Dimension Reduction

Vahid Partovi Nia

Lecture 06

McGill

Outline

Suppose

$$\mathbf{x} = (x_1, \dots, x_p)^\top.$$

Dimensions reductuin is about projection of $\mathbf{x}$ onto a new space

$$\mathbf{z} = (z_1, \dots, z_m)^\top$$

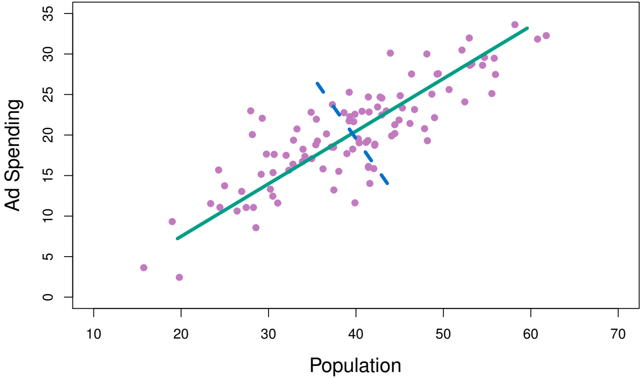
with a new dimension $m < p$.

If projection is linear

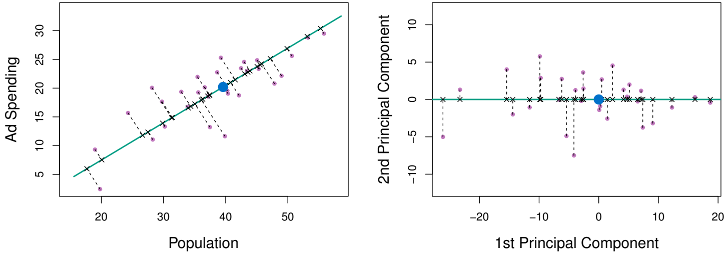
$$
\begin{aligned}
z_1 &= \sum_{j=1}^{p} \phi_{j1} x_j \\
z_2 &= \sum_{j=1}^{p} \phi_{j2} x_j \\
&\vdots \\
z_m &= \sum_{j=m}^{p} \phi_{jm} x_j
\end{aligned}
$$

How to find good coefficients ϕ_{jm}

What linear projection mean?



Data projection



How to find projection coefficients

Remember for a univariate random variable x

$$\begin{aligned} \mathrm{E}(\phi x) &= \phi \mathrm{E}(x) \\ \mathrm{V}(\phi x) &= \phi^2 \mathrm{V}(x) \end{aligned}$$

For a multivariate $\mathbf{x}, \mathrm{E}(\mathbf{x}) = \boldsymbol{\mu}, \mathrm{V}(\mathbf{x}) = \boldsymbol{\Sigma}$

$$\begin{aligned} \mathrm{E}(\boldsymbol{\phi}^\top \mathbf{x}) &= \boldsymbol{\phi}^\top \mathrm{E}(\mathbf{x}) = \phi \boldsymbol{\mu} \\ \mathrm{V}(\boldsymbol{\phi}^\top \mathbf{x}) &= \boldsymbol{\phi}^\top \mathrm{V}(\mathbf{x}) \boldsymbol{\phi} = \boldsymbol{\phi}^\top \boldsymbol{\Sigma} \boldsymbol{\phi} \end{aligned}$$

Principal components

- Find ϕ_1 so that $\mathrm{V}(\mathbf{z}_1) = \mathrm{V}(\boldsymbol{\phi}_1^\top \mathbf{x})$ is maximized.
- Find ϕ_2 so that $\phi_2 \perp \phi_1$ and $\mathrm{V}(\mathbf{z}_2) = \mathrm{V}(\boldsymbol{\phi}_2^\top \mathbf{x})$ is maximized.
- Find ϕ_3 so that $\phi_3 \perp \phi_1, \phi_2$ and $\mathrm{V}(\mathbf{z}_3) = \mathrm{V}(\boldsymbol{\phi}_3^\top \mathbf{x})$ is maximized.

$$\vdots$$

- Find ϕ_m so that $\phi_3 \perp \phi_1, \ldots, \phi_{m-1}$ and $\mathrm{V}(\mathbf{z}_m) = \mathrm{V}(\boldsymbol{\phi}_m^\top \mathbf{x})$ is maximized.

Find the first projection

Suppose $\mathrm{V}(\mathbf{x}) = \mathbf{\Sigma}$ has an eigen value decomposition $\mathbf{\Sigma} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^\top$

- $\mathbf{\Lambda} = \mathrm{diag}(\lambda_i)$
- $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$

It is easy to show that $\lambda_{\max} = \max \frac{\mathrm{V}(\boldsymbol{\phi}^\top\mathbf{x})}{\boldsymbol{\phi}^\top\boldsymbol{\phi}}$ and the maximizer is $\hat{\boldsymbol{\phi}} = \mathbf{e}_{\max}$

```
import pandas as pd
path='data/'
filename = path+'Auto.csv'
auto = pd.read_csv(filename,
        na_values=['?'], na_filter=True)
auto = auto.dropna()
```

```
import pandas as pd
path='data/'
filename = path+'Auto.csv'
auto = pd.read_csv(filename,
        na_values=['?'], na_filter=True)
auto = auto.dropna()


X = auto[['cylinders', 'displacement',
        'horsepower', 'weight',
        'acceleration']]

y = auto['mpg']
```

PCA

```
import pandas as pd
path='data/'
filename = path+'Auto.csv'
auto = pd.read_csv(filename,
        na_values=['?'], na_filter=True)
auto = auto.dropna()

X = auto[['cylinders', 'displacement',
        'horsepower', 'weight',
        'acceleration']]

y = auto['mpg']

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

PCA

