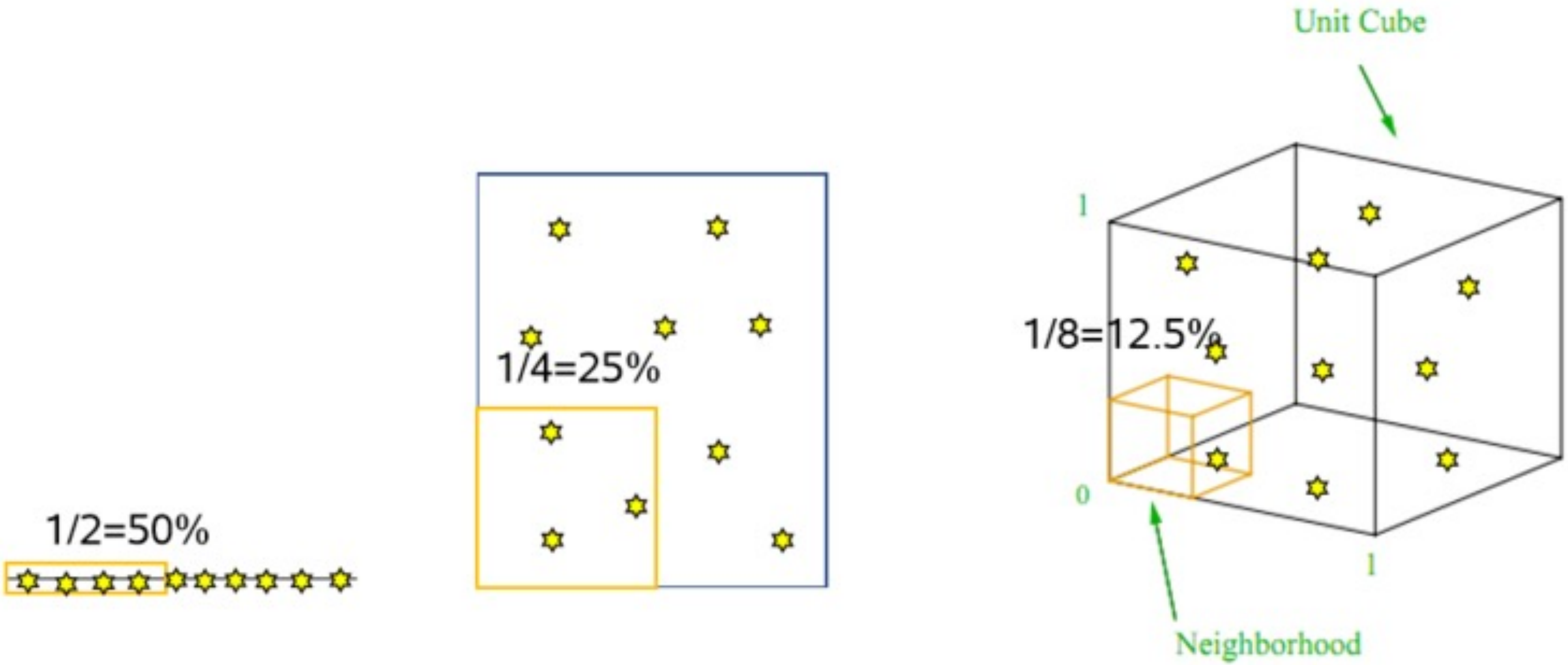
차원축소와 군집화

이준희, 김현우

Contents

Curse of Dimensionality



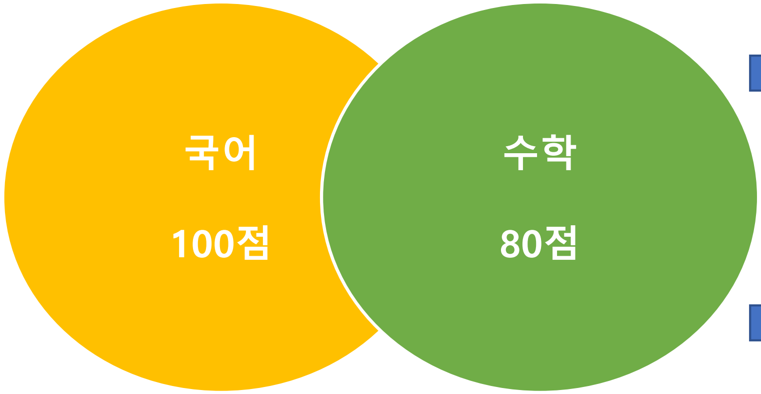
Feature Selection vs Feature Extraction

키	몸무게	흡연 유무	평균 기상 시각	예상 수명
180	70	1	10	80
170	60	0	7	120

키	~~몸무게~~	~~흡연 유무~~	평균 기상 시각	예상 수명
180	~~70~~	~~1~~	10	80
170	~~60~~	~~0~~	7	120

키	몸무게	바른 생활 습관	예상 수명
180	80	50	80
170	60	100	120

Principal Component Analysis



$(0.5, 0.5) \Rightarrow 100 \times 0.5 + 80 \times 0.5 = 90$

$(0.6, 0.4) \Rightarrow 100 \times 0.6 + 80 \times 0.4 = 92$

Which one would make better MEAN???
How Do we know??
=> Feature Extraction

