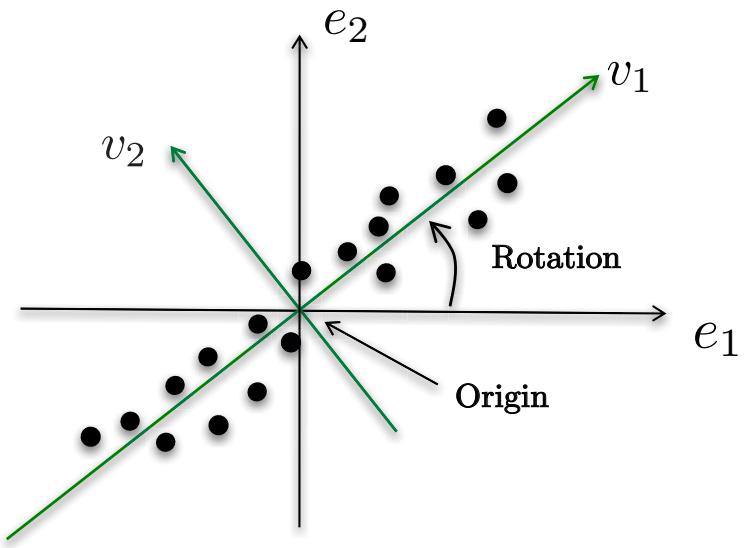
- **Consequence:** The first basis function or **principal direction $\mathbf{v}_1$** captures the **largest possible variance of data**.

  The second basis function or principal direction v_2 captures the largest possible variance of data while satisfying **orthogonality constraint $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = 0$**. etc



Formalization

- PCA defines an **orthogonal transformation** that maps the data to a new coordinate system $(v_1, v_2, \ldots, v_K)$ called **principal directions** such that the v_k's **capture the largest possible data variances**.



- **Notes:**

  *(1) PCA requires the data to be **centered**.*

  *(2) PCA does not say anything about data **normalization**, but its analysis may change (PCA is not invariant w.r.t. normalization).*

Covariance Matrix

- **Definition:** The **Covariance** Matrix C is defined as

$$C = X^T X$$

d x d

d x n

X= n x d data matrix
n = number of data
d = number of dimensions

- Covariance matrix C encodes all data variances **along each dimension**:

$$C_{\alpha\alpha} = \langle X_{\cdot,\alpha}, X_{\cdot,\alpha} \rangle = \|X_{\cdot,\alpha}\|_2^2 = \sum_{i=1}^{n} |X_{i\alpha}|^2 = \sum_{i=1}^{n} x_{i,\alpha}^2$$

Variance of data along α^{th} dimension

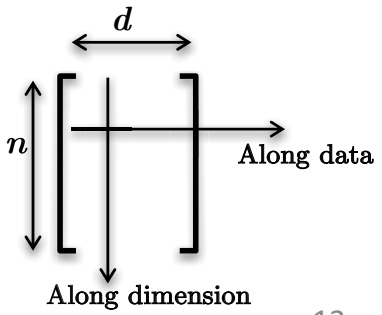
$$C_{\alpha\beta} = \langle X_{\cdot,\alpha}, X_{\cdot,\beta} \rangle = \sum_{i=1}^{n} X_{i\alpha} X_{i\beta} = \sum_{i=1}^{n} x_{i,\alpha} x_{j,\alpha}$$

Covariance of data along α^{th} and β^{th} dimensions

Note: x_i are zero-centered along each dimension α:

$$\mathbb{E}(\{x_i\}) = \sum_{i=1}^{n} x_{i,\alpha} = \sum_{i=1}^{n} X_{i\alpha} = 0 \ \forall \alpha$$

X (n x d) =



PCA = EVD of Covariance Matrix

- **Proof:** Let us show that $Cv_1 = \lambda_1 v_1$

Principal direction (PD) $\mathbf{v_1}$ ← $Cv_1 = \lambda_1 v_1$ → **Largest variance of data along PD $\mathbf{v_1}$**

Vector $\mathbf{v_1}$ $= 1^{\text{st}}$ principal direction

$$= \text{Direction of largest data variance}$$

$$= \underset{\|v\|_2=1}{\text{argmax}} \sum_{i=1}^{n} |\langle x_i, v \rangle|^2$$

Square distance of data projected on v

Matrix notation ⇕

$$= \underset{\|v\|_2=1}{\text{argmax}} \quad \|Xv\|_2^2 = (Xv)^T(Xv) = v^T X^T X v = v^T C v$$

⇓

Spectral solution: Solution v_1 is the largest eigenvector of C

$$Cv_1 = \lambda_1 v_1 \;\rightarrow\; v_1^T C v_1 = v_1^T \lambda_1 v_1 = \lambda_1 \|v_1\|_2^2 = \lambda_1 = \underset{\|v\|_2=1}{\text{argmax}} \sum_{i=1}^{n} |\langle x_i, v \rangle|^2 \qquad \square$$

PCA = EVD of Covariance Matrix

- **Vector $\mathbf{v_2}$** = 2nd principal direction
  = Direction of *second* largest data variance
  $$= \underset{\|v\|_2=1}{\operatorname{argmax}} \; v^T C v \;\; \text{s.t.} \;\; \langle v, v_1 \rangle = 0$$
  $$\Downarrow$$
  $$C v_2 = \lambda_2 v_2, \;\; \text{with} \;\; \lambda_1 \geq \lambda_2$$

