Data Pre-processing

Google Machine Learning Study.
2018.07.28 Daeyoung Kim

CONTENTS

WHY DATA PREPROCESSING?

"***Dirty Data***" *is often seen in real world.*

incomplete

irrelevant

noisy

unreliable

Inaccurate prediction

cleaning **/** normalization **/** transformation **/** feature extraction **/** etc.

STEP 1

DATA CLEANING

00 TASK OF DATA CLEANING

1. Fill in missing values

2. Identify outlier and smooth noisy data

3. Correct inconsistent data

01 FILL IN MISSING VALUES

1. **Ignore** the tuple

2. Fill in the missing value **manually**

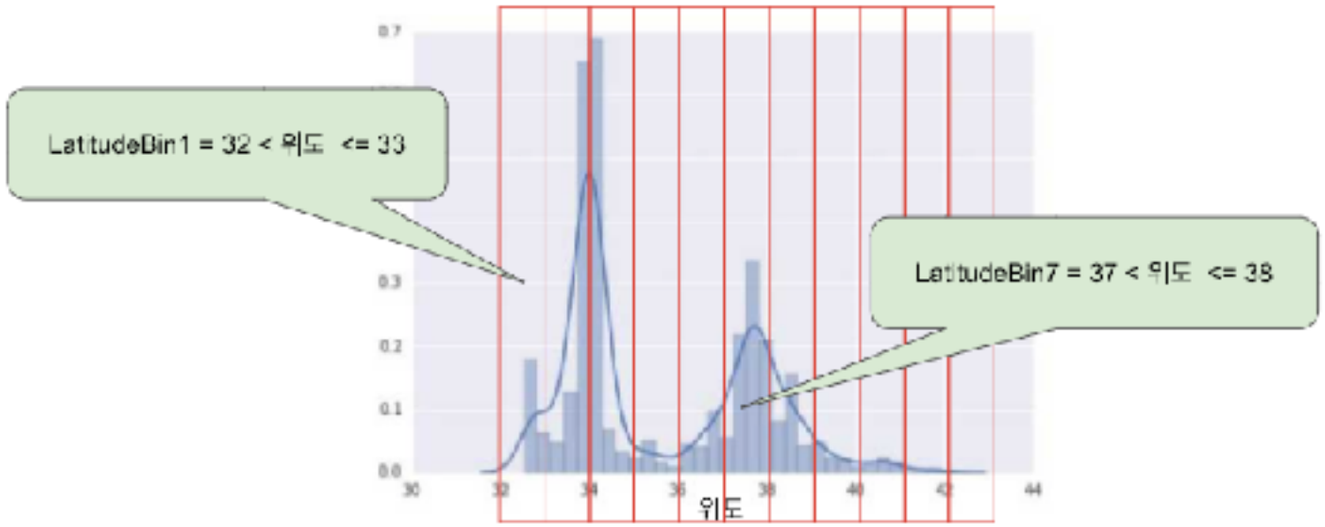
3. Use a **global constant** to fill the missing value

