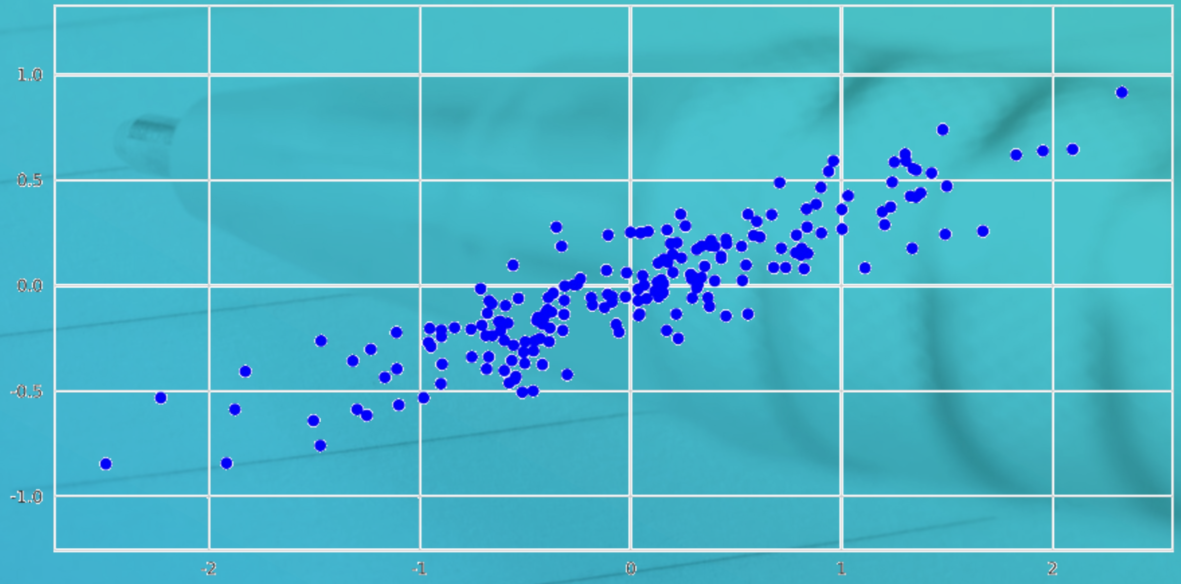
URBAN GEOGRAPHIC INFORMATION SYSTEM

Python – Statistics I

Chun-Hsiang Chan
Department of Geography,
National Taiwan Normal University

Outline

- PCA in Mathematics
- PCA in Python



Review

- Before we explain PCA, we need to review the mathematical meaning of three basic descriptive statistics, including expectation, variance, and covariance.
- In previous courses or your understanding, these three parameters usually perform as above equations.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