Principal Component Analysis

x축 방향으로 퍼진 정도

x, y축 방향으로
함께 퍼진 정도

$$\begin{bmatrix} 3 & 2 \\ 2 & 4 \end{bmatrix}$$

x, y축 방향으로
함께 퍼진 정도

y축 방향으로 퍼진 정도

데이터 : $X \in \mathbb{R}^{n \times d}$, e에 정사영된 데이터 : $Xe \in \mathbb{R}^{n \times 1}$

$V(Xe) = \frac{1}{n}\Sigma(Xe - E(Xe))^2 = \frac{1}{n}\Sigma(Xe)^2 \quad (E(Xe) = 0)$

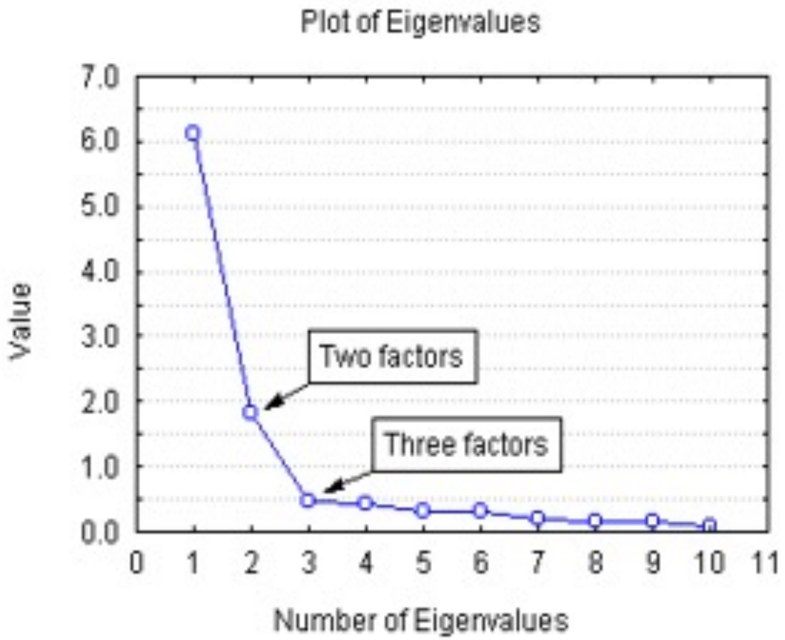
$V(Xe) = \frac{1}{n}(Xe)^T(Xe) = e^T\left(\frac{X^TX}{n}\right)e = e^T\Sigma e \quad (|e|^2 = 1)$

cf) $L = e^T\Sigma e - \lambda(|e|^2 - 1)$, $\frac{\partial}{\partial e}L = 2\Sigma e - 2\lambda e = 0 \quad \therefore \Sigma e = \lambda e$