4. Use the attribute **mean**

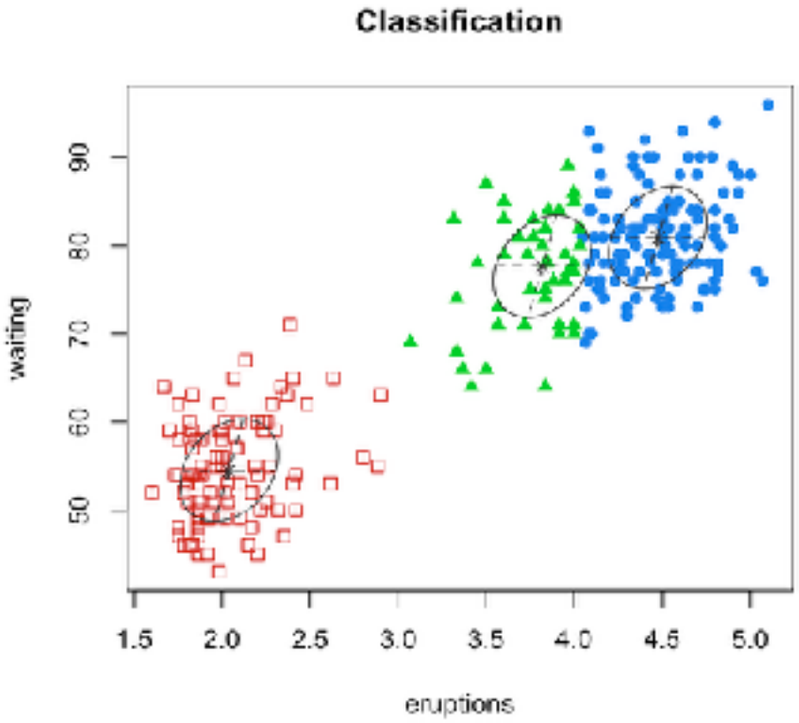
5. Use the **most probable value**

02 **SMOOTH** NOISY DATA

1. **Binning** method



2. Clustering



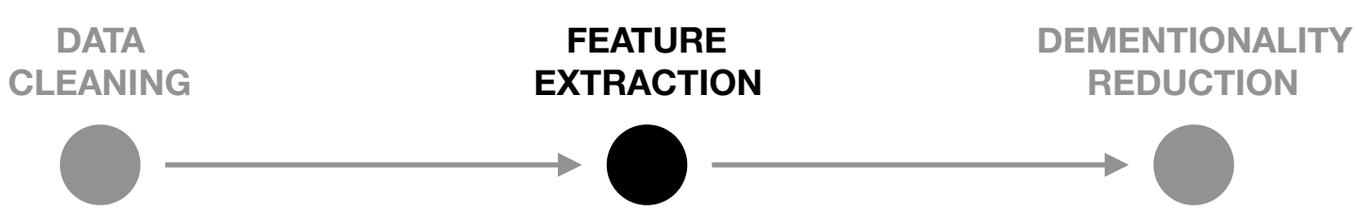
03 **CORRECT** INCONSISTENT DATA

1. Manually detection with external reference

2. Semi-automatic using various tools

STEP 2

FEATURE EXTRACTION



00 **SELECT** RELEVANT FEATURES

Discard irrelevant features

Example: Select features for predicting milage of a car

Engine Capacity	**(O)**
Top Speed	**(O)**
Color	(X)