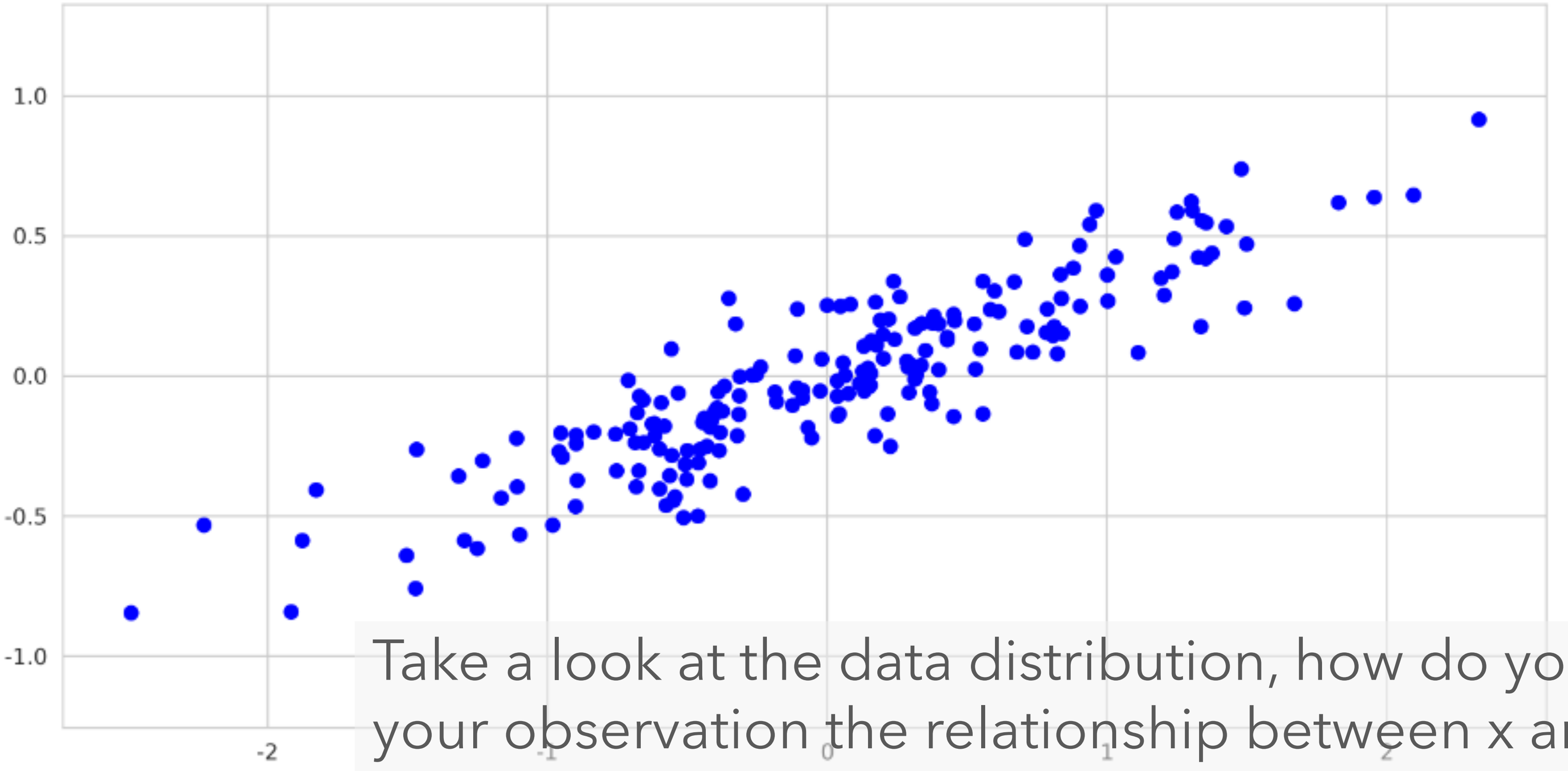
$$var(x) = \sigma^2 = \left(\sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$Cov(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

