建立模型 - Python (1)

David Chiu

課程資料

- 所有課程補充資料、投影片皆位於
  - https://github.com/ywchiu/ctbcpy

機器學習

機器學習

- 機器學習的目的是：歸納（Induction）
  從詳細事實到一般通論

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E

-- Tom Mitchell (1998)

- 找出有效的預測模型
  一開始都從一個簡單的模型開始
  藉由不斷餵入訓練資料，修改模型
  不斷提升預測績效

機器學習步驟

使用者行為

機器學習問題分類

- 監督式學習 (Supervised Learning)
  - 回歸分析 (Regression)
  - 分類問題 (Classification)

- 非監督式學習 (Unsupervised Learning)
  - 降低維度 (Dimension Reduction)
  - 分群問題 (Clustering)

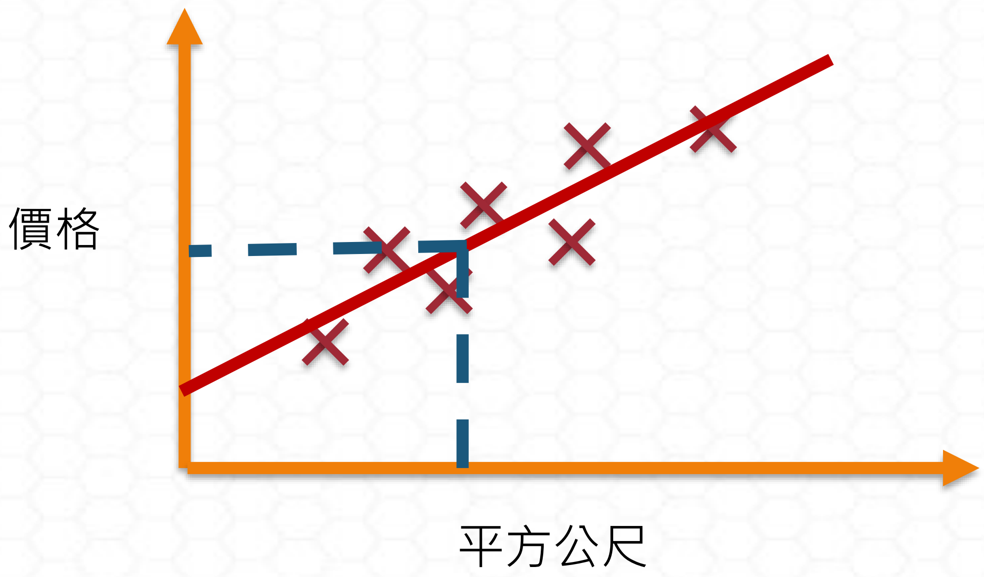
使用監督式學習進行預測

- 迴歸分析
  - 使用一組**已知對應值**的資料產生的模型，預測新資料的對應值
  - e.g. 股價預測

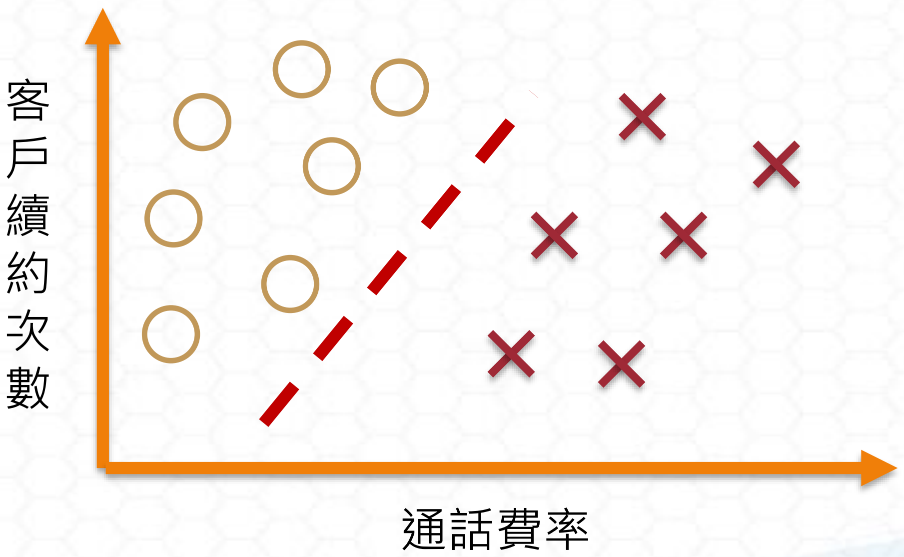
- 分類問題
  - 根據已知標籤的訓練資料集(Training Set)，產生一個新模型，用以預測測試資料集(Testing Set)的標籤。
  - e.g. 客戶流失分析