$V(Xe) = e^T\Sigma e = e^T\lambda e = \lambda e^T e = \lambda$

$\therefore$ EigenVector를 통한 정사영 ⟶ variance가 eigenvalue임

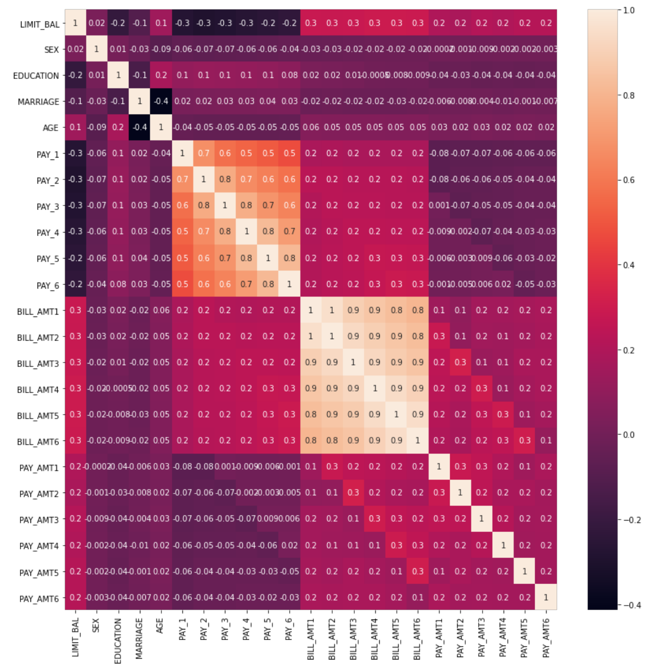
Principal Component Analysis



$$\frac{\sum_{j=1}^{m} \lambda_j}{\sum_{i=1}^{d} \lambda_i} = 0.9$$

Principal Component Analysis Example

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 1 to 30000
Data columns (total 24 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   LIMIT_BAL   30000 non-null  object
 1   SEX         30000 non-null  object
 2   EDUCATION   30000 non-null  object
 3   MARRIAGE    30000 non-null  object
 4   AGE         30000 non-null  object
 5   PAY_1       30000 non-null  object
 6   PAY_2       30000 non-null  object
 7   PAY_3       30000 non-null  object
 8   PAY_4       30000 non-null  object
 9   PAY_5       30000 non-null  object
 10  PAY_6       30000 non-null  object
 11  BILL_AMT1   30000 non-null  object
 12  BILL_AMT2   30000 non-null  object
 13  BILL_AMT3   30000 non-null  object
 14  BILL_AMT4   30000 non-null  object
 15  BILL_AMT5   30000 non-null  object
 16  BILL_AMT6   30000 non-null  object
 17  PAY_AMT1    30000 non-null  object
 18  PAY_AMT2    30000 non-null  object
 19  PAY_AMT3    30000 non-null  object
 20  PAY_AMT4    30000 non-null  object
 21  PAY_AMT5    30000 non-null  object
 22  PAY_AMT6    30000 non-null  object
 23  default     30000 non-null  object
```



Principal Component Analysis Example

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

cols_bill = ["BILL_AMT" + str(i) for i in range(1, 7)]
print("대상 속성명 : ", cols_bill)

scaler = StandardScaler()
df_cols_scaled = scaler.fit_transform(X_features[cols_bill])
pca = PCA(n_components = 2)
pca.fit(df_cols_scaled)

print("PCA Component 별 변동성 : ", pca.explained_variance_ratio_)
```