Review

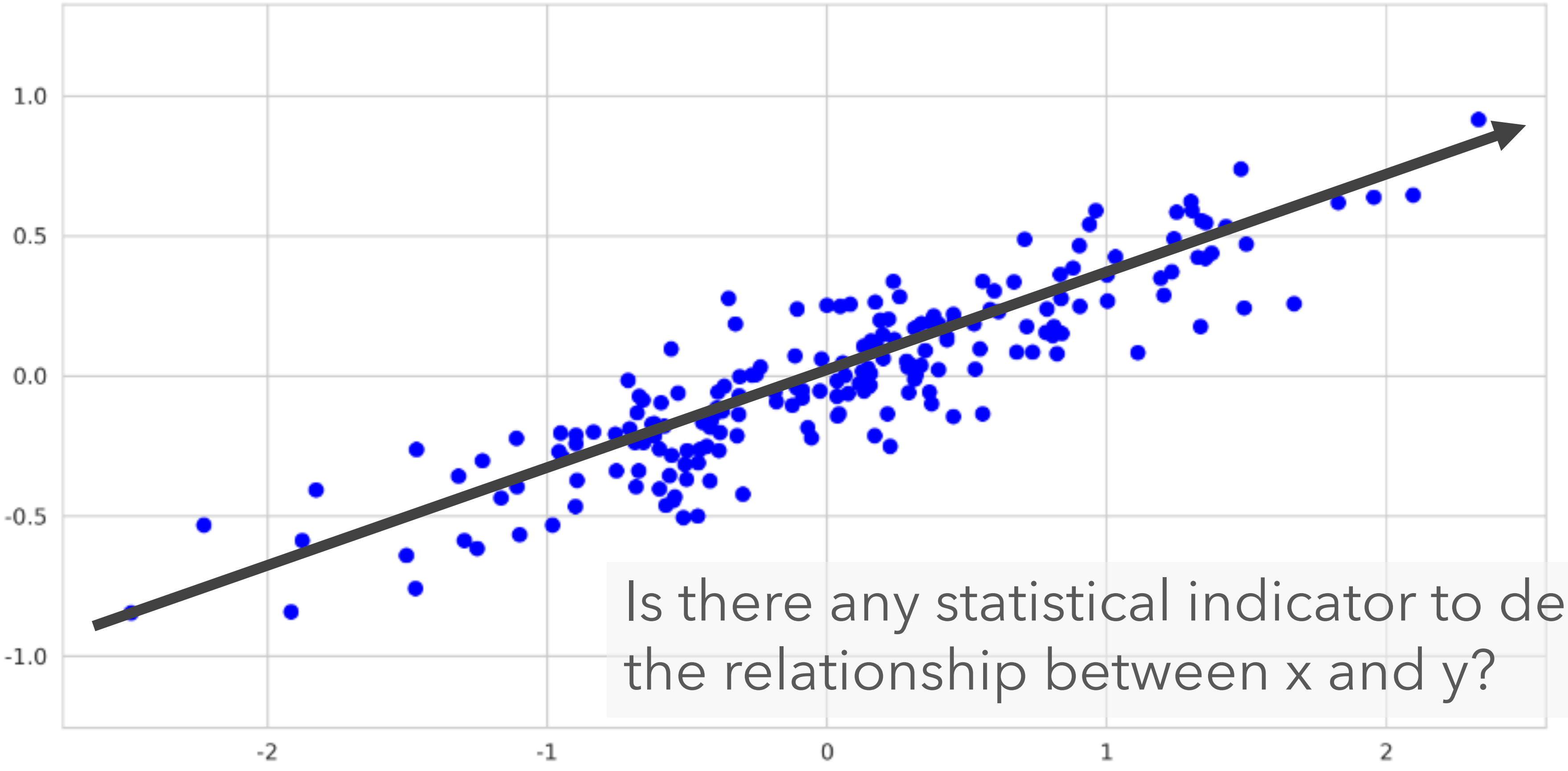
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \mid var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$



Take a look at the data distribution, how do you explain your observation the relationship between x and y?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid \ var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \ \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$



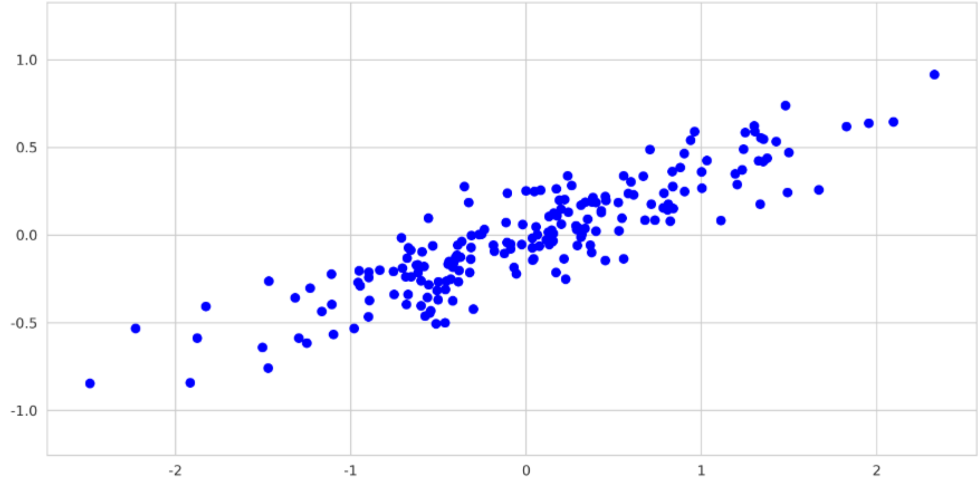
Is there any statistical indicator to describe the relationship between x and y?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \;\mid\; var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \;\mid\; Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

- Pearson‘s correlation coefficient
- Given two parameters x_i and y_i, where i ranges from 1 to n.
  Then, Pearson’s correlation coefficient could be defined as follows.

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2 \sum_{i=1}^{n}(y_i - \mu_y)^2}}$$



Question 1

If x is highly correlated with y, and then what do you expect from their covariance and standard deviations?