使用監督式學習進行預測

迴歸分析



分類問題

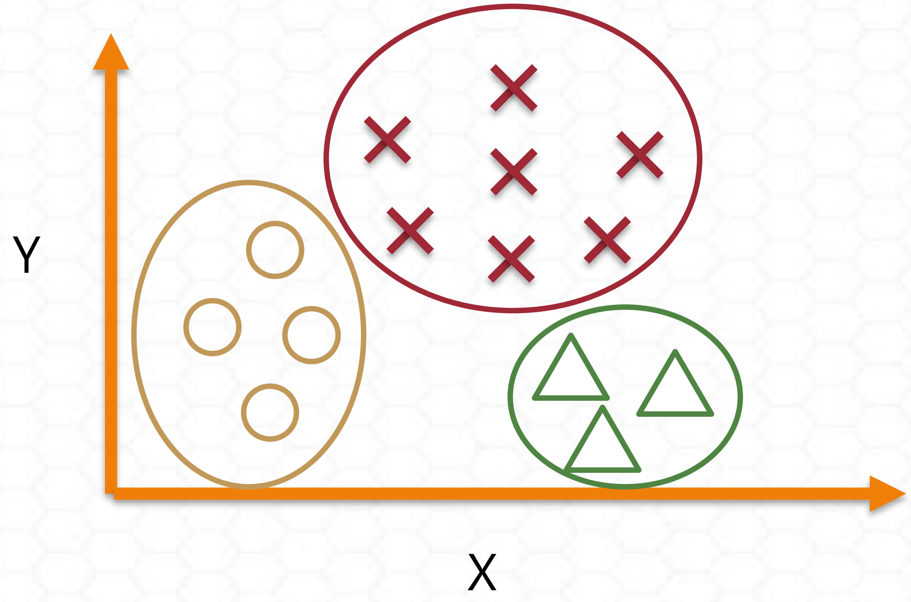


使用非監督式學習找出隱藏的架構

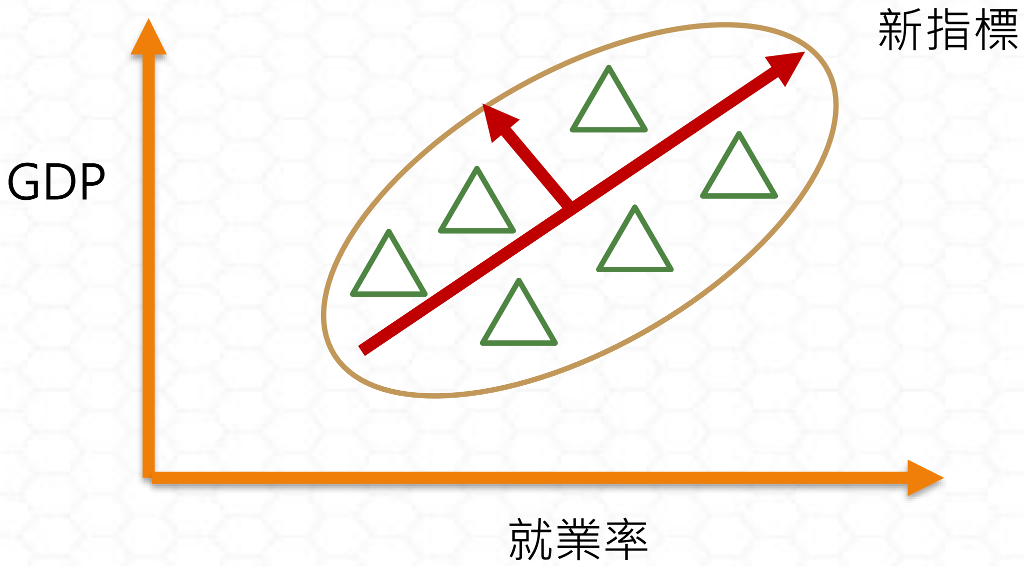
- 降低維度
  - 產生一有最大變異數的欄位線性組合，可用來降低原本問題的維度與複雜度
  - e.g. 濃縮用到的特徵，編纂成一個新指標
- 分群問題
  - 物以類聚 (近朱者赤、近墨者黑)
  - e.g.將客戶分層