01 FEATURE EXTRACTION FOR **TEXT**

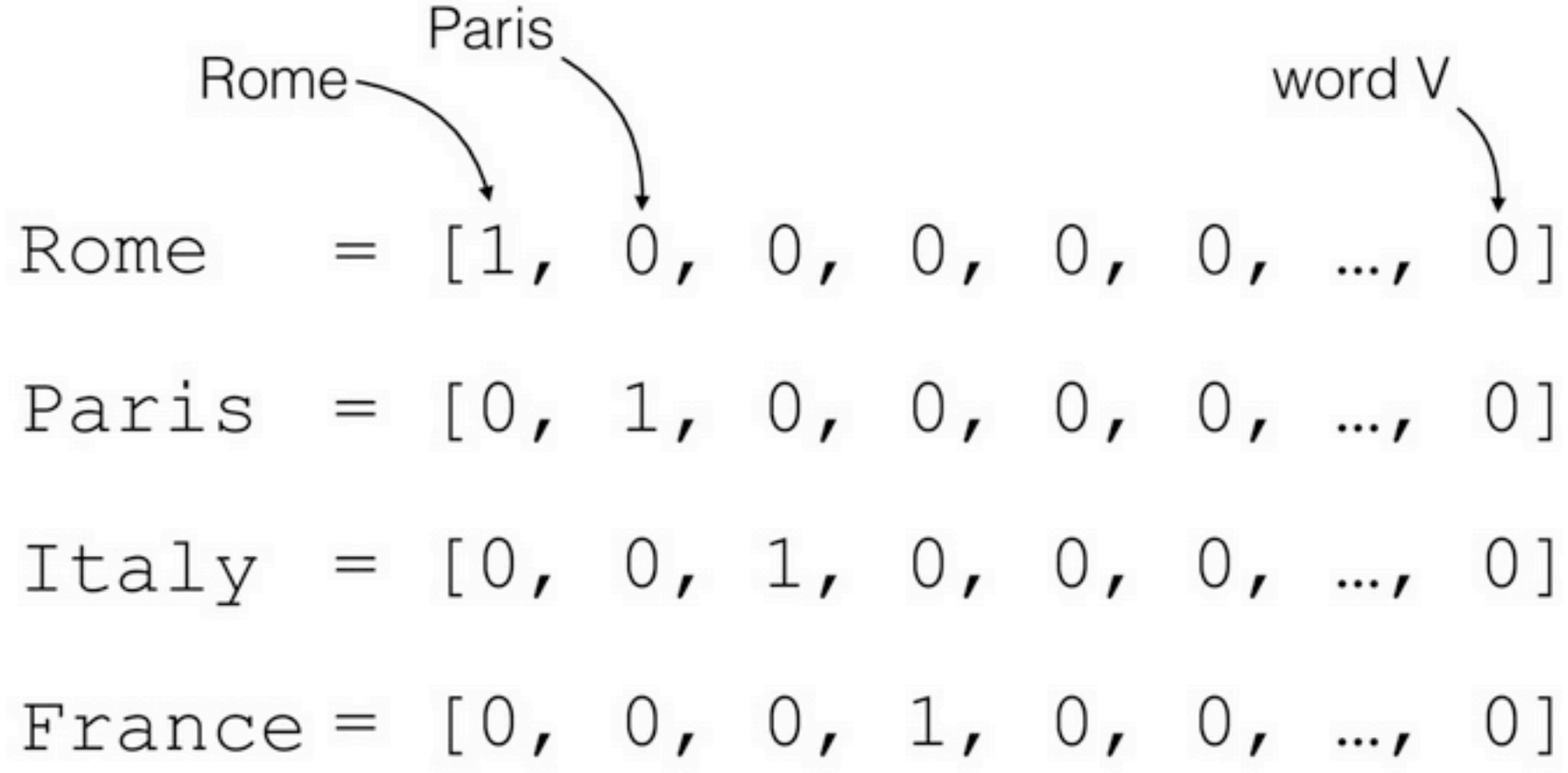
1. **One-hot** vector

2. **Bag of Words (BOW)**

   - **Count** vectorizer
   - **TF-IDF** vectorizer

3. **Word Embeddings**

01-1 ONE-HOT VECTOR



01-2 BAG OF WORDS (**BOW**)

"*The quick brown fox jumps over the lazy dog*"

"*Never jump over the lazy dog quickly*"

Dictionary

```
{
    brown: 0,
    dog: 1,
    fox: 2,
    jump: 3,
    jumps: 4,
    lazy: 5,
    never: 6,
    over: 7,
    quick: 8,
    quickly: 9,
    the: 10,
}
```

Vectorization →

[1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2]

[0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1]

< Count Vectorizer >

01-2 BOW: **TF-IDF**

Term **F**requency - **I**nverse **D**ocument **F**requency

$$tf-idf(d,t) = tf(d,t) \cdot idf(t)$$

$$tf(d,t) : term\ frequncy$$

$$df(t) : document\ frequncy$$

$$n : \sharp\ of\ documents$$

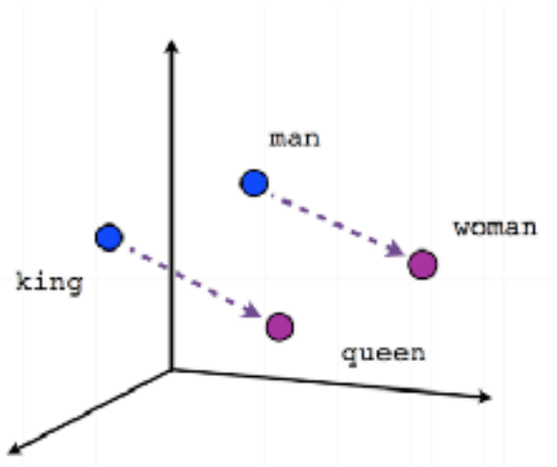
$$idf(d,t) = log\frac{n}{1+df(t)}$$

```
In [1]: from sklearn.feature_extraction.text import TfidfVectorizer
        corpus = [
            'The quick brown fox jumps over the lazy dog',
            'Never jump over the lazy dog quickly'
        ]
```

