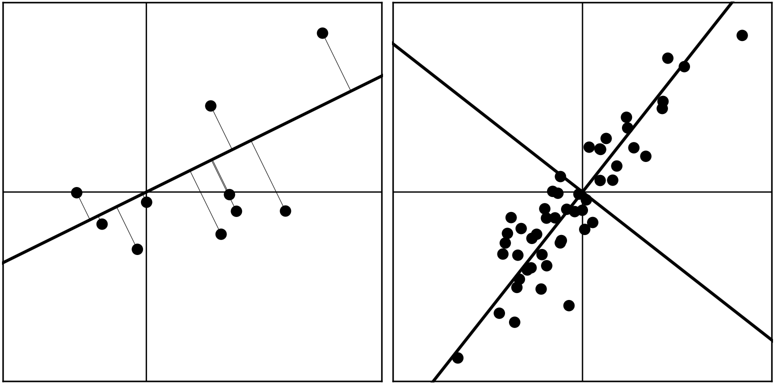
16. Principal Component Analysis (PCA)

Bruce E. Shapiro

Getting Started in Machine Learning

Last revised: March 23, 2019

Dimensionality Reduction and Coordinate Transformation



Derivation of PCA

$$\text{let } \mathbf{X} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ \vdots & \vdots \\ x_{m-1} & y_{m-1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 \\ \hline \vdots \\ \hline \mathbf{x}_{m-1} \end{bmatrix}$$

$$\text{centered } \mathbf{X}' = \begin{bmatrix} x_0 - \mu_x & y_0 - \mu_y \\ x_1 - \mu_x & y_1 - \mu_y \\ \vdots & \vdots \\ x_{m-1} - \mu_x & y_{m-1} - \mu_y \end{bmatrix}$$

$$\text{Scatter Matrix } \mathbf{S} = \mathbf{X}'^{\mathsf{T}}\mathbf{X}'$$

Singular Value Decomposition

Theorem

Singular Value Decomposition *Any* $n \times m$ *matrix* $\mathbf{X}$ *can be uniquely decomposed as a product*

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}$$

where $\mathbf{U}$ *and* $\mathbf{V}$ *are orthogonal matrices and* $\mathbf{\Sigma}$ *is a diagonal matrix.*

Singular Value Decomposition

Theorem

Singular Value Decomposition *Any $n \times m$ matrix $\mathbf{X}$ can be uniquely decomposed as a product*

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\boldsymbol{\Sigma}$ is a diagonal matrix.

Theorem

Let $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ be the SVD of $\mathbf{X}$. Then

$$\mathbf{X} = s_1\mathbf{u}_1\mathbf{v}_1^T + s_2\mathbf{u}_2\mathbf{v}_2^T + \cdots + s_k\mathbf{u}_k\mathbf{v}_k^T$$

where $\mathbf{u}_i$ and $\mathbf{v}_j^T$ are the column and row vectors of $\mathbf{U}$ and $\mathbf{V}$, respectively, and s_j are the diagonal elements of $\boldsymbol{\Sigma}$.

PCA from SVD

Theorem

Principal Component Calculation Theorem. *Let* $\mathbf{X}$ *represent a zero centered data set with singular value decomposition*

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

Then the principal directions of $\mathbf{X}$ *are the column vectors of* $\mathbf{V}$.

Goal: Extract column vectors of $\mathbf{v}$ from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$\mathbf{S} = \mathbf{X}^{\top}\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^{\top})^{\top}(\mathbf{U}\Sigma\mathbf{V}^{\top})$$

Goal: Extract column vectors of $\mathbf{v}$ from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$\begin{aligned}\mathbf{S} &= \mathbf{X}^\top\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^\top)^\top(\mathbf{U}\Sigma\mathbf{V}^\top) \\ &= \mathbf{V}\Sigma\mathbf{U}^\top\mathbf{U}\Sigma\mathbf{V}^\top\end{aligned}$$

Goal: Extract column vectors of $\mathbf{v}$ from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$
\begin{aligned}
\mathbf{S} &= \mathbf{X}^{\mathsf{T}}\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}})^{\mathsf{T}}(\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}}) \\
&= \mathbf{V}\Sigma\mathbf{U}^{\mathsf{T}}\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}} \\
\mathbf{S} &= \mathbf{V}\Sigma^2\mathbf{V}^{\mathsf{T}}
\end{aligned}
$$