Review

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

- **Expectation**

$$E(X) = \sum_x xP(X = x) = \mu$$

- **Variance**

$$\begin{aligned} var(X) &= E([X-\mu]^2) \\ &= E(X^2 - 2\mu X + \mu^2) \\ &= E(X^2) - 2\mu E(X) + \mu^2 \\ &= E(X^2) - 2\mu^2 + \mu^2 \\ &= E(X^2) - \mu^2 \\ &= E(X^2) - E(X)^2 \end{aligned}$$

Characteristics of Expectation

$E(aX + bY) = aE(X) + bE(Y)$, a, b $\in \mathbb{R}$
X and Y can be independent or dependent.

$E(XY) = E(X)E(Y)$
Where $cov(X,Y) = 0$

Review

$E(aX + bY) = aE(X) + bE(Y), \mathrm{a}, \mathrm{b} \in \mathbb{R}$
X and Y can be independent or dependent.
$E(XY) = E(X)E(Y)$
Where $cov(X, Y) = 0$

- Covariance
- If x and y are independent…

$$var(X + Y) = var(X) + var(Y)$$

- If x and y are dependent…

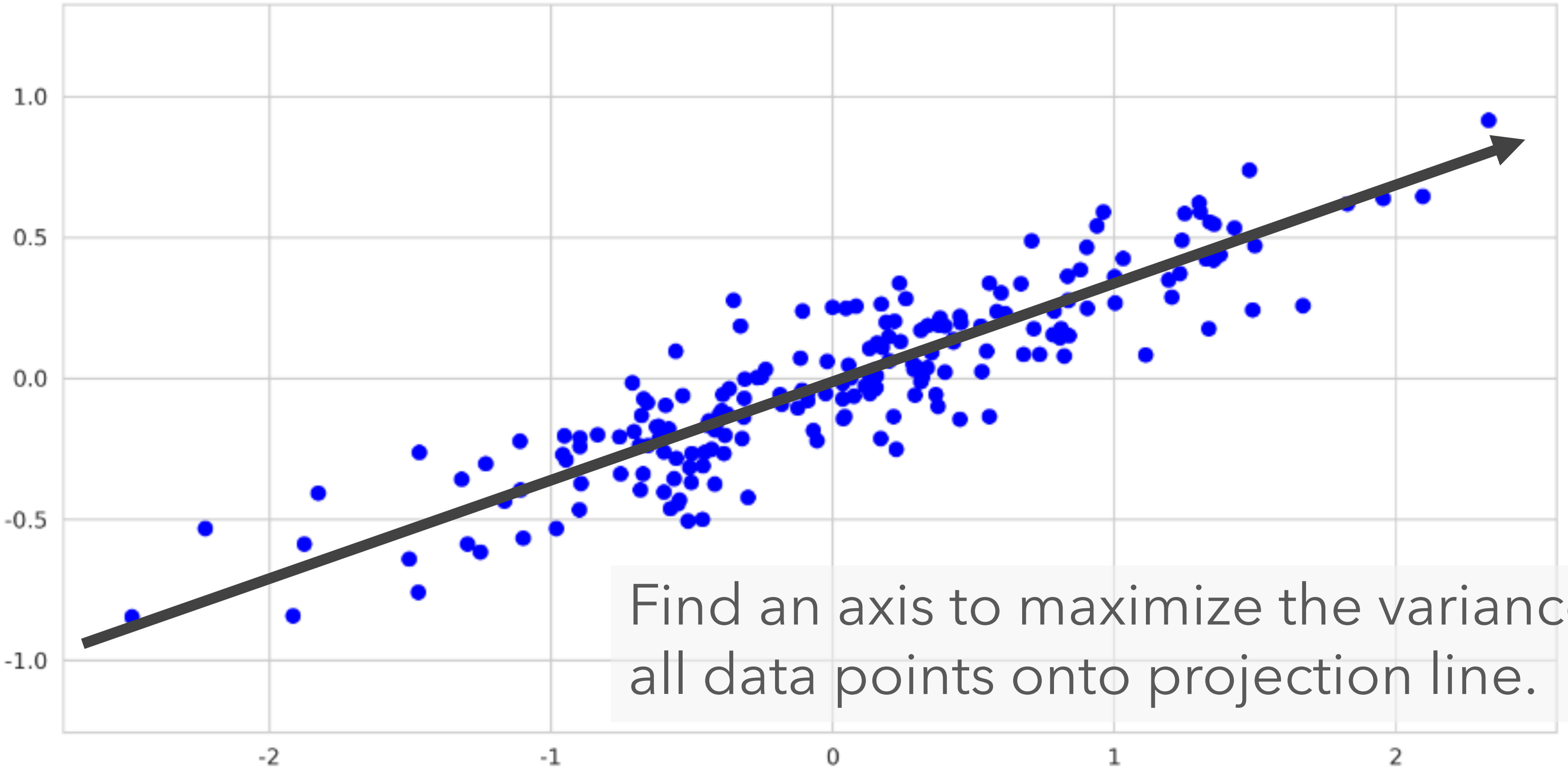
$$var(X + Y) = E([(X + Y) - E(X + Y)]^2)$$
$$= E\left(\left[(X + Y) - \left(E(X) + E(Y)\right)\right]^2\right)$$
$$= E\left(\left[\left(X - E(X)\right) + \left(Y - E(Y)\right)\right]^2\right)$$
$$= E\left(\left(X - E(X)\right)^2 + 2\left(X - E(X)\right)\left(Y - E(Y)\right) + \left(Y - E(Y)\right)^2\right)$$
$$= E\left[\left(X - E(X)\right)^2\right] + E\left[\left(Y - E(Y)\right)^2\right] + 2E\left[\left(X - E(X)\right)\left(Y - E(Y)\right)\right]$$
$$= var(X) + var(Y) + 2cov(X, Y)$$

