Unsupervised Learning: Dimension Reduction

Industrial AI Lab.
Dimension Reduction

• Motivation:
  – Can we describe high-dimensional data in a “simpler” way?

→ Dimension reduction without losing too much information

→ Find a low-dimensional, yet useful representation of the data
Dimension Reduction

• Why dimensionality reduction?
  – insights into the low-dimensional structures in the data (visualization)
  – Fewer dimensions ⇒ Less chances of overfitting ⇒ Better generalization
  – **Speeding** up learning algorithms
    • Most algorithms scale badly with increasing data dimensionality
  – Less **storage** requirements (data compression)

  – Note: Dimensionality Reduction is different from Feature Selection
    • ... although the goals are kind of the same
  – Dimensionality reduction is more like “**Feature Extraction**”
    • Constructing a small set of new features from the original features
Highly Correlated Data

• How?
  idea: highly correlated data contains redundant features
Principal Component Analysis (PCA)

- Each example $x$ has 2 features $\{x_1, x_2\}$
- Consider ignoring the feature $x_2$ for each example
- Each 2-dimensional example $x$ now becomes 1-dimensional
  \[ x = \{x_1\} \]
- Are we losing much information by throwing away $x_2$?
  - **No.** Most of the data spread is along $x_1$ (very little variance along $x_2$)
Principal Component Analysis (PCA)

• Each example $x$ has 2 features $\{x_1, x_2\}$
• Consider ignoring the feature $x_2$ for each example
• Each 2-dimensional example $x$ now becomes 1-dimensional
  \[ x = \{x_1\} \]
• Are we losing much information by throwing away $x_2$?
• Yes, the data has substantial variance along both features (i.e., both axes)
Principal Component Analysis (PCA)

- Now consider a change of axes
- Each example $x$ has 2 features $\{u_1, u_2\}$
- Consider ignoring the feature $u_2$ for each example
- Each 2-dimensional example $x$ now become 1-dimensional
  $$x = \{u_1\}$$
- No. Most of the data spread is along $u_1$ (very little variance along $u_2$)
Principal Component Analysis (PCA)

• Data → projection onto unit vector $\hat{u}_1$
  – PCA is used when we want projections capturing maximum variance directions
  – Principal Components (PC): directions of maximum variability in the data
  – Roughly speaking, PCA does a change of axes that can represent the data in a succinct manner
Principal Component Analysis (PCA)

- HOW?
  1. Maximize variance (most separable)
  2. Minimize the sum-of-squares (minimum squared error)
PCA Algorithm: Pre-processing

• Given data

\[ x^{(i)} = \begin{bmatrix} x_{1}^{(i)} \\ \vdots \\ x_{n}^{(i)} \end{bmatrix}, \quad X = \begin{bmatrix} \cdots (x^{(1)})^T & \cdots \\ \cdots (x^{(2)})^T & \cdots \\ \vdots \\ \cdots (x^{(m)})^T & \cdots \end{bmatrix} \]

• Shifting (zero mean) and rescaling (unit variance)

1. Shift to zero mean

\[ \mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \]

\[ x^{(i)} \leftarrow x^{(i)} - \mu \quad \text{(zero mean)} \]

2. [optional] Rescaling (unit variance)

\[ \sigma_{j}^2 = \frac{1}{m-1} \sum_{i=1}^{m} m \left( x_{j}^{(i)} \right)^2 \]

\[ x_{j}^{(i)} \leftarrow \frac{x_{j}^{(i)}}{\sigma_{j}} \]
PCA Algorithm: Maximize Variance

- Find unit vector $u$ such that maximizes variance of projections

Note: $m \approx m - 1$ for big data

\[
\text{variance of projected data} = \frac{1}{m} \sum_{i=1}^{m} (u^T x^{(i)})^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)}^T u)^2 \\
= \frac{1}{m} \sum_{i=1}^{m} (x^{(i)}^T u)^T (x^{(i)}^T u) = \frac{1}{m} \sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u \\
= u^T \left( \frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)^T} \right) u \\
= u^T Su \quad (S = \frac{1}{m} X^T X : \text{sample covariance matrix})
\]
Maximize Variance

- In an optimization form

\[
\begin{align*}
\text{maximize} & \quad u^T S u \\
\text{subject to} & \quad u^T u = 1
\end{align*}
\]

\[
 u^T S u = u^T \lambda u = \lambda u^T u = \lambda \quad \text{(Eigen analysis : } S u = \lambda u) 
\]

\[
\Rightarrow \text{ pick the largest eigenvalue } \lambda_1 \text{ of covariance matrix } S \\
\Rightarrow u = u_1 \text{ is the } \lambda_1 \text{'s corresponding eigenvector} \\
\Rightarrow u_1 \text{ is the first principal component (direction of highest variance in the data)}
\]
Minimize the Sum-of-Squared Error

\[ e^{(i)} = x^{(i)} - u(x^{(i)}^T u) \]

\[
\|e^{(i)}\|^2 = \|x^{(i)}\|^2 - (x^{(i)}^T u)^2 = \|x^{(i)}\|^2 - (x^{(i)}^T u)^T (x^{(i)}^T u) = \|x^{(i)}\|^2 - u^T x^{(i)} x^{(i)}^T u
\]

\[
\frac{1}{m} \sum_{i=1}^{m} \|e^{(i)}\|^2 = \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)}\|^2 - \frac{1}{m} \sum_{i=1}^{m} u^T x^{(i)} x^{(i)}^T u
\]

\[
= \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)}\|^2 - u^T \left( \frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)}^T \right) u
\]
Minimize the Sum-of-Squared Error