Goal: Extract column vectors of $\mathbf{v}$ from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$
\begin{aligned}
\mathbf{S} &= \mathbf{X}^{\mathsf{T}}\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}})^{\mathsf{T}}(\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}}) \\
&= \mathbf{V}\Sigma\mathbf{U}^{\mathsf{T}}\mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}} \\
\mathbf{S} &= \mathbf{V}\Sigma^2\mathbf{V}^{\mathsf{T}} \\
\mathbf{S}\mathbf{V} &= \mathbf{V}\Sigma^2\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{V}\Sigma^2\mathbf{I} = \mathbf{V}\Sigma^2
\end{aligned}
$$

Goal: Extract column vectors of $\mathbf{v}$ from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$
\begin{aligned}
\mathbf{S} &= \mathbf{X}^\mathsf{T}\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^\mathsf{T})^\mathsf{T}(\mathbf{U}\Sigma\mathbf{V}^\mathsf{T}) \\
&= \mathbf{V}\Sigma\mathbf{U}^\mathsf{T}\mathbf{U}\Sigma\mathbf{V}^\mathsf{T} \\
\mathbf{S} &= \mathbf{V}\Sigma^2\mathbf{V}^\mathsf{T} \\
\mathbf{S}\mathbf{V} &= \mathbf{V}\Sigma^2\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{V}\Sigma^2\mathbf{I} = \mathbf{V}\Sigma^2
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{S}\left[\begin{array}{c|c|c|c} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{k-1} \end{array}\right] &= \left[\begin{array}{c|c|c|c} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{k-1} \end{array}\right] \begin{bmatrix} s_0^2 & 0 & \cdots & 0 \\ 0 & s_1^2 & & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & s_{k-1}^2 \end{bmatrix} \\
&= \left[\begin{array}{c|c|c|c} s_0^2\mathbf{v}_0 & s_1^2\mathbf{v}_1 & \cdots & s_{k-1}^2\mathbf{v}_{k-1} \end{array}\right]
\end{aligned}
$$

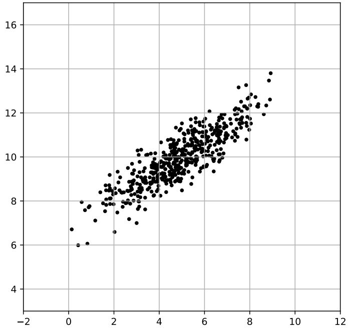
Goal: Extract column vectors of v from SVD

Let $\mathbf{X}$ be the centered feature matrix (previously) $\mathbf{X}$. Then

$$
\begin{aligned}
\mathbf{S} &= \mathbf{X}^\mathsf{T}\mathbf{X} = (\mathbf{U}\Sigma\mathbf{V}^\mathsf{T})^\mathsf{T}(\mathbf{U}\Sigma\mathbf{V}^\mathsf{T}) \\
&= \mathbf{V}\Sigma\mathbf{U}^\mathsf{T}\mathbf{U}\Sigma\mathbf{V}^\mathsf{T} \\
\mathbf{S} &= \mathbf{V}\Sigma^2\mathbf{V}^\mathsf{T} \\
\mathbf{S}\mathbf{V} &= \mathbf{V}\Sigma^2\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{V}\Sigma^2\mathbf{I} = \mathbf{V}\Sigma^2 \\
\mathbf{S}\left[\ \mathbf{v}_0 \mid \mathbf{v}_1 \mid \cdots \mid \mathbf{v}_{k-1}\ \right] &= \left[\ \mathbf{v}_0 \mid \mathbf{v}_1 \mid \cdots \mid \mathbf{v}_{k-1}\ \right]\begin{bmatrix} s_0^2 & 0 & \cdots & 0 \\ 0 & s_1^2 & & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & s_{k-1}^2 \end{bmatrix} \\
&= \left[\ s_0^2\mathbf{v}_0 \mid s_1^2\mathbf{v}_1 \mid \cdots \mid s_{k-1}^2\mathbf{v}_{k-1}\ \right] \\
\mathbf{S}\mathbf{v}_j &= s_j^2\mathbf{v}_j,\ j = 0, 1, \ldots, k-1
\end{aligned}
$$