```
import pandas as pd
path='data/'
filename = path+'Auto.csv'
auto = pd.read_csv(filename,
        na_values=['?'], na_filter=True)
auto = auto.dropna()

X = auto[['cylinders', 'displacement',
        'horsepower', 'weight',
        'acceleration']]

y = auto['mpg']

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X.values)
Z = pca.transform(X)

import matplotlib.pyplot as plt
%matplotlib inline
plt.scatter(Z[:,0], Z[:,1]);
```

Standardized PCA

```
from sklearn.preprocessing import scale
X_std = scale(X.values)
pca_std = PCA(n_components=2)
pca_std.fit(X_std)
Z_std = pca_std.transform(X_std)
plt.scatter(Z_std[:,0], Z_std[:,1]);
```

Principal component regression

After finding the coefficients, projecting x to the new dimension provides $\mathbf{Z}_{n\times m}$.
Now one can use the projected dimensions to predict a response variable y.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

$$\begin{aligned} \mathbf{Z}_{n\times m} &= \mathbf{X}_{n\times p}\boldsymbol{\Phi}_{p\times m} \\ \mathbf{y} &= \mathbf{Z}\boldsymbol{\theta} + \boldsymbol{\varepsilon} \end{aligned}$$

This is called principal component regression, $\mathbf{Z}$ involves no collinearity!

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
X_simple = X[['horsepower']]
lr.fit(X_simple,y)
lr.score(X_simple, y)
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
X_simple = X[['horsepower']]
lr.fit(X_simple,y)
lr.score(X_simple, y)


pcr = LinearRegression()
Z_simple = Z[:,0].reshape(-1,1)
pcr.fit(Z_simple, y)
pcr.score(Z_simple,y)
```

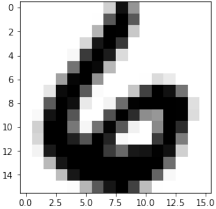
If regression with y is the reason of projection, it makes sense to project x to orthogonal axes, while correlation to y is accounted for.

- PCA: $\max \mathrm{V}(\boldsymbol{\phi}^\top \mathbf{x})$
- PLS: $\max \mathrm{V}(\boldsymbol{\phi}^\top \mathbf{x})\mathrm{cor}^2(\boldsymbol{\phi}\mathbf{x}, y)$

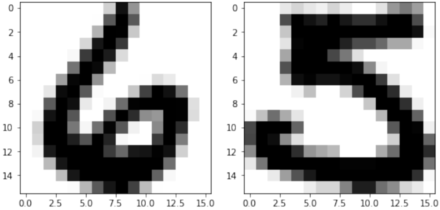
```
from sklearn.cross_decomposition import PLSRegression
pls = PLSRegression(n_components=2)
pls.fit(X, y)
W = pls.transform(X)
plt.scatter(W[:,0], W[:,1]);
```

```
from sklearn.cross_decomposition import PLSRegression
pls = PLSRegression(n_components=2)
pls.fit(X, y)
W = pls.transform(X)
plt.scatter(W[:,0], W[:,1]);


pls.score(X,y)
```

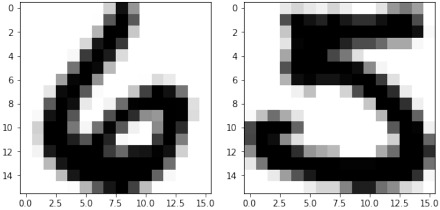
```
path='data/'
filename = path+'ziptrain.csv'
import numpy as np
zipdata = np.loadtxt(filename)

zipdata.shape


plt.imshow(-zipdata[0, 1:].reshape(16,16), "gray");
```



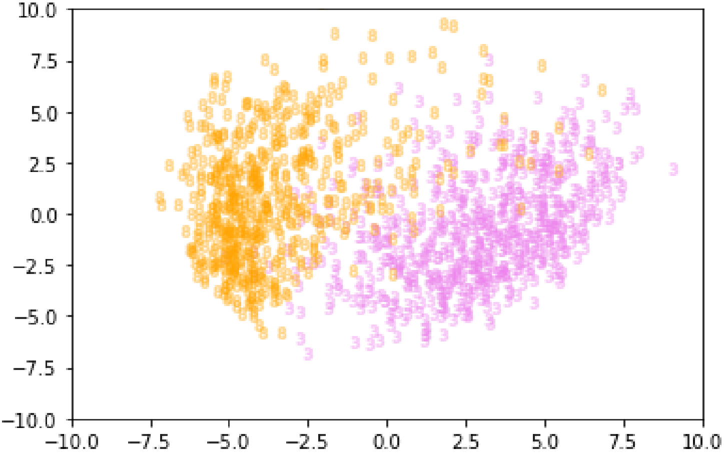
```
path='data/'
filename = path+'ziptrain.csv'
import numpy as np
zipdata = np.loadtxt(filename)

zipdata.shape


plt.imshow(-zipdata[0, 1:].reshape(16,16), "gray");
```



```
path='data/'
filename = path+'ziptrain.csv'
import numpy as np
zipdata = np.loadtxt(filename)

zipdata.shape


plt.imshow(-zipdata[0, 1:].reshape(16,16), "gray");
```




```
zipdata3=zipdata[zipdata[:, 0] == 3]
zipdata8=zipdata[zipdata[:, 0] == 8]
zipdata38 = np.vstack([zipdata3, zipdata8])

pca = PCA(n_components=2)
pca.fit(zipdata38[:, 1:])
Z = pca.transform(zipdata38[:,1:])
plt.scatter(Z[:,0], Z[:,1], c= zipdata38[:,0], alpha=0.3);
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0