Why Do We Need Dimension Reduction?

- Here comes the first question into your mind.
  - Why do we need dimension reduction?
  - What's the importance of dimension reduction?
  - Can we directly import all datasets into your model without dimension reduction?
- Statistical models (e.g., linear regression) have several assumptions when you adopt them. One of them is "all variables have to be linearly independent," indicating no collinearity.
- To achieve this goal, various methods were developed for orthogonalizing parameters and reducing the dimension of the dataset, such as PCA, LDA, LLE, and Laplacian Eigenmaps.

PCA – Math

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ \mid \ var(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \ \mid Cov(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

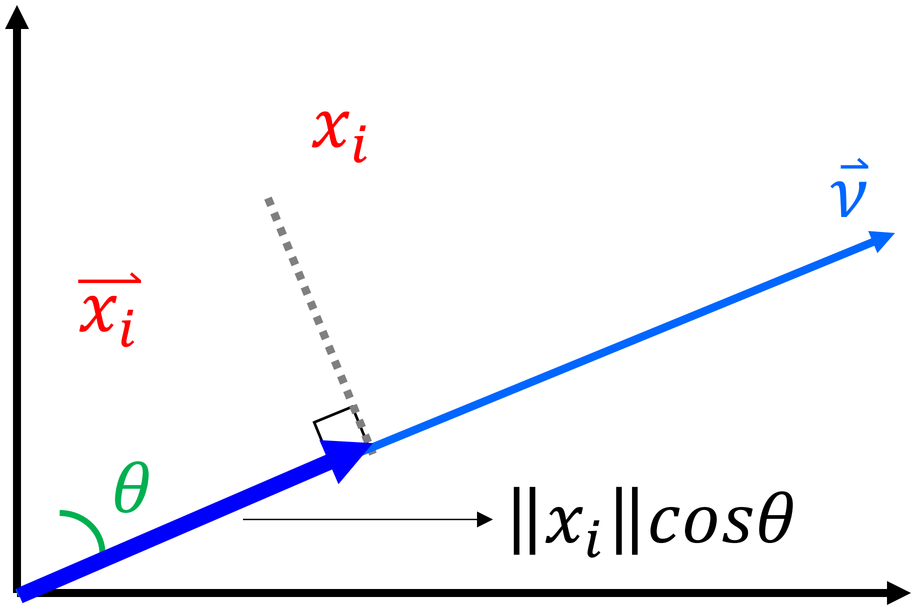


Find an axis to maximize the variance of all data points onto projection line.