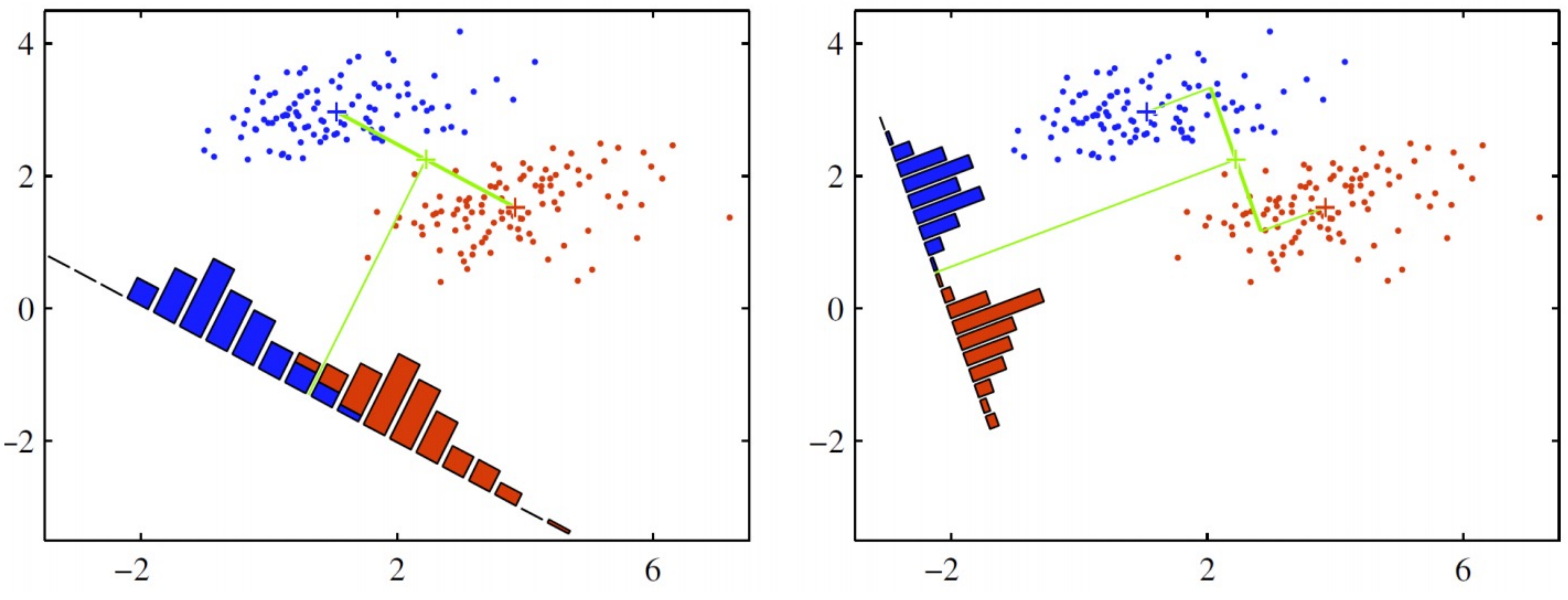
```
대상 속성명 :  ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']
PCA Component 별 변동성 :  [0.90555253 0.0509867 ]
```