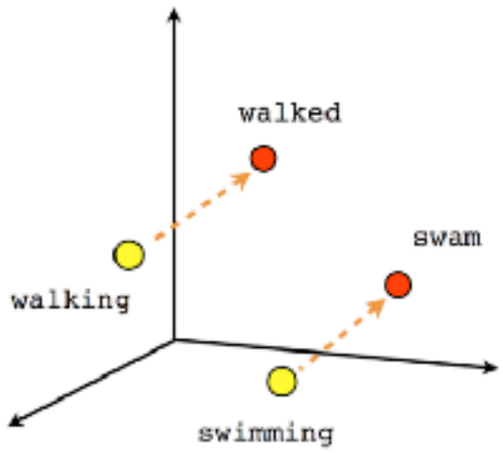
```
In [2]: tfidv = TfidfVectorizer().fit(corpus)
        tfidv.transform(corpus).toarray()
```

```
Out[2]: array([[0.36408901, 0.25905233, 0.36408901, 0.        , 0.36408901,
                0.25905233, 0.        , 0.25905233, 0.36408901, 0.        ,
                0.51810466],
               [0.        , 0.3174044 , 0.        , 0.44610081, 0.        ,
                0.3174044 , 0.44610081, 0.3174044 , 0.        , 0.44610081,
                0.3174044 ]])
```

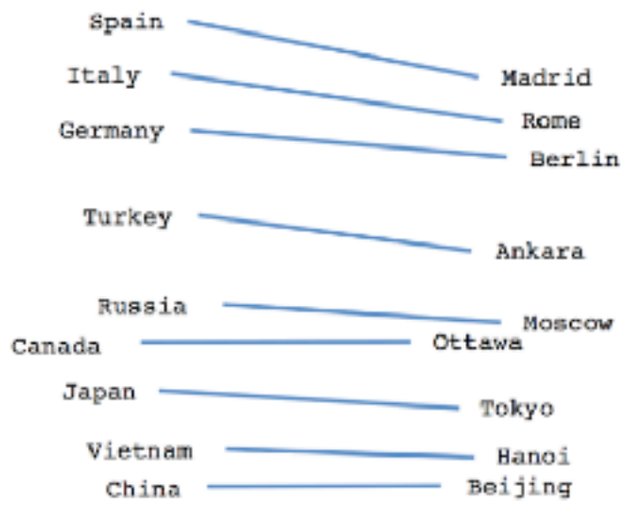
01-3 WORD EMBEDDING (**Word2vec**)



Male-Female



Verb tense



Country-Capital

02 FEATURE EXTRACTION FOR **IMAGE**

Mainly **dependent on the type** of dataset / images

Mean Subtraction **/** Normalization **/** PCA **/** Whitening

Low-level

- Edge detection
- Corner detection
- Blob detection
- Ridge detection
- Scale-invariant feature transform

Shape based

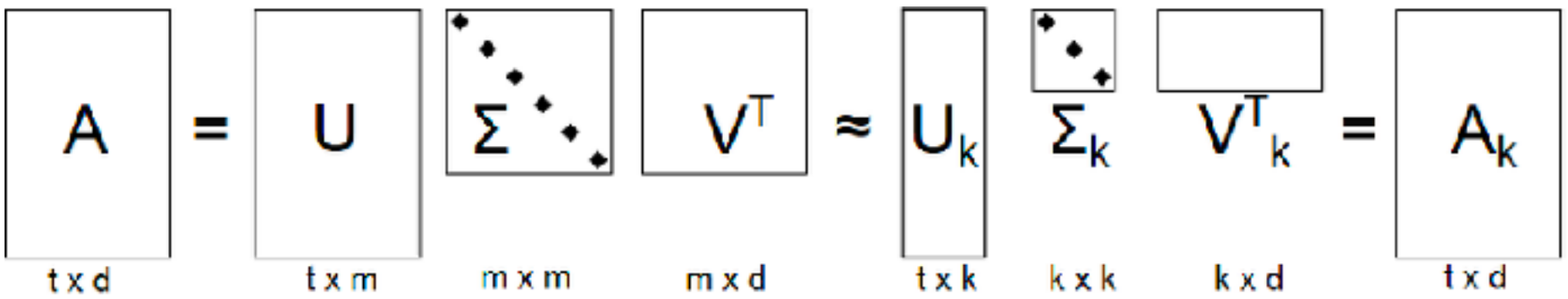
- Thresholding
- Blob extraction
- Template matching
- Hough transform