使用非監督式學習找出隱藏的架構

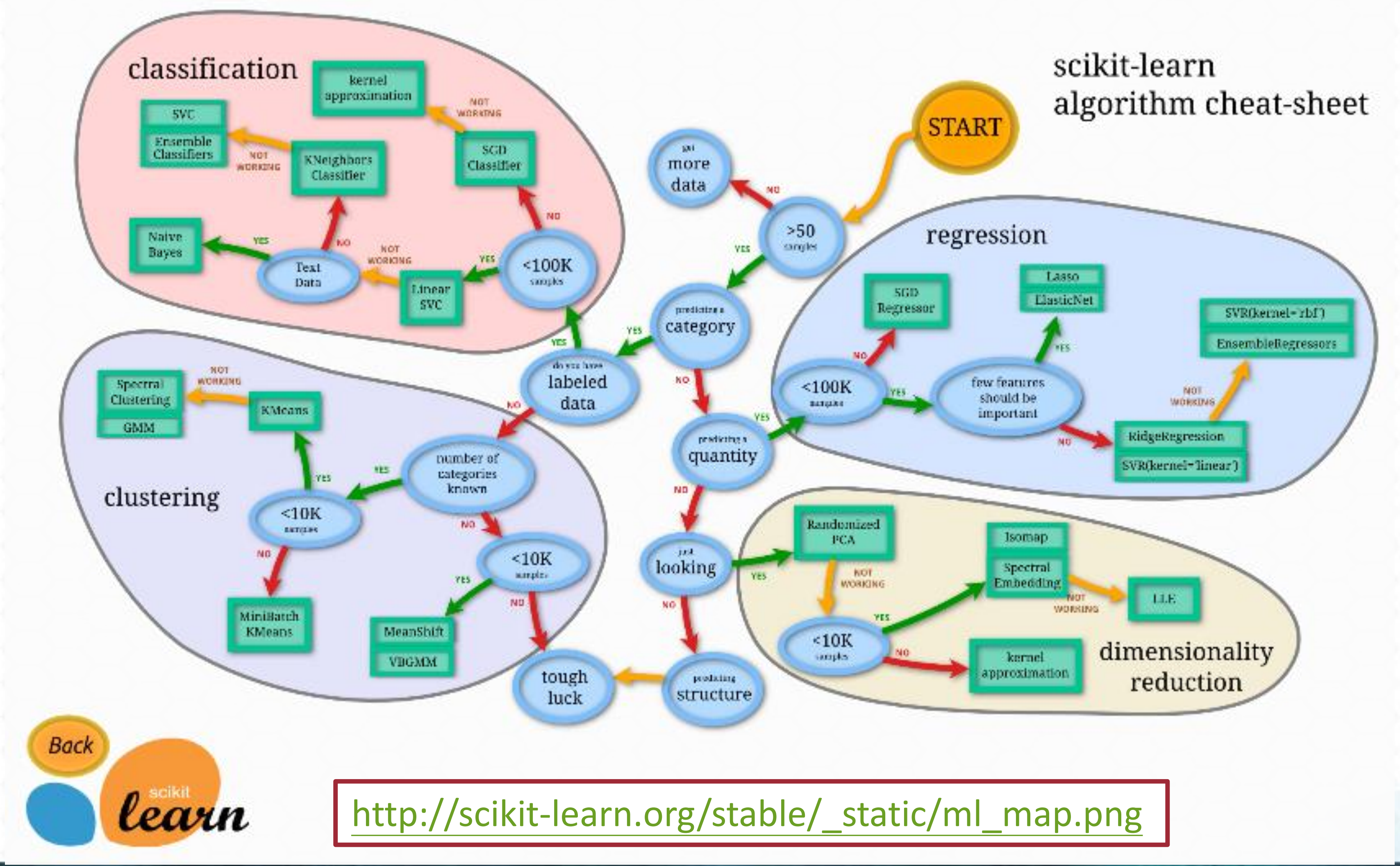
分群問題



降低維度



機器學習地圖



http://scikit-learn.org/stable/_static/ml_map.png

非監督式學習

機器學習問題分類

- 監督式學習 (Supervised Learning)
  - 回歸分析 (Regression)
  - 分類問題 (Classification)

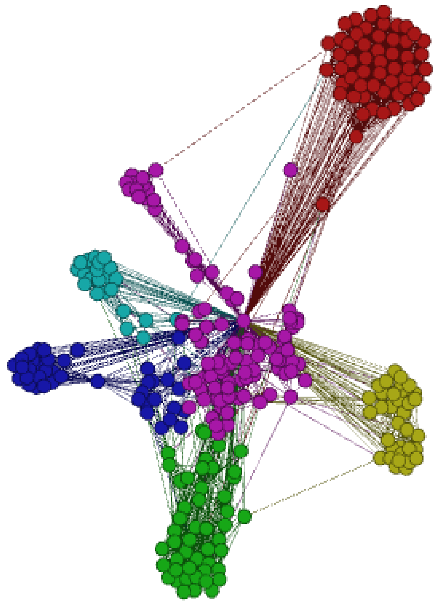
- 非監督式學習　(Unsupervised Learning)
  - 降低維度 (Dimension Reduction)
  - 分群問題 (Clustering)

非監督式學習

- 降低維度
  - 產生一有最大變異數的欄位線性組合，可用來降低原本問題的維度與複雜度
  - e.g. 濃縮用到的特徵，編纂成一個新指標
- 分群問題
  - 物以類聚 (近朱者赤、近墨者黑)
  - e.g.將客戶分層

分群應用

- 市場分析
  - 將客戶依行為跟特徵做不同區隔
  - 產品定位
  - 區分市場
- 產品搭配銷售
  - 將同類型的產品組合成紅綠標組合
- 社會網路分析
  - 找出相似的朋友群
- 搜尋結果分組
  - 找出類似文章或主題