PCA – Math

$$\cos\theta = \frac{{x_i}^T \cdot v}{\|x_i\|\|v\|}$$

- Given a point "x" and project onto a vector "v".



$$\|x_i\|cos\theta = \|x_i\| \frac{{x_i}^T \cdot v}{\|x_i\|\|v\|} = \frac{{x_i}^T \cdot v}{\|v\|}$$

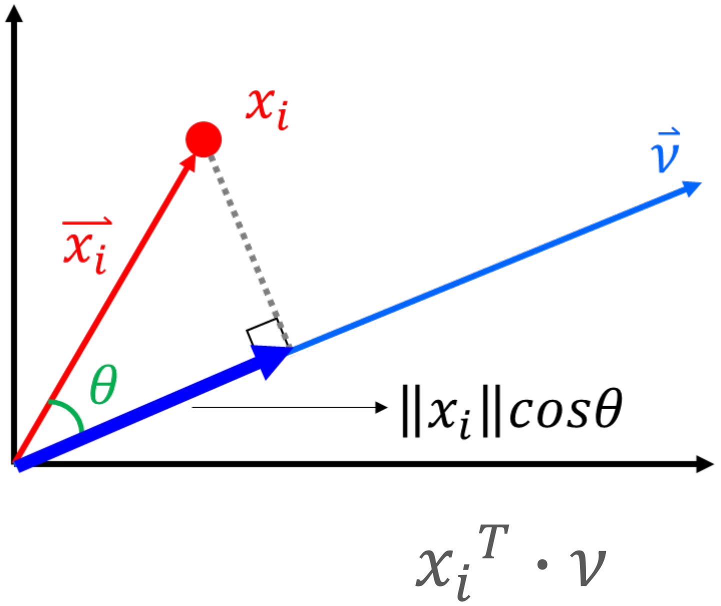
$$if\ v\ is\ unit\ vector \ldots\ \|v\| = 1$$

$$= \frac{{x_i}^T \cdot v}{\|v\|} = {x_i}^T \cdot v$$

PCA – Math

$$X = \begin{bmatrix} | & | & \cdots & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \rightarrow X^T = \begin{bmatrix} - & x_1 & - \\ - & x_2 & - \\ - & \vdots & - \\ - & x_n & - \end{bmatrix}$$

$$P = \begin{bmatrix} {x_1}^T \\ {x_2}^T \\ \vdots \\ {x_n}^T \end{bmatrix} = X^T u \Rightarrow solve\ P$$



PCA – Math

$$J(u) = \|P^2\| = P^T P = (X^T u)^T (X^T u) = u^T X X^T u$$

$$\underset{u}{\text{argmax}}\ J(u) = u^T X X^T u, where\ subject\ to\ u^T u = 1$$

Add Lagrange multiplier

$$\underset{u,\lambda}{\text{argmax}}\ J(u, \lambda) = u^T X X^T u + \lambda(1 - u^T u)$$

$$\nabla_u J(u, \lambda) = \nabla_u \left(u^T X X^T u + \lambda(1 - u^T u)\right) = 0$$

$$\Rightarrow 2XX^T u - 2\lambda u = 0$$

$$\Rightarrow XX^T u = \lambda u$$

XX^T → $cov(X)$

λ → eigenvalue

u → eigenvector → $Au = \lambda u$

PCA – Math

$$XX^T u = \lambda u$$

Eigenvector is a symmetry matrix
u is an unit vector

$$uu^T = u^T u = 1$$

$when\ u\ is\ the\ eigen\ vector$

$$J(u) = \|P^2\| = u^T XX^T u = u^T \lambda u = \lambda u^T u = \lambda$$

Given an ***eigenvector****, the* ***total square of projected values*** *is the* ***eigenvalue*** = λ

PCA – Math

$XX^Tu = \lambda u$

A is a square symmetric matrix has orthogonal eigenvectors with different eigenvalues.

$[x_1, \lambda_1]$, $[x_2, \lambda_2]$

$$\begin{cases} Ax_1 = \lambda_1 x_1 \\ Ax_2 = \lambda_2 x_2 \end{cases}$$

$$x_1{}^T A x_2 = x_1{}^T \lambda_2 x_2 = \lambda_2 x_1{}^T x_2$$

Equal

$$x_1{}^T A^T x_2 = (Ax_1)^T x_2 = (\lambda_1 x_1)^T x_2 = \lambda_1 x_1{}^T x_2$$

$$(\because A \in symmetric\ matrix, \therefore A = A^T)$$

Eigenvector is a symmetry matrix

u is an unit vector

$$uu^T = u^T u = 1$$

$$\lambda_2 x_1{}^T x_2 = \lambda_1 x_1{}^T x_2$$

$$x_1{}^T x_2 (\lambda_2 - \lambda_1) = 0$$

Orthogonal $x_1{}^T x_2 = 0$

All eigenvalues are different

PCA – Math

- Conversion between orthogonal bases

$$u_i \cdot u_j = u_i^T \cdot u_j = \begin{cases} 1, if\ i = j \\ 0, otherwise \end{cases}$$