PCA in Python: Generate Gaussian Cloud

```
from numpy.random import multivariate_normal as MVN
mu=np.array([5,10])
sigma=np.array([[3,2],[2, 1.7]])
X=MVN(mu, sigma, 500)
... plotting code
```



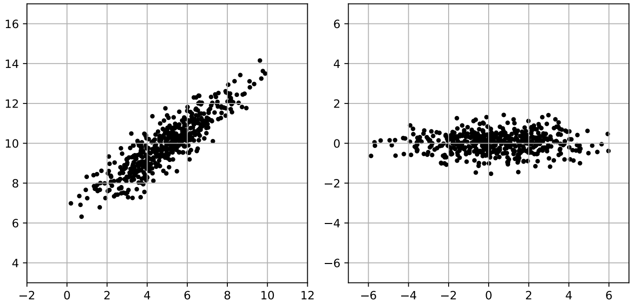
PCA on Gaussian Cloud

```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(X)

print("Components:")
print(pca.components_)
print("Variance Ratio:",pca.explained_variance_ratio_)
```

```
Components:
[[-0.81741707 -0.57604629]
 [ 0.57604629 -0.81741707]]
Variance Ratio: [0.94157304 0.05842696]
```

Transformation on Gaussian Cloud

```
P=pca.transform(X)
fig,ax=plt.subplots(nrows=1,ncols=2)
ax[0].scatter(X[:,0],X[:,1],marker=".",c="k")
ax[1].scatter(P[:,0],P[:,1],marker=".",c="k")
... other plot commands omitted
```



PCA on Auto-MPG Cylinder Data

Read and format data: 6 features, 3 classes

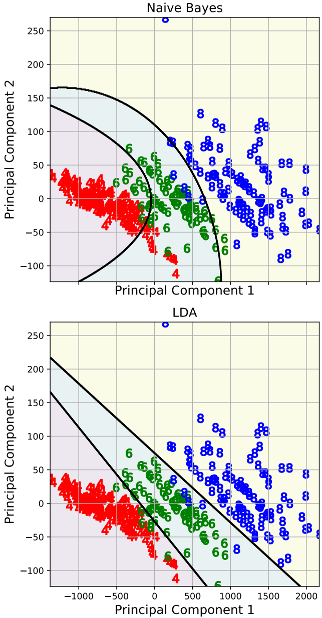
```
import pandas as pd
import numpy as np

data=pd.read_fwf("https://archive.ics.uci.edu/ml/
   machine-learning-databases/auto-mpg/auto-mpg.data",
   header=None,na_values="?")
data.columns=("mpg","cyl","displ","hp","weight",
   "accel","model","origin","carname")
data = data.dropna(axis=0)

cars=np.array(data[["cyl","mpg","displ","hp","weight",
    "accel"]])
cars=np.array([line for line in cars
    if line[0] in [4,6,8]])
Y=cars[:,0]/2-2
X=cars[:,1:]
```

Find first 2 Principal Components:

```
pca=PCA(n_components=2)
pca.fit(X)
comps=pca.components_
explain=pca.explained_variance_ratio_
for comp, frac in zip(comps,explain):
    print(round(100*frac,5),"percent:", comp)
```

```
99.76511 percent: [-0.0076786   0.11404841
    0.03920751  0.99267056 -0.00138875]
0.19845 percent: [-0.01797915  0.94308579
    0.30724002 -0.1206763  -0.03614883]
```

Of the five features, more than 99.95% is concentration in 2 dimensions.


Naive Bayes
KNN(4)
LDA
QDA
Principal Component 1
Principal Component 2
250
200
150
100
50
0
−50
−100
−1000
−500
0
500
1000
1500
2000

PCA for Image Compression

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

g=Image.open("BUILDING.JPG")
plt.imshow(g)
```

Convert to Gray Level

```
G=g.convert("L")
A=np.asarray(G)
plt.matshow(A,cmap="gray")
```

SVD on Image

Function **`PC(n)`** will truncate the expansion at **`n`**:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V} = s_1\mathbf{u}_1\mathbf{v}_1^\top + s_2\mathbf{u}_2\mathbf{v}_2^\top + \cdots + s_k\mathbf{u}_k\mathbf{v}_k^\top$$