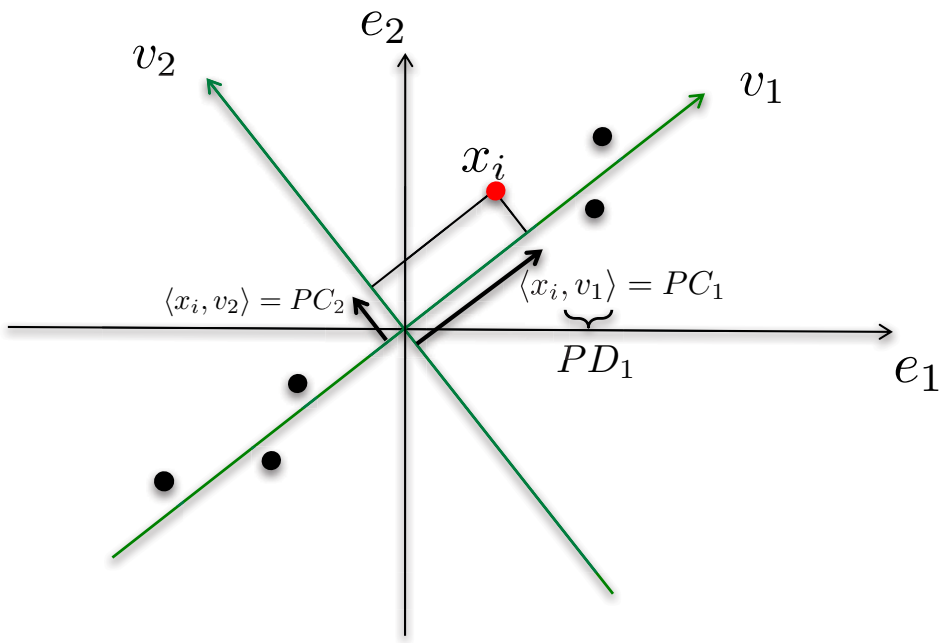
- **Vector $\mathbf{v_3}$** = 3rd principal direction ...

- **Matrix factorization:** Full EVD of C
  $$C = V \Lambda V^T$$
  $$\text{with} \;\; V = \Big[v_1, ..., v_d \Big], \; V^T V = I_d, \; \Lambda = \text{diag}(\lambda_1, ..., \lambda_d)$$

Principal Components

- **Definition:** PCs are the coordinates of original data projected into the basis of principal directions (PDs):

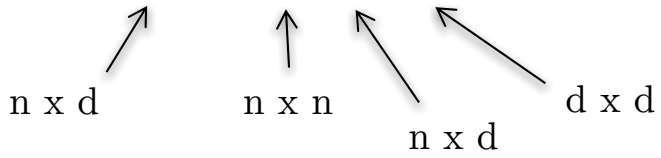
$$X_{pca} = XV$$



PCA = SVD of Data Matrix

- **Matrix factorization:**

$$X = U\Sigma V^T \quad \text{with} \quad UU^T = I_n, \; VV^T = I_d$$



- **Relationship between EVD and SVD:**

$$C = X^T X = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma(U^T U)\Sigma V^T = V_{svd}\Sigma^2 V_{svd}^T$$
$$= V_{evd}\Lambda V_{evd}^T$$

$$\rightarrow \quad \begin{cases} V_{svd} = V_{evd} \\ \Sigma^2 = \Lambda \quad \rightarrow \; \lambda_k = \sigma_k^2 \\ X_{pca} = XV_{evd} = U_{svd}\Sigma \end{cases}$$

- **Q: PCA with EVD or SVD?** It depends on the size of data matrix X:

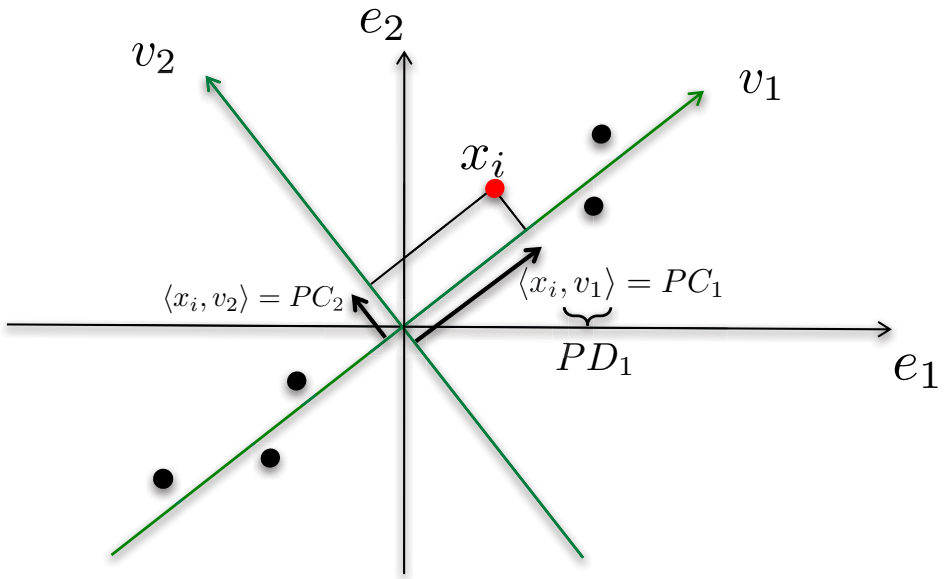
  (1) For d>n: use SVD.

  (2) For d<n: use EVD.

PCA as Dimensionality Reduction

- **Essential observation:** Most (linear) data **are concentrated along the first principal directions:**

$$x_i = \langle x_i, v_1 \rangle v_1 + \langle x_i, v_2 \rangle v_2$$
$$\approx \langle x_i, v_1 \rangle v_1$$



The first PDs are enough to provide a **good data representation**, i.e.