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_d \\ | & | & & | \end{bmatrix} \Rightarrow U^T U = I = U^{-1} = U^T$$

$$x = y_1 u_1 + y_2 u_2 + \cdots + y_d u_d = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_d \\ | & | & & | \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} = Uy$$

$$\Rightarrow y = U^{-1} x = U^T x$$

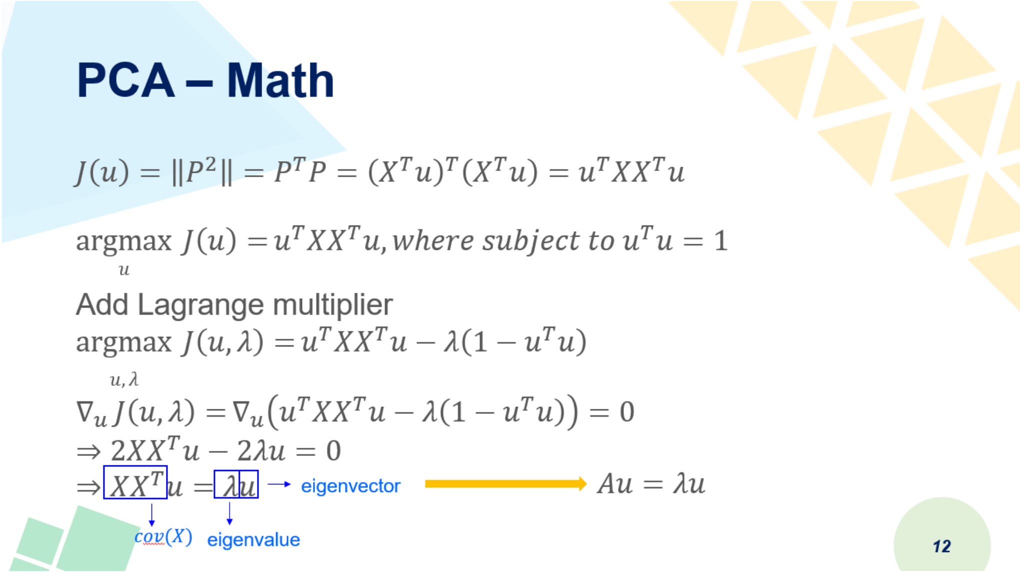
PCA – Math

$XX^T v = \lambda v \Rightarrow AV = \lambda V$

where V is eigenvector and λ is eigenva

Principal component PC

$AV = \lambda V$



PCA – Math

From Singular Vector Decomposition (SVD)

$$A = U\Sigma V^T \Rightarrow AV = U\Sigma \Rightarrow \lambda = \frac{\Sigma^2}{N}$$

U are the principal components scaled to unit norm
Σ is a diagonal matrix with singular values
V contains principal axes

$$cov(A, A) = \frac{1}{N} A^T A = V \frac{\Sigma^2}{N} V^T = VEV^T$$

V is the eigenvector of the covariance matrix
$E = \frac{\Sigma^2}{N}$ are the eigenvalues of covariance matrix

PCA – Math

$$A = U\Sigma V^T;\ E = \frac{\Sigma^2}{N}$$

Variable loading L (A onto PC)

To compute the variable loading matrix, we need to compute the cross-covariance matrix between original variable and principal components.

$$cov(A, PC) = \frac{A^T PC}{N}$$

where A is the original variables and PC is the standardized principal components ($Standardize\ PC = \sqrt{N}U$).

$$cov(A, PC) = \frac{A^T\sqrt{N}U}{N} = \frac{V\Sigma U^T U}{\sqrt{N}} = V\frac{\Sigma}{\sqrt{N}} = V\sqrt{E} = L$$

PCA – Math

The steps of PCA

1. Find the sample mean $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
2. Subtract mean
3. Compute covariance matrix $C = \frac{1}{n}XX^T = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)\,(x_i - \mu)$
4. Find the eigenvalues of C and arrange them into descending order

   $\lambda_1 > \lambda_2 > \cdots > \lambda_d, \{u_1, u_2, \ldots, u_d\}$
5. The transformation is $y = U^T X$.