```
U,Sigma,V=np.linalg.svd(A)
UT=U.T
def PC(n):
    M=np.zeros_like(A).astype(float)
    for j in range(n):
        B=np.outer(UT[j], V[j])*Sigma[j]
        M+=B
    return(M)
```

Results of several runs

5 Components

10 Components

20 Components

30 Components

40 Components

50 Components

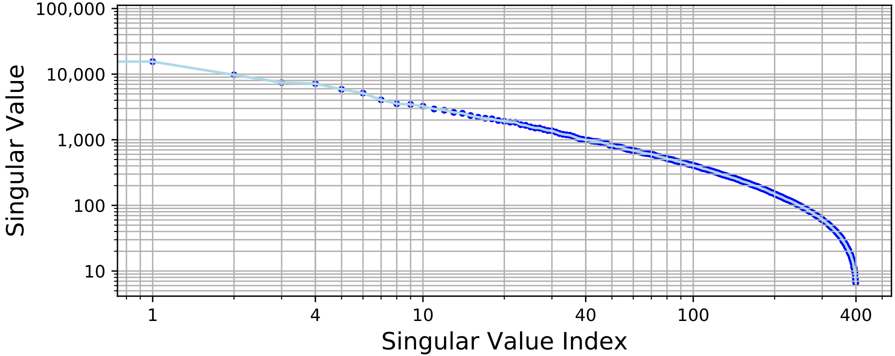
100 Components

200 Components

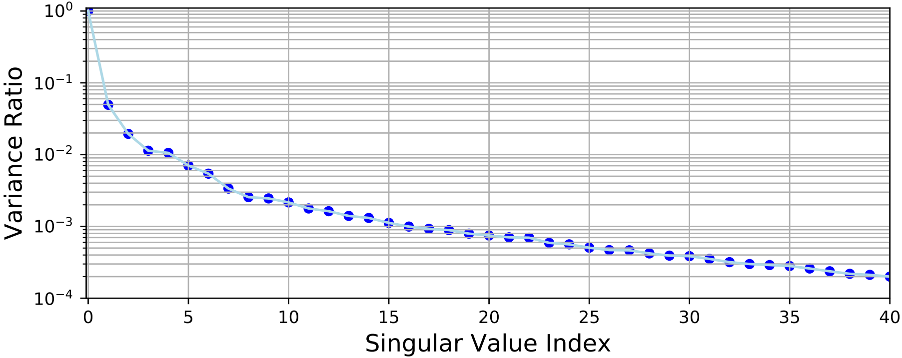
400 Components

Singular Values

Singular Values are square roots of eigenvalues of $\mathbf{S} = \mathbf{X}^T\mathbf{X}$



Relative Variance Ratio



$$r_i = \frac{s_i^2}{s_0^2}, \ \ s_i = \text{singular value}$$

The Curse of Dimensionality

- Hypersphere: $V(S_d) = \dfrac{\pi^{d/2} r^d}{\Gamma(1 + d/2)}$

The Curse of Dimensionality

- Hypersphere: $V(S_d) = \dfrac{\pi^{d/2} r^d}{\Gamma(1 + d/2)}$
- Spherical Shell: $\dfrac{V_{shell}}{V(S_d)} = \dfrac{K r^d - K(r - \epsilon)^d}{K r^d}$ where $K = \pi^{d/2}/\Gamma(1 + d/2)$ is a constant for any given d.

The Curse of Dimensionality

- Hypersphere: $V(S_d) = \dfrac{\pi^{d/2} r^d}{\Gamma(1 + d/2)}$
- Spherical Shell: $\dfrac{V_{shell}}{V(S_d)} = \dfrac{Kr^d - K(r-\epsilon)^d}{Kr^d}$ where $K = \pi^{d/2}/\Gamma(1 + d/2)$ is a constant for any given d.

$$\frac{V_{shell}}{V(S_d)} = \frac{r^d - (r-\epsilon)^d}{r^d} = 1 - \left(1 - \frac{\epsilon}{r}\right)^d \to 1 \text{ as } d \to \infty$$

Nearly all volume concentrated in shell.

The Curse of Dimensionality

- Hypersphere: $V(S_d) = \dfrac{\pi^{d/2} r^d}{\Gamma(1 + d/2)}$
- Spherical Shell: $\dfrac{V_{shell}}{V(S_d)} = \dfrac{Kr^d - K(r - \epsilon)^d}{Kr^d}$ where $K = \pi^{d/2}/\Gamma(1 + d/2)$ is a constant for any given d.

$$\frac{V_{shell}}{V(S_d)} = \frac{r^d - (r - \epsilon)^d}{r^d} = 1 - \left(1 - \frac{\epsilon}{r}\right)^d \to 1 \text{ as } d \to \infty$$

  Nearly all volume concentrated in shell.
- **Counter-intuitive consequence for data science: nearly all data concentrated on edges and corners of large data sets.**

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$
- Basis along axes, $\mathbf{e}_i = \underbrace{(\overbrace{0, 0, \ldots, 0}^{\text{all 0's}}, 1, \overbrace{0, \ldots, 0, 0}^{\text{all 0's}})}_{\text{length d}}$

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$
- Basis along axes, $\mathbf{e}_i = \underbrace{(\overbrace{0, 0, \ldots, 0}^{\text{all 0's}}, 1, \overbrace{0, \ldots, 0, 0}^{\text{all 0's}})}_{\text{length d}}$