$$\|X - X_K\|_2^2 \text{ small for a small } K$$

Approximation of X with first K PDs:

$$X_K = U \Sigma_K V_K^T$$

Truncated Σ with first K data variances

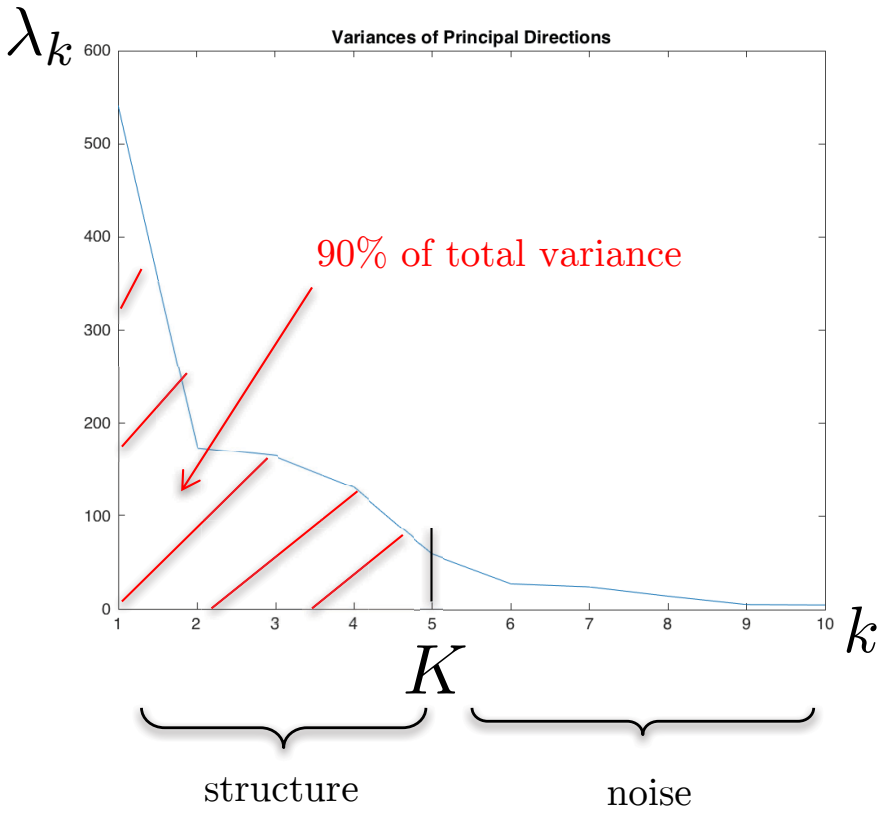
Truncated V with first K PDs

How to select K?

- **Rule:** Data variance is captured by each principal direction, then if one wants to **retain 90% of total variance** then K must be selected as follows:

$$\frac{\sum_{k=1}^{K} \lambda_k}{\sum_{k=1}^{d} \lambda_k} \geq 0.9$$

YaleBFaces dataset



PCA as Visualization Tool

- **Use first principal components for visualization:** If high-dimensional data are linear, follow a GMM distribution then they can be visualized in 2D, 3D spaces.

Demo: Standard/Linear PCA

- Run **code01.ipynb**

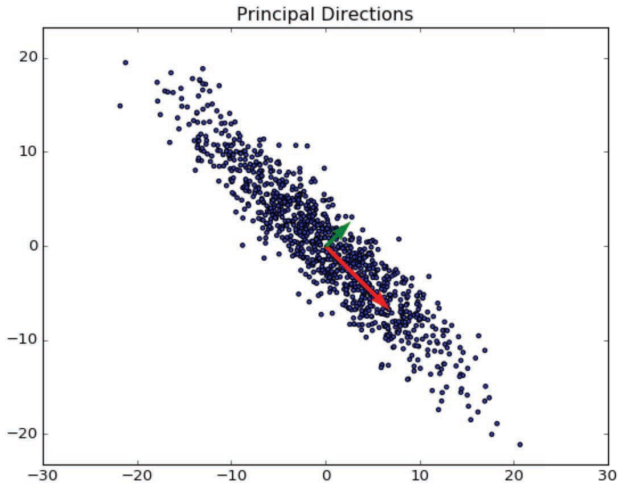
```
# Zero-centered data
Xzc = X - np.mean(X,axis=0)

# Covariance matrix
CovX = (Xzc.T).dot(Xzc)

# Compute largest 5 eigenvectors/eigenvalues
nb_pca = 5
CovX = scipy.sparse.csr_matrix(CovX)
lamb, U = scipy.sparse.linalg.eigsh(CovX, k=nb_pca, which='LM') # U = d x nb_pca
EVec = U[:,::-1] # largest = index 0
EVal = lamb[::-1]

# Principal Components
Xpc = X.dot(EVec)

# Principal Directions
v1 = EVec[:2,0]
v2 = EVec[:2,1]
```



Outline

- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- **Sparse PCA**
- Robust PCA
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- Conclusion

Sparse PCA

Q: Is PCA interpretable?

- **Motivation:** Standard PCA is able to

  (1) Capture most variability information contained in data.

  (2) Identify uncorrelated information (because principal directions are orthogonal).

- However, PCA is **limited in feature interpretation**: It is hard to to identify the **most relevant features** for each principal direction.

- Example: Analysis of genes with standard PCA gives:

Q: What genes are the most meaningful?

`Principal_Directions =`

Gene1	-0.4038	0.2179	-0.2073	-0.0912	0.0826	-0.1198
Gene2	-0.4055	0.1861	-0.2350	-0.1027	0.1128	-0.1629
Gene3	-0.1244	0.5406	0.1415	0.0784	-0.3498	0.2759
...	-0.1732	0.4556	0.3524	0.0548	-0.3558	0.0540
	-0.0572	-0.1701	0.4812	0.0491	-0.1761	-0.6256
	-0.2844	-0.0142	0.4753	-0.0634	0.3158	-0.0523
	-0.3998	-0.1896	0.2531	-0.0650	0.2151	-0.0027
	-0.2936	-0.1892	-0.2431	0.2855	-0.1853	0.0551
	-0.3566	0.0171	-0.2076	0.0967	0.1061	-0.0342
	-0.3789	-0.2485	-0.1188	-0.2050	-0.1564	0.1731
	0.0111	0.2053	-0.0705	0.8037	0.3430	-0.1753
	0.1151	0.3432	0.0920	-0.3008	0.6004	0.1698
	0.1125	0.3085	-0.3261	-0.3034	-0.0799	-0.6263

→

`Sparse_Principal_Directions =`

-0.4767	0	0	0	0	0
-0.4951	0	0	0	0	0
0	0.8053	0	0	0	0
0	0.5929	0	0	0	0
0.1152	0	0.6172	0	0	0
0	0	0.5809	0	0	0
-0.1938	0	0.5145	0	0	0
-0.3447	0	0	0	0	0
-0.4143	0	0	0	0	0
-0.4317	0	0	0	0	0
0	0	0	1.0000	0	0
0	0	0	0	1.0000	0
0	0	-0.1300	0	0	-1.0000

⇒ Solution: *Sparse PCA*

Sparse PCA Techniques

- Mainly, **three classes** exist:
  (1) Lasso PCA
  (2) Elastic PCA
  (3) Power PCA (based on power method)

- **Lasso PCA:**

  Advantage: Great feature selection technique as it finds **sparse, accurate and robust solutions**.