• In an optimization form

\[
\text{minimize } \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)}\|^2 - u^T \left( \frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)T} \right) u \\
\text{constant given } x_i \underbrace{\text{maximize}}_{\text{maximize}}
\]

\[
\Rightarrow \text{maximize } u^T \left( \frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)T} \right) u = \max u^T S u
\]

\[
\therefore \text{minimize } \text{error}^2 = \text{maximize variance}
\]
Dimension Reduction Method \((n \rightarrow k)\)

1. Choose top \(k\) (orthonormal) eigenvectors, \(U = [u_1, u_2, \ldots, u_k]\)
2. Project \(x_i\) onto span \(\{u_1, u_2, \ldots, u_k\}\)

\[
    z^{(i)} = \begin{bmatrix}
        u_1^T x^{(i)} \\
        u_2^T x^{(i)} \\
        \vdots \\
        u_k^T x^{(i)}
    \end{bmatrix}
\]

or \(z = U^T x\)

- \(x^{(i)} \rightarrow\) projection onto unit vector \(u \Rightarrow u^T x^{(i)} = \text{distance from the origin along } u\)
Principal Component Analysis

• Data $\rightarrow$ projection onto unit vector $u$
Pictorial Summary of PCA
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

# data generation
m = 5000
mu = np.array([0, 0])
sigma = np.array([[3, 1.5],
                 [1.5, 1]])

X = np.random.multivariate_normal(mu, sigma, m)
X = np.asmatrix(X)

fig = plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'k.' )
plt.axis('equal')
plt.show()
```
```python
S = 1/(m-1)*X.T*X
S = np.asmatrix(S)

D, V = np.linalg.eig(S)

idx = np.argsort(-D)
D = D[idx]
V = V[:, idx]

print(D)
print(V)

[[ 3.78868797  0.19209281]
 [ 0.88056479  0.47392579]
 [ 0.47392579  0.88056479]]
```
```python
h = V[1,0]/V[0,0]
xp = np.arange(-6, 6, 0.1)
yp = h*xp

fig = plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'k. ')
plt.plot(xp, yp, 'r', linewidth=4.0)
plt.axis('equal')
plt.show()
```
Python Codes

\[ Z = X \times V[:,1] \]

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.hist(Z, 51)
plt.show()
```
PCA Example

- Multiple video camera records of spring and mass system
- Optimal data representation
  - Find the most informative point of view

source:
Multivariate Time Series

System order from
- Laws of physics or
- Data

Measured observations

Principal components
PCA Example

\[ x^{(i)} = \begin{bmatrix} x \text{ in camera 1} \\ y \text{ in camera 1} \\ x \text{ in camera 2} \\ y \text{ in camera 2} \\ x \text{ in camera 3} \\ y \text{ in camera 3} \end{bmatrix}, \quad X = \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ \vdots & (x^{(1)}) & \cdots & (x^{(m)}) \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \]

```python
from six.moves import cPickle

X = cPickle.load(open('./data_files/pca_spring.pkl', 'rb'))
X = np.asmatrix(X.T)

print(X.shape)
m = X.shape[0]

(273, 6)``,

24
PCA Example

```python
plt.figure(figsize=(15, 7))
plt.subplot(131)
plt.plot(X[:, 0], -X[:, 1], 'r')
plt.axis('equal')
plt.title('Camera 1')

plt.subplot(132)
plt.plot(X[:, 2], -X[:, 3], 'b')
plt.axis('equal')
plt.title('Camera 2')

plt.subplot(133)
plt.plot(X[:, 4], -X[:, 5], 'k')
plt.axis('equal')
plt.title('Camera 3')

plt.show()
```
PCA Example

```python
X = X - np.mean(X, axis=0)
S = 1/(m-1)*X.T*X
S = np.asmatrix(S)
D, V = np.linalg.eig(S)
idx = np.argsort(-D)
D = D[idx]
V = V[:, idx]
print(D)
print(V)
```

```
[ 2.46033089e+04  3.22747042e+02  8.73851124e+01  8.19527660e+01
  3.19467195e+01  7.42861585e+00]
[[ 0.36881064  0.62298194 -0.12571821 -0.42647348  0.52121775 -0.0807439
  0.35632379  0.57286174  0.132303  0.59881765 -0.40143215  0.08734045]
 [ 0.58419477 -0.22610057 -0.20325551 -0.47751523 -0.58153918  0.00857804]
 [ 0.08652315 -0.02671281  0.75692234 -0.14177391 -0.06010869 -0.62861422]
 [ 0.4159798 -0.29900638  0.49374948  0.05637591  0.32442517  0.62075559]
 [-0.46389987  0.37746931  0.32963322 -0.45633202 -0.34660023  0.45308403]]
```
PCA Example

```python
plt.figure(figsize=(10,6))
plt.stem(np.sqrt(D))
plt.show()
```
PCA Example

# relative magnitudes of the principal components

```python
z = X*V
xp = np.arange(0,m)/24

plt.figure(figsize=(10, 6))
plt.plot(xp, Z)
plt.yticks([])
plt.show()
```
PCA Example

```python
## projected onto the first principal component
# 6 dim -> 1 dim (dim reduction)
# relative magnitude of the first principal component

z = X*V[:,0]

plt.figure(figsize=(10, 6))
plt.plot(z)
plt.yticks([])
plt.show()
```
PCA Example