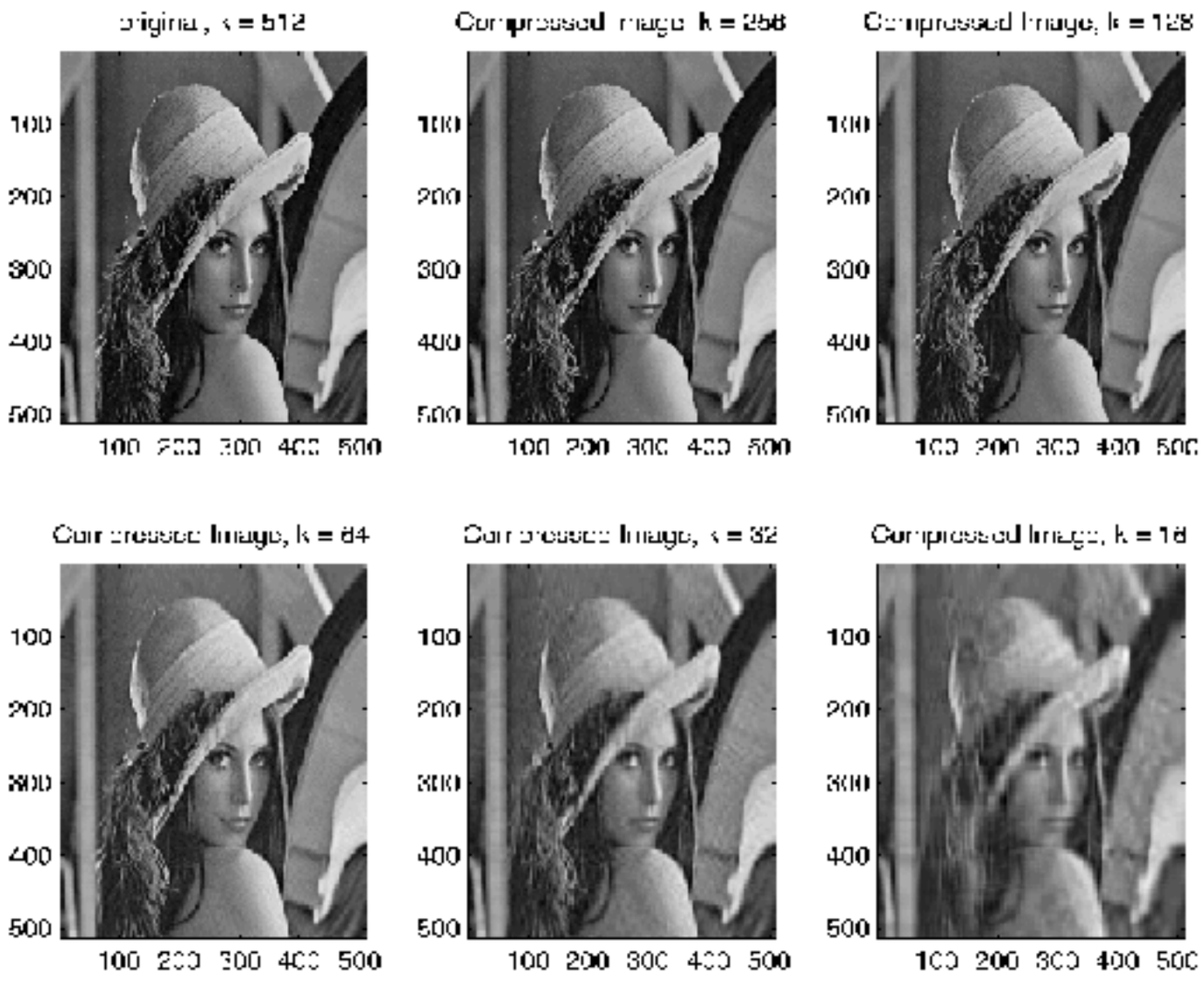
STEP 3

DEMENTIONALITY REDUCTION

01

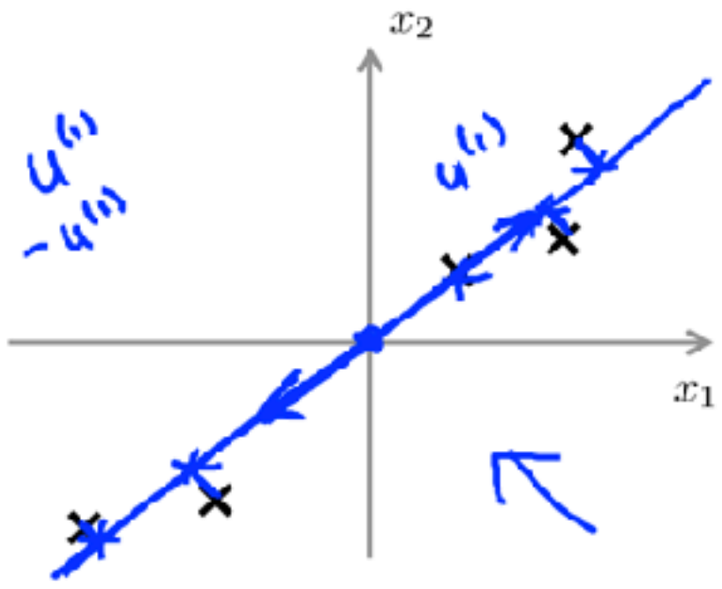
SINGULAR VALUE DECOMPOSITION (**SVD**)