  Limitation: The **number of features** that can be selected by Lasso is **at most** $\boldsymbol{n}$, the number of data. It may be an issue in some applications like in bioinformatics:

  n = # microarray data = 100

  d = # gene predictors = 10,000

  $\Rightarrow$ Lasso PCA can select at most 100 genes. *Solution:* Elastic PCA.

Elastic PCA

- Elastic PCA solves an **elastic net regression** problem:

$$\min_{A,B} \|X - XBA^T\|_F^2 + \lambda_2\|B\|_F^2 + \lambda_1\|B\|_1 \quad \text{s.t.} \quad A^TA = I_K$$

Data fidelity term

Elastic net regression

L1 term forces sparse solution

Sparse principal directions:

$$sPD_j = V_{\cdot,j} = \frac{B^\star_{\cdot,j}}{\|B^\star_{\cdot,j}\|_2}$$

Sparse principal components:

$$X_{spca} = XV$$

Algorithm

- Optimization problem is **non-smooth but convex** (separately):

Initialization: $A^{m=0} = V_K^{pca}$ (standard PCA)

Iterate until convergence:

Step 1: $B^{m+1} = \underset{B}{\operatorname{argmin}} \ \|X - XBA^m\|_F^2 + \lambda_2\|B\|_F^2 + \lambda_1\|B\|_1 \quad (1)$

Step 2: $A^{m+1} = \underset{A}{\operatorname{argmin}} \ \|X - XB^{m+1}A\|_F^2 \ \text{ s.t. } \ A^TA = I_K \quad (2)$

$\Rightarrow$ Problem (1) can be solved efficiently by FISTA.

$\Rightarrow$ Problem (2) can be solved by SVD.

Demo: Sparse PCA

- Run **code02.ipynb**

```
# Data matrix
mat = scipy.io.loadmat('datasets/pitprops.mat')
covX = mat['covX']
n = covX.shape[0]
data_features = mat['data_features']
d = data_features.shape[0]
print(n,d)
for i,x in enumerate(data_features):
    print('Feature',i,' ',x[0].squeeze())
```

```
13 13
Feature 0   topdiam
Feature 1   length
Feature 2   moist
Feature 3   testsg
Feature 4   ovensg
Feature 5   ringtop
Feature 6   ringbut
Feature 7   bowmax
Feature 8   bowdist
Feature 9   whorls
Feature 10   clear
Feature 11   knots
Feature 12   diaknot
```

```
# Compute first K eigenvalues
lamb, U = scipy.sparse.linalg.eigsh(covX, k=6, which='LM')
EVec = U[:,::-1] # largest = index 0
EVal = lamb[::-1]
Principal_Directions = EVec
Variances_PDs = EVal
print('Principal_Directions=\n',Principal_Directions)
print('\nVariances_PDs=\n',Variances_PDs)
```

```
Principal_Directions=
 [[ 0.40379375 -0.21785161 -0.20729009 -0.09121408  0.08263463 -0.11980254]
 [ 0.40554469 -0.18612694 -0.23503529 -0.10271814  0.11278958 -0.16288841]
 [ 0.12440384 -0.54064233  0.14148843  0.07844352 -0.3497708   0.27590138]
 [ 0.17322061 -0.45563728  0.35242301  0.0547739  -0.35575798  0.05401703]
 [ 0.05717394  0.17007092  0.48121257  0.04911295 -0.17609779 -0.62555738]
 [ 0.2844251   0.01419526  0.47525703 -0.06343453  0.31582691 -0.05230119]
 [ 0.39984124  0.18963655  0.25310152 -0.06498405  0.2150702  -0.00265839]
 [ 0.29355595  0.1891525  -0.24305282  0.28554422 -0.18532988  0.05511879]
 [ 0.356629   -0.01712421 -0.20764231  0.09672331  0.10611263 -0.03422194]
 [ 0.37891541  0.24845341 -0.1187667  -0.20504369 -0.15638517  0.17314828]
 [-0.01109383 -0.20530293 -0.07045228  0.80365507  0.34299286 -0.17531222]
 [-0.11508371 -0.34317293  0.09199914 -0.30080388  0.60036837  0.16978256]
 [-0.1125137  -0.30853263 -0.3261138  -0.30337798 -0.07990471 -0.62630723]]

Variances_PDs=
 [ 4.21863285  2.37810068  1.878226    1.10938969  0.91004708  0.81541317]
```

```
Sparse_Principal_Directions:
[[ 0.47674415  0.          0.         -0.         -0.          0.        ]
 [ 0.49506433  0.          0.         -0.         -0.          0.        ]
 [-0.         -0.80527358  0.         -0.         -0.          0.        ]
 [ 0.         -0.59290342 -0.         -0.         -0.          0.        ]
 [-0.11517843 -0.          0.61724992  0.          0.          0.        ]
 [-0.          0.          0.58087446 -0.         -0.         -0.        ]
 [ 0.19375402 -0.          0.5144896   0.          0.         -0.        ]
 [ 0.34473217 -0.          0.         -0.          0.         -0.        ]
 [ 0.41425518 -0.          0.         -0.         -0.         -0.        ]
 [ 0.43170824 -0.         -0.          0.          0.         -0.        ]
 [-0.          0.          0.          1.         -0.          0.        ]
 [ 0.          0.          0.         -0.          1.          0.        ]
 [ 0.          0.         -0.12995327 -0.         -0.         -1.        ]]
```