PCA – Python

scikit learn | Install | User Guide | API | Examples | Community | More | Go

Prev | Up | Next

scikit-learn 1.1.2
Other versions

Please **cite us** if you use the software.

sklearn.decomposition.PCA
Examples using sklearn.decomposition.PCA

sklearn.decomposition.PCA

class `sklearn.decomposition.`**`PCA`**(*n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None*) [source]

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

Notice that this class does not support sparse input. See `TruncatedSVD` for an alternative with sparse data.

Read more in the User Guide.

Parameters::

n_components : *int, float or 'mle', default=None*

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'` and `svd_solver == 'full'`, Minka's MLE is used to guess the dimension. Use of `n_components == 'mle'` will interpret `svd_solver == 'auto'` as `svd_solver == 'full'`.

If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

If `svd_solver == 'arpack'`, the number of components must be strictly less than the minimum of n_features and n_samples.

PCA – Python

```
# linear algebra
from sklearn.decomposition import PCA
# set experiment samples
rn_state = np.random.RandomState(1)
X = np.dot(rn_state.rand(2, 2),rn_state.randn(2, 200)).T
# original data distribution
plt.figure(figsize=[12,6],dpi=300)
plt.scatter(X[:, 0], X[:, 1], c='b')
plt.axis('equal')
plt.show()
```

PCA – Python

```
# call PCA
pca = PCA(n_components=2) # number of preserved components
pca.fit(X)
# show results
print(pca.mean_)
print(pca.explained_variance_)
print(pca.components_)
```

PCA – Python

```
# plot results
plt.figure(figsize=[12,6],dpi=300)
arrowprops = dict(arrowstyle='->', linewidth=2, color='r')
plt.scatter(X[:, 0], X[:, 1], c=[[0,0,1,0.1]])
plt.annotate('', pca.mean_ +
pca.components_[0]*2*np.sqrt(pca.explained_variance_[0]),
             pca.mean_, arrowprops=dict(arrowstyle='->', linewidth=2, color='g'))
plt.annotate('', pca.mean_ +
pca.components_[1]*2*np.sqrt(pca.explained_variance_[1]),
             pca.mean_, arrowprops=dict(arrowstyle='->', linewidth=2, color='g'))
plt.axis('equal')
plt.show()
```

PCA – Python

```
# project to PC1
pca1 = PCA(n_components=1)
pca1.fit(X)
X_pca = pca1.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)

# plot projected data
X_new = pca1.inverse_transform(X_pca)
plt.figure(figsize=[12,6],dpi=300)
plt.scatter(X[:, 0], X[:, 1], alpha=0.3)
plt.scatter(X_new[:, 0], X_new[:, 1], c='r', alpha=0.2)
plt.axis('equal')
plt.show()
```

PCA – Python

```
# variable loading calculation
variable_loading = pca.components_.T * np.sqrt(pca.explained_variance_)
# plot variable loading
plt.subplots(figsize=[12,6], dpi=300)
plt.subplot(121)
plt.title('PC1', fontsize=18)
plt.bar(np.arange(2), variable_loading[0])
plt.plot(np.arange(-0.5,2.0,0.5),np.zeros(5),'k')
plt.xticks(np.arange(2),['X1', 'X2'], fontsize=14)
plt.ylabel('Variable Loadings', fontsize=16)
plt.subplot(122)
plt.bar(np.arange(2), variable_loading[1])
plt.plot(np.arange(-0.5,2.0,0.5),np.zeros(5),'k')
plt.title('PC2', fontsize=18)
plt.xticks(np.arange(2),['X1', 'X2'], fontsize=14)
plt.ylabel('Variable Loadings', fontsize=16)
plt.show()
```

PCA – Python

```
# project to PC1
pca1 = PCA(n_components=1)
pca1.fit(X)
X_pca = pca1.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
# plot projected data
X_new = pca1.inverse_transform(X_pca)
plt.figure(figsize=[12,6], dpi=300)
plt.scatter(X[:, 0], X[:, 1], alpha=0.3)
plt.scatter(X_new[:, 0], X_new[:, 1], c='r', alpha=0.2)
plt.axis('equal')
plt.show()
```

Question Time

- **Assignment:**
- **Download today's lab practice and upload to moodle.**
- **Thx**

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw
Web: toodou.github.io

SCAN ME