Principal Component Analysis Example

	Naïve	PCA 2	PCA 3	PCA 6
Time(sec)	4.5	2.5	2.6	4.4
ACC	81%	79%	79%	80%

Linear Discriminant Analysis

!= Latent Dirichlet Allocation



Linear Discriminant Analysis

$$y = \vec{w}^T \vec{x}$$

$$m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n$$

$$m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n$$

$$J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{w^T S_B w}{w^T S_W w}$$

$$S_B = (m_1 - m_2)(m_1 - m_2)^T$$

$$S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T$$

$$m_2 - m_1 = w^T (m_2 - m_1)$$

$$m_k = w^T m_k$$

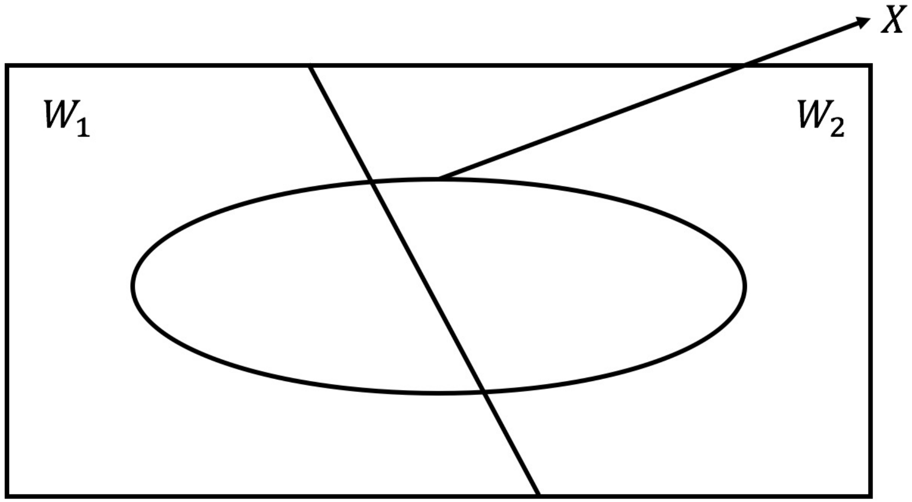
$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

$$S_W w = \lambda S_B w$$

$$S_B^{-1} S_W w = \lambda w$$

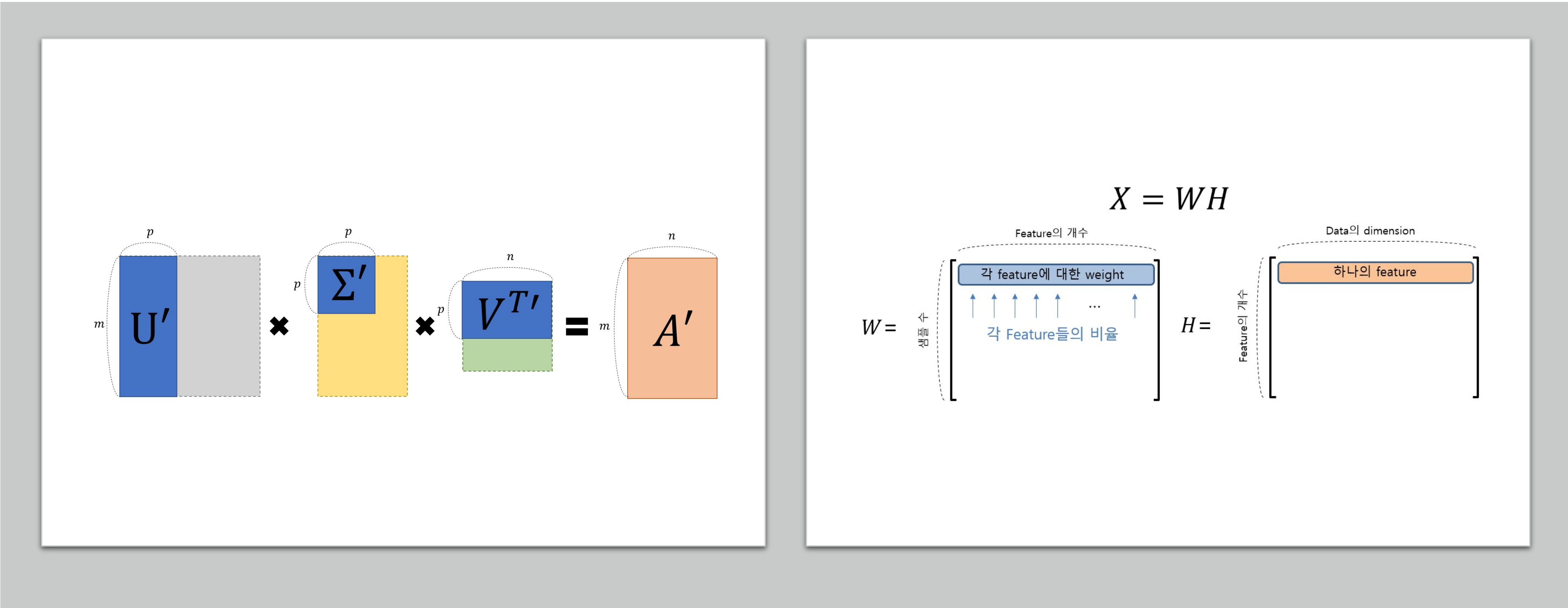
$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$$