Outline

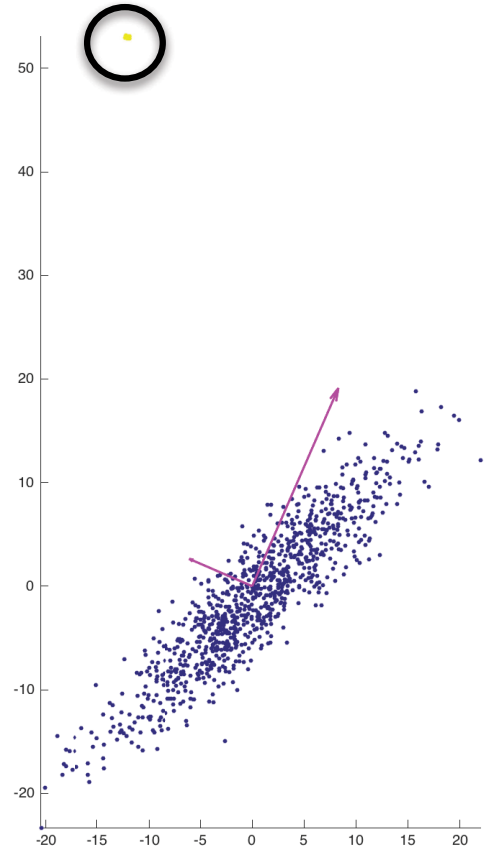
- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- **Robust PCA**
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- Conclusion

Robust PCA

Q: Is PCA robust to outliers?

- **Motivation:** Standard/sparse PCA are **sensitive to outliers**, i.e. a single outlier may change significantly the true PCA solution.

- **Solution:** Robust PCA is a technique that **separates the outliers from the clean data** where PCA is performed.



Formalization

- **Standard PCA:** $\min_{L} \|X - L\|_F^2 \quad \text{s.t.} \quad \text{rank}(L) = K$

- **Robust PCA:** $\min_{L,S} \underbrace{\text{rank}(L) + \lambda \, \text{card}(S)}_{\text{NP-hard combinatorial problem}} \quad \text{s.t.} \quad X = L + S \quad (1)$

  $\Rightarrow$ Continuous relaxation needed.

  Data

  Low-rank matrix
  Standard PCA
  (structure)

  Sparse matrix
  captures outliers
  (no structure)

  $\Downarrow$ Convex relaxation

$$\min_{L,S} \|L\|_\star + \lambda \, \|S\|_1 \quad \text{s.t.} \quad X = L + S \quad (2)$$

Strong result: *Solution of (2) is almost the solution of (1)*

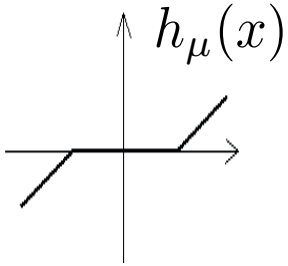
Algorithm

- **ADMM technique:** Fast, robust and accurate solutions.

Initialization: $L^{m=0} = X \qquad S^{m=0} = Z^{m=0} = 0$

Iterate until convergence:

$$\begin{cases} L^{m+1} = U h_{1/r}(\Lambda) V^T \quad \text{with} \quad U\Lambda V^T \overset{svd}{=} X - S^m + Z^m/r \\ S^{m+1} = h_{\lambda/r}\left(X - L^{m+1} + Z^m/r\right) \\ Z^{m+1} = Z^m + r(X - L^{m+1} - S^{m+1}) \end{cases}$$