- Main Diagonal, $\mathbf{1} = \overbrace{(1, 1, \ldots, 1)}^{\text{all 1's}}$

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$
- Basis along axes, $\mathbf{e}_i = \underbrace{(\overbrace{0, 0, \ldots, 0}^{\text{all 0's}}, 1, \overbrace{0, \ldots, 0, 0}^{\text{all 0's}})}_{\text{length d}}$
- Main Diagonal, $\mathbf{1} = \overbrace{(1, 1, \ldots, 1)}^{\text{all 1's}}$
- Dot Product, $\mathbf{e}_i \cdot \mathbf{1} = \|\mathbf{e}_i\| \|\mathbf{1}\| \cos\theta_i$ hence $\cos\theta_i = \dfrac{1}{\sqrt{d}}$

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$
- Basis along axes, $\mathbf{e}_i = \underbrace{(\overbrace{0, 0, \ldots, 0}^{\text{all 0's}}, 1, \overbrace{0, \ldots, 0, 0}^{\text{all 0's}})}_{\text{length d}}$
- Main Diagonal, $\mathbf{1} = \overbrace{(1, 1, \ldots, 1)}^{\text{all 1's}}$
- Dot Product, $\mathbf{e}_i \cdot \mathbf{1} = \|\mathbf{e}_i\| \|\mathbf{1}\| \cos\theta_i$ hence $\cos\theta_i = \dfrac{1}{\sqrt{d}}$
- As d becomes large $\theta \to \pi/2$

Curse of Dimensionality (2)

- Hypercube, corners at $\overbrace{(\pm 1, \pm 1, \cdots, \pm 1)}^{\text{length d}}$
- Basis along axes, $\mathbf{e}_i = \underbrace{(\overbrace{0, 0, \ldots, 0}^{\text{all 0's}}, 1, \overbrace{0, \ldots, 0, 0}^{\text{all 0's}})}_{\text{length d}}$
- Main Diagonal, $\mathbf{1} = \overbrace{(1, 1, \ldots, 1)}^{\text{all 1's}}$
- Dot Product, $\mathbf{e}_i \cdot \mathbf{1} = \|\mathbf{e}_i\| \|\mathbf{1}\| \cos\theta_i$ hence $\cos\theta_i = \dfrac{1}{\sqrt{d}}$
- As d becomes large $\theta \to \pi/2$
- Counter-intuitive consequence 2: **Diagonals all approach perpendicularity to principal axes. PCA projections may obscure clusters along diagonals.**

Curse of Dimensionality (3)

- Gaussian (Normal) random data has mean 0 and standard deviation 1 (when normalized).
- When the dimension becomes very large the probability mass moves away from the mean.
  - Centered at $\sqrt{d}$
  - Standard deviation $1\sqrt{2}$

References

1. MPG data from: Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository `http://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science.
2. Hanalei Village Photograph by the author, 26 May 2017.
3. Blum A, Hopcraft J, Kannan R (2018) *Foundations of Data Science* `https://www.cs.cornell.edu/jeh/book.pdf`, accessed 23 March 2019.