Linear Discriminant Analysis

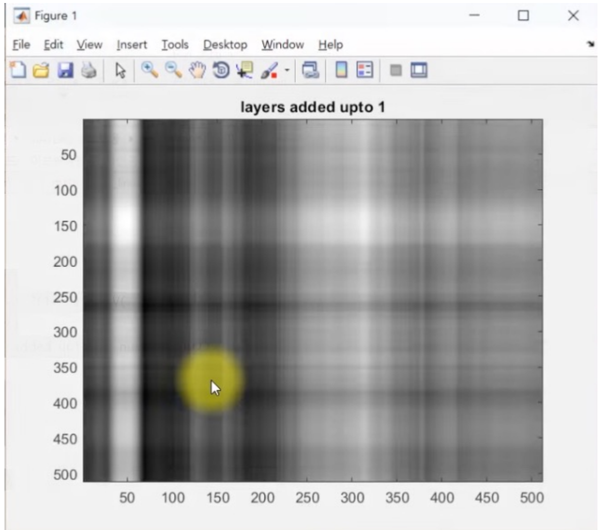


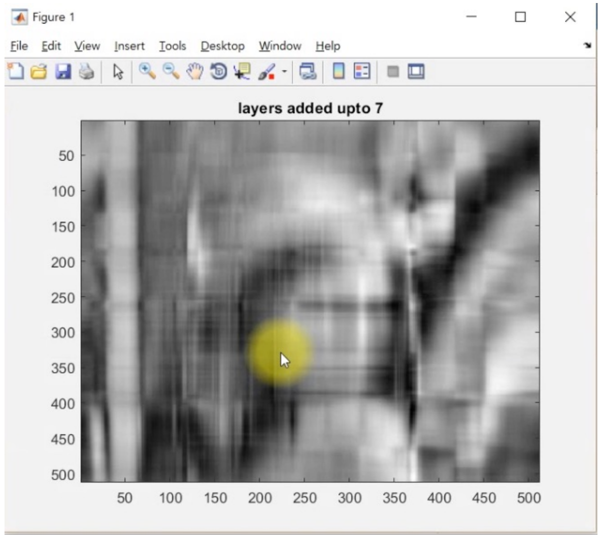
$$P(W_i|x) = \frac{P(x|W_i)P(W_i)}{P(x)}$$
$$= \frac{P(x|W_i)P(W_i)}{P(x|W_1)P(W_1) + P(x|W_2)P(W_2)}$$

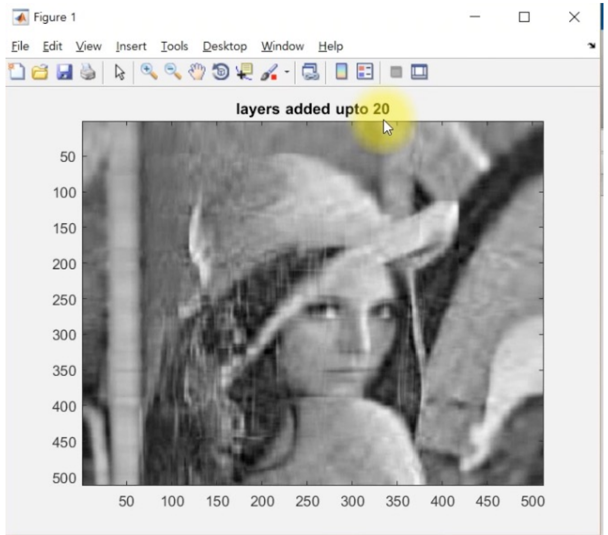
Singular Value Decomposition
Non-Negative Matrix Factorization