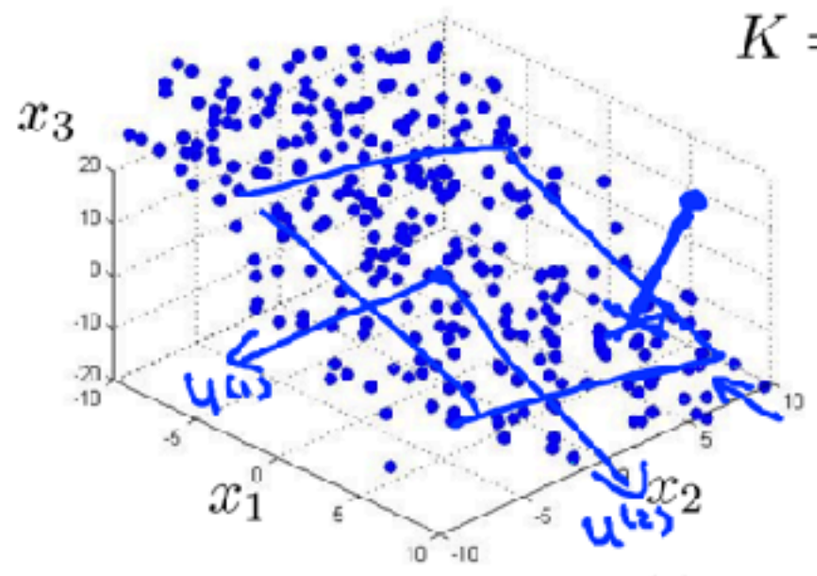


02 PRINCIPLE COMPONENT ANALYSIS (**PCA**)

Principal Component Analysis (PCA) problem formulation



$3D \rightarrow 2D$
$K = 2$



Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Andrew Ng

02 PRINCIPLE COMPONENT ANALYSIS (**PCA**)

Principal Component Analysis (PCA) algorithm summary

→ After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

Sigma = $\frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$

$X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$

Sigma = (1/m) * X' * X;