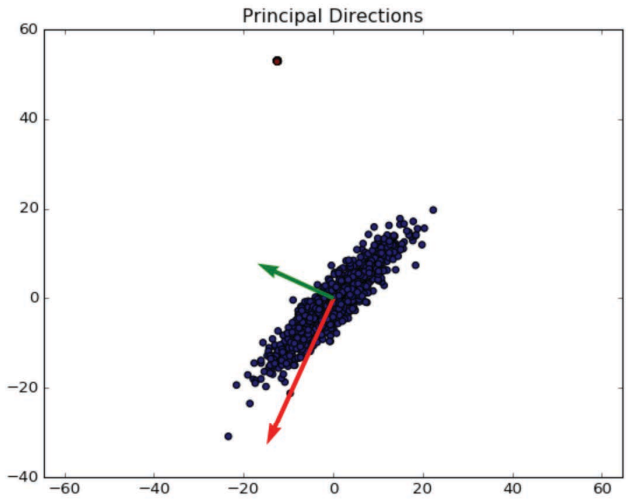


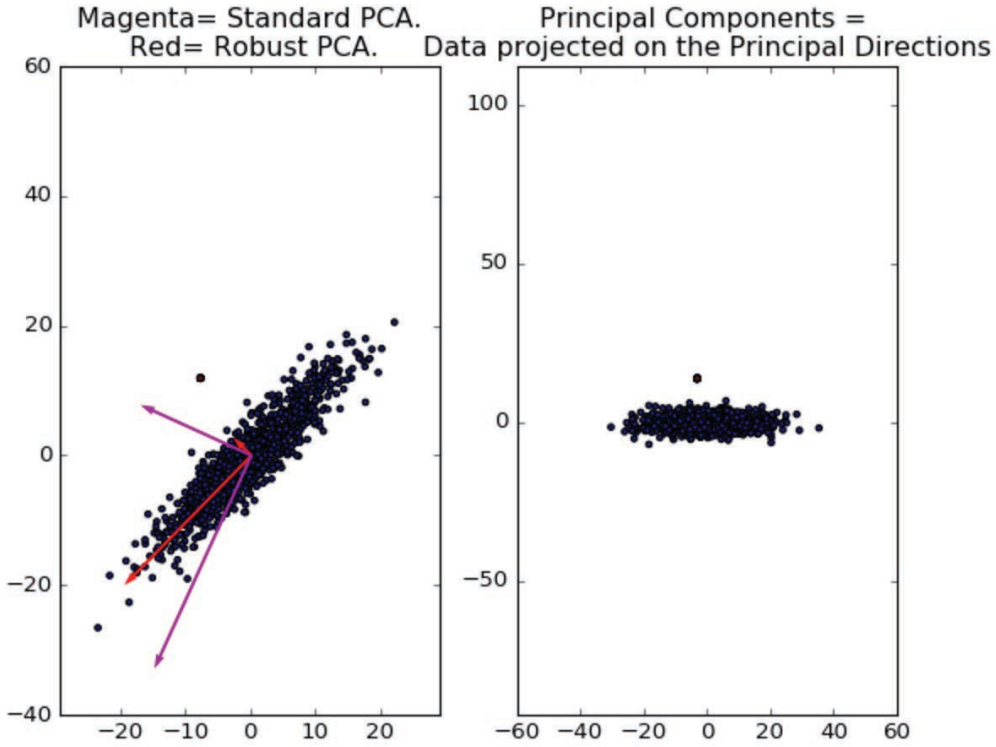
Demo: Robust PCA

- Run **code03.ipynb**

```
# Run Robust PCA
X = Xref - np.mean(Xref,axis=0)
L = X
S = np.zeros(X.shape)
Z = np.zeros(X.shape)
n,m = X.shape
min_nm = np.min([n,m])
r = 1
lambdaN = 1.
lambdaS = 0.1
for i in range(1000):

    # Update L
```





Outline

- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- Robust PCA
- **PCA on Networks**
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- Conclusion

PCA on Graphs

Q: Can do PCA on networks like Facebook?

- **Motivation:** When **data similarities** are available or can be computed, it enhances PCA.

- **Formalization:**

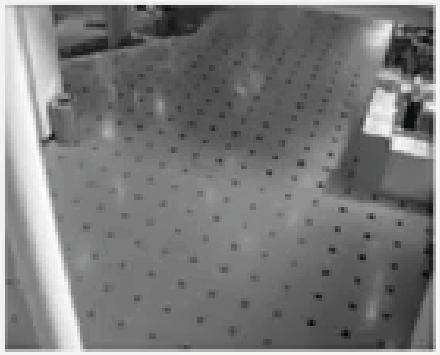
$$\min_{L,S} \ \text{rank}(X) + \lambda_s \ \text{card}(S) + \lambda_G \underbrace{\|L\|_{G\_smooth}}_{\text{Force smoothness on graphs}} \ \text{s.t.} \quad X = L + S$$

$$\Downarrow \quad \text{Continuous convex relaxation}$$

$$\min_{L,S} \ \|X\|_\star + \lambda_s \ \|S\|_1 + \lambda_G \|L\|_{G\_Dir} \ \text{s.t.} \quad X = L + S$$

Demo – Video Surveillance

- Separate background from **moving objects**: State-of-the-art [ICCV'15]


Outline

- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- Robust PCA
- PCA on Networks
- **Non-Negative Matrix Factorization (NMF)**
- Sparse Coding (SC)
- Conclusion

Non-Negative Matrix Factorization

Q: Is PCA the best linear data representation?

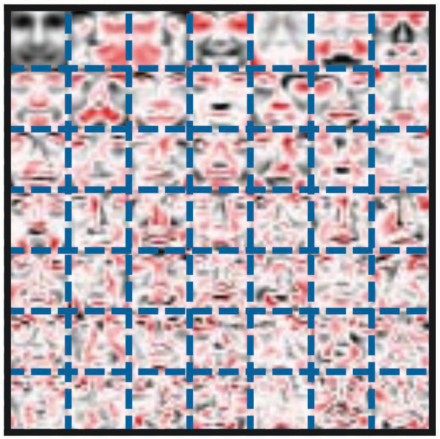
- **Motivation:** PCA vs. NMF

  PCA learns the ***main variations*** *of data.*

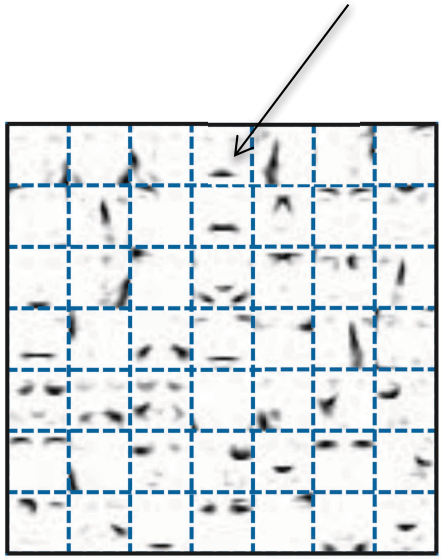
  ⇒ Data representation is based on main directions of data variations.

  NMF learns the ***most common parts*** *of data.*

  ⇒ Data representation is based on main common data parts.



PCA



NMF

[Lee-Seung'99]

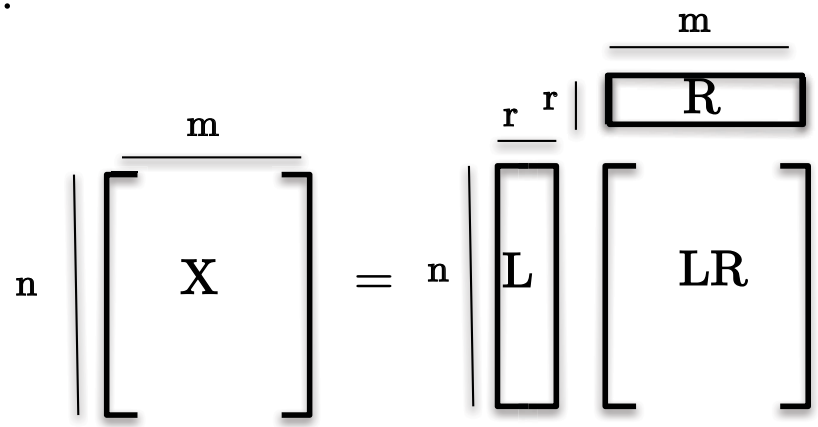
Matrix Factorization

- PCA and NMF are both **factorized** models:

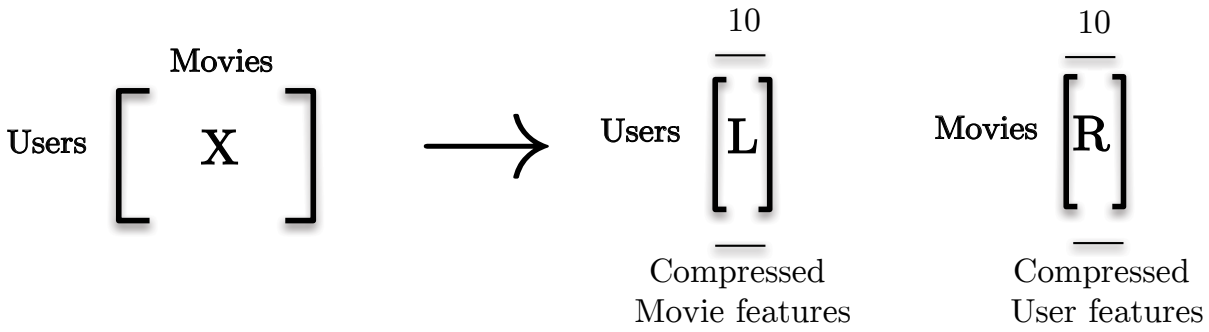
$$\text{PCA} : X \overset{svd}{=} U\Lambda V^T$$

$$\text{NMF} : X = LR \;\; \text{with} \;\; \underbrace{L, R \geq 0}$$

Essential constraints to Identify data parts

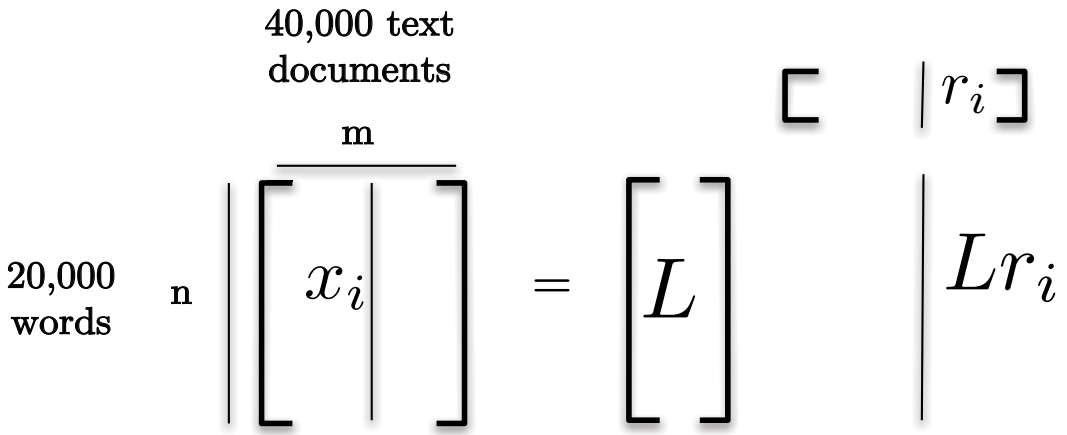


- **Dimensionality reduction technique:** L,R are small low-rank matrices (compared to X). They can be interpreted as **compressed features!**



Linear Representation

- Text document representation:



$$x_i = Lr_i$$

$\Rightarrow$ ***Each document is represented by a linear combination of compressed word features.***

How to Compute L,R?

- Factorization of the form: $X = LR \;\text{ with }\; L, R \geq 0$

  can be solved by optimization these loss functions:

  (1) Least-squares loss:

$$\min_{L,R \geq 0} \; \|X - LR\|_F^2$$

  (2) Kullback-Leibler (relative entropy) loss: (histogram distances)

$$\min_{L,R \geq 0} \; \mathrm{KL}(X, LR) = -\sum_{ij} X_{ij} \log \frac{(LR)_{ij}}{X_{ij}}$$

Algorithms

- **Several techniques exist:**

  (11) Multiplicative update techniques

  Advantage: Monotonic.

  Limitation: Slow to converge.

  (2) ADMM, Primal-Dual techniques

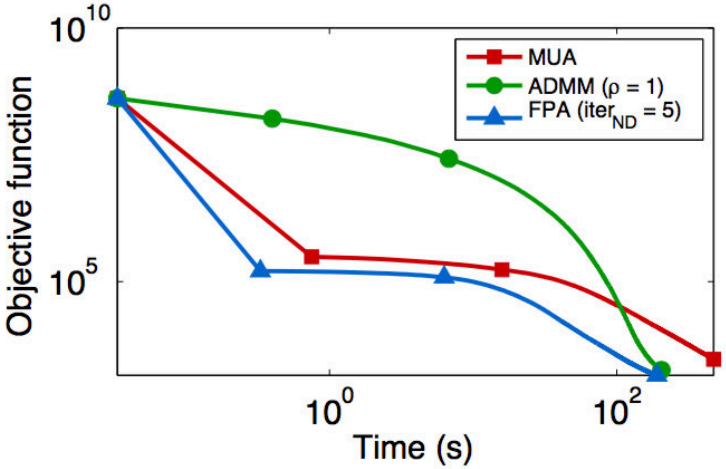
  Advantage: Fast.

  Limitation: No theoretical guarantee.

  (3) Power Methods (most recent)

  Advantage: Fast.

  Limitation: No theoretical guarantee.



Objective function versus time.

Outline

- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- Robust PCA
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- **Sparse Coding (SC)**
- Conclusion

Sparse Coding

Q: Can do better than PCA and NMF?

- **Motivation:** PCA and NMF make **strict properties** about the dictionary D used for linear representation:

$$z = Dx$$

dictionary

PCA
D captures **main directions of data variations**

NMF
D captures **main common parts of data**

Q: How to relax these properties to learn more generic filters?