Singular Value Decomposition
Non-Negative Matrix Factorization







PCA – Low Level Coding

```
def new_coordinates(X,eigenvectors):
    for i in range(eigenvectors.shape[0]):
        if i == 0:
            new = [X.dot(eigenvectors.T[i])]
        else:
            new = np.concatenate((new,[X.dot(eigenvectors.T[i])]),axis=0)
    return new.T
```

```
def custom_pca(X,number):
    scaler = StandardScaler()
    x_std = scaler.fit_transform(X)
    features = x_std.T
    cov_matrix = np.cov(features)

    eigenvalues = lin.eig(cov_matrix)[0]
    eigenvectors = lin.eig(cov_matrix)[1]

    new_coordinates(x_std,eigenvectors)

    new_coordinate = new_coordinates(x_std,eigenvectors)

    index = eigenvalues.argsort()
    index = list(index)

    for i in range(number):
        if i==0:
            new = [new_coordinate[:,index.index(i)]]
        else:
            new = np.concatenate(([new_coordinate[:,index.index(i)]],new),
                                 axis=0)
    return new.T
```

PCA – Low Level Coding

```
custom_pca(X, 3)
```

```
array([[ 0.31019368, -1.08215716, -0.07983642],
       [ 1.28092404, -0.43132556,  0.13533091],
       [ 1.38766381,  0.78428014, -0.12911446],
       [ 0.95087515, -1.15737142,  1.6495519 ],
       [ 1.84222365,  0.88189889,  0.11493111],
       [-1.12563709, -0.52680338,  0.06564012],
       [-2.71174416,  0.63290138,  0.71195473],
       [-0.03100441, -0.20059783, -0.50339479],
       [ 2.29618509,  0.07661447,  0.01087174],
       [-0.61585248, -0.205764  ,  1.82651199],
       [-1.73320252,  1.29971699,  0.09045178],
       [-0.82366049, -0.57164535, -0.27123176],
       [ 0.75619512,  0.73995175, -0.76710616],
       [-0.42344386,  0.26555394, -1.41533681],
       [-0.39581307, -1.64646874,  0.24104031],
       [-0.88581498,  0.15195119, -0.82271209],
       [ 0.24587691,  0.39139878, -1.15801831],
       [ 0.14741103, -1.22874561, -0.03110396],
       [-0.7161265 , -0.56781471, -0.86180345],
       [ 0.24475107,  2.39442622,  1.19337361]])
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 3)
print(pca.fit_transform(X_scaled))
```

```
[[-0.31019368 -1.08215716 -0.07983642]
 [-1.28092404 -0.43132556  0.13533091]
 [-1.38766381  0.78428014 -0.12911446]
 [-0.95087515 -1.15737142  1.6495519 ]
 [-1.84222365  0.88189889  0.11493111]
 [ 1.12563709 -0.52680338  0.06564012]
 [ 2.71174416  0.63290138  0.71195473]
 [ 0.03100441 -0.20059783 -0.50339479]
 [-2.29618509  0.07661447  0.01087174]
 [ 0.61585248 -0.205764    1.82651199]
 [ 1.73320252  1.29971699  0.09045178]
 [ 0.82366049 -0.57164535 -0.27123176]
 [-0.75619512  0.73995175 -0.76710616]
 [ 0.42344386  0.26555394 -1.41533681]
 [ 0.39581307 -1.64646874  0.24104031]
 [ 0.88581498  0.15195119 -0.82271209]
 [-0.24587691  0.39139878 -1.15801831]
 [-0.14741103 -1.22874561 -0.03110396]
 [ 0.7161265  -0.56781471 -0.86180345]
 [-0.24475107  2.39442622  1.19337361]]
```