→ `[U,S,V] = svd(Sigma);`

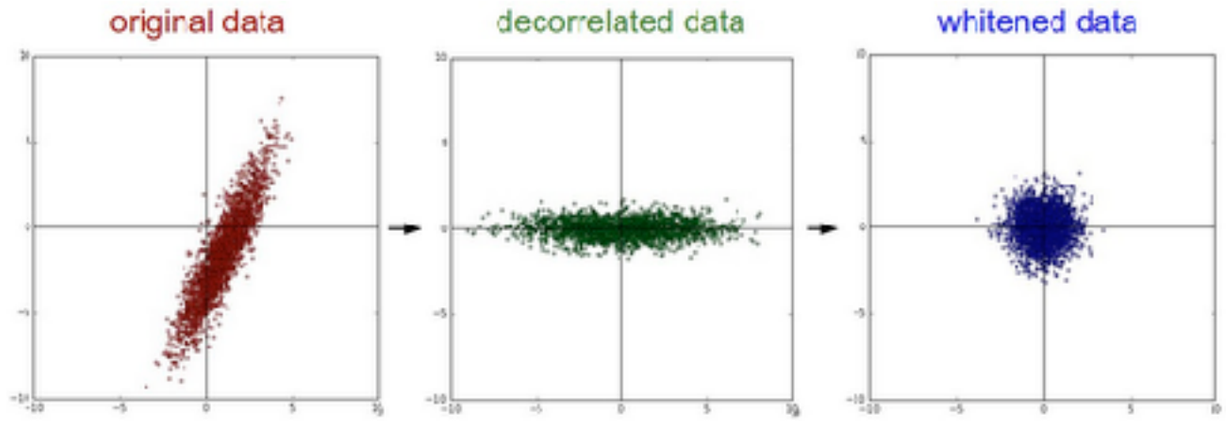
→ `Ureduce = U(:,1:k);`

→ `z = Ureduce' *x;`

$x \in \mathbb{R}^n$ ~~$x_0 = 1$~~

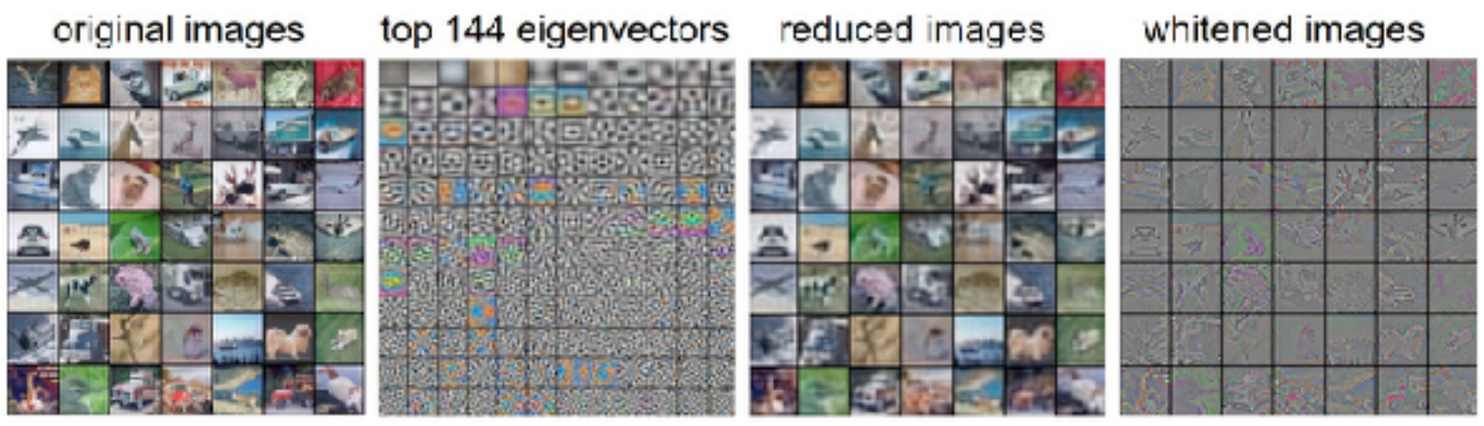
02 PRINCIPLE COMPONENT ANALYSIS (**PCA**)

PCA & Whitening




```
# Assume input data matrix X of size [N x D]
X -= np.mean(X, axis = 0) # zero-center the data (important)
cov = np.dot(X.T, X) / X.shape[0] # get the data covariance matrix
U,S,V = np.linalg.svd(cov)
Xrot = np.dot(X, U) # decorrelate the data
Xrot_reduced = np.dot(X, U[:,:100]) # Xrot_reduced becomes [N x 100]

# whiten the data:
# divide by the eigenvalues (which are square roots of the singular values)
Xwhite = Xrot / np.sqrt(S + 1e-5)
```



03 AND SO ON...

- Independent Component Analysis (**ICA**)
- Non-negative Matrix Factorization (**NMF**)
- Eigen Decomposition
- Random Projection
- Factor Analysis (**FA**)

Reference

1. S. Kotsiantis, D. Kanellopoulos, P. Pintelas,

"Data Preprocessing for Supervised Learning", *International Journal of Computer Science*, 2007, Vol 1

Link**:** https://pdfs.semanticscholar.org/c640/1e515a58fc36c37fc97e3b0cb18ce4682743.pdf

2. Data Preprocessing Techniques for Data Mining

Link: http://iasri.res.in/ebook/win_school_aa/notes/Data_Preprocessing.pdf

3. Data Preprocessing Steps for Machine Learning & Data analytics

Link: https://www.youtube.com/watch?v=NBm4etNMT5k

4. Scikit-Learn의 문서 전처리 기능

Link: https://datascienceschool.net/view-notebook/3e7aadbf88ed4f0d87a76f9ddc925d69/

5. Image Compression with SVD

Link: http://fourier.eng.hmc.edu/e161/lectures/svdcompression.html

6. CS231n (CNN for Visual Recognition), Setting up the data and the model

Link: http://cs231n.github.io/neural-networks-2/

7. Machine Learning by Andrew Ng, Dimensionality Reduction

Link: https://www.coursera.org/learn/machine-learning/home/welcome