- **Sparse coding** (recently a.k.a. **dictionary learning**):

  New data assumption: **Represent data as a sparse linear combination of a few filters.**

- **Note:** This is the **best** linear representation and feature extraction technique, and **for any kind of data**. A class of deep learning techniques use sparse coding at core feature extraction *"deconvolutional neural networks."*

Formalization

- **Optimization problem:**

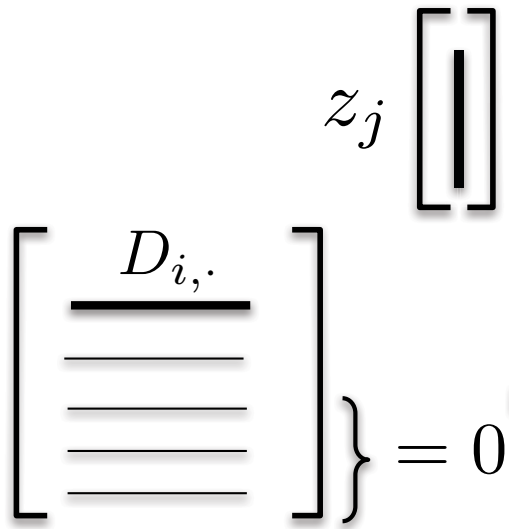
$$\min_{D,z_j} \sum_{j=1}^{n} \|x_j - Dz_j\|_2^2 + \lambda \|z_j\|_1 \quad \text{s.t.} \quad \|D_{i,.}\|_2 \leq 1 \ \forall i$$

Forces sparsity

Controls filter energies

$$\text{En}_i = \|D_{i,.}\|_2 = \begin{cases} 1 \\ 0 \end{cases}$$

Algorithm can learn the **best number** of filters



Algorithm

- **Non-smooth and convex optimization:**

$$\min_{Z,D} \sum_{j=1}^{n} \|x_j - Dz_j\|_2^2 + \lambda\|z_j\|_1 \quad \text{s.t.} \quad \|D_{i,.}\|_2 \leq 1 \ \forall i$$

$$\Updownarrow$$

$$\min_{Z,D} \|X - DZ\|_F^2 + \lambda\|Z\|_1 \quad \text{s.t.} \quad \|D_{i,.}\|_2 \leq 1 \ \forall i$$

Initialization: $D^{m=0} = \text{randn}$

Iterate until convergence:

$$\begin{cases} Z^{m+1} = \arg\min_{Z} \|X - D^m Z\|_F^2 + \lambda\|Z\|_1 \\ D^{m+1} = \arg\min_{D} \|X - DZ^{m+1}\|_F^2 \quad \text{s.t.} \quad \|D_{i,.}\|_2 \leq 1 \ \forall i \end{cases}$$

$\Rightarrow$ *Each sub-optimization problems can be solved efficiently by FISTA.*

Demo: Sparse Coding

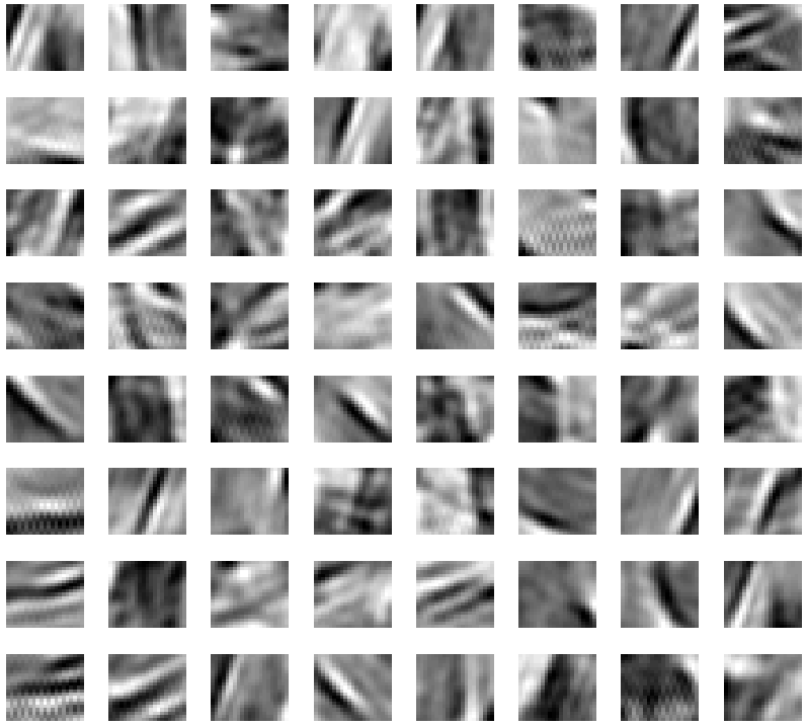
- Run code04.ipynb

Emergence of simple-cell receptive field properties by learning a sparse code for natural images

Bruno A. Olshausen* & David J. Field

We show that a learning algorithm that attempts to find sparse linear codes for natural scenes will develop a complete family of localized, oriented, bandpass receptive fields, similar to those found in the primary visual cortex.

```
# Sparse coding Algorithm
# Parameter
m = 64 # number of basis functions in D
[d,n] = X.shape # d = data dimentionality, n = number of data
print(d,n)
lambdaDF = 1e-1
```



Learned Dictionary=
Human visual filters (V1 cells)
in the primary visual cortex

Outline

- The Feature Extraction Problem
- Standard Principal Component Analysis (PCA)
- Sparse PCA
- Robust PCA
- PCA on Networks
- Non-Negative Matrix Factorization (NMF)
- Sparse Coding (SC)
- **Conclusion**

Summary

- **Feature Extraction Problem:**
  - *(1) Handcrafted filters/features: less popular.*
  - *(2) Learned filters/features: more and more common.*

- **Learned filters = Data representation problem:**
  - *(1) Linear Representations*
  - *(2) Non-linear representations*

- ***Linear Representations:***
  - *(1) PCA:* based on data variances.
  - *(2) NMF:* based on common parts of data
  - *(3) Sparse Coding:* based on sparse representation (highly adaptable technique)

Questions?