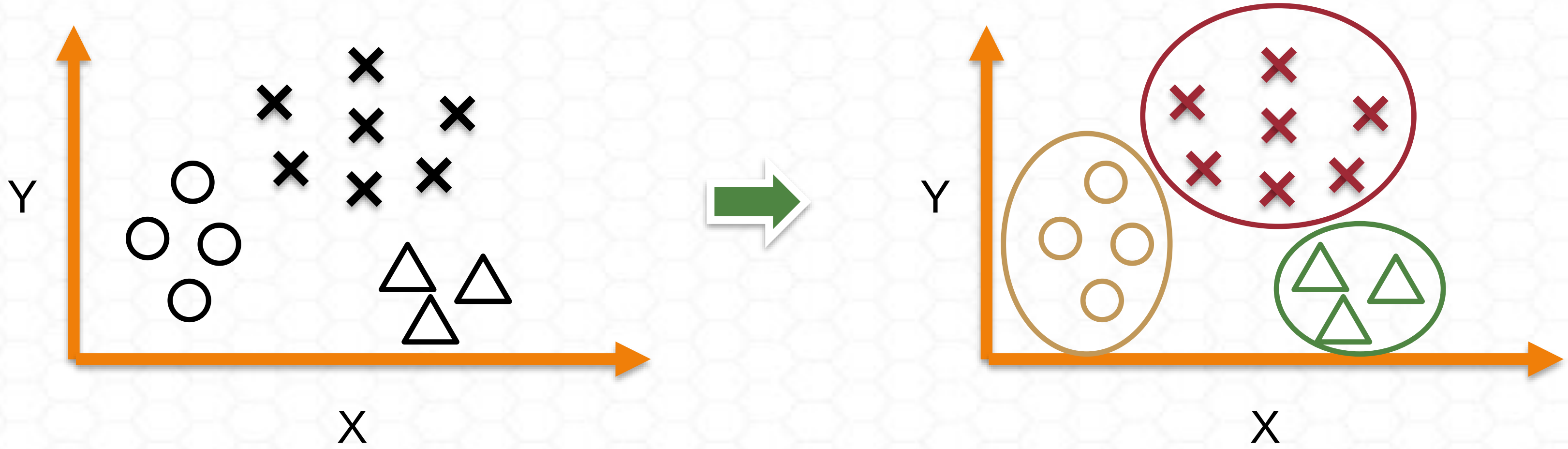


分群問題

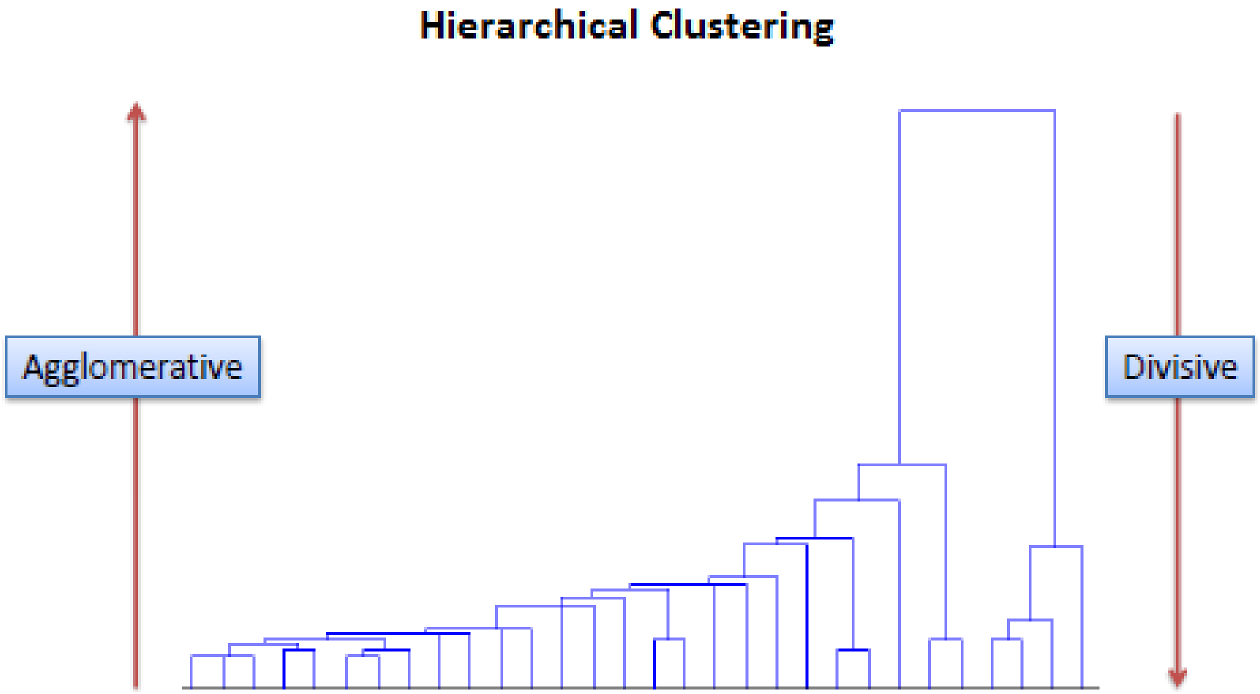
- 特色
  - 沒有正確答案 (標籤)
  - 依靠自身屬性相似度，物以類聚

- 如何判斷相似度
  - 以『距離』作為分類的依據，『相對距離』愈近的，『相似程度』愈高，歸類成同一群組。

資料分群



階層式分群



聚合式分群

- 階層式分群法可由樹狀結構的底部開始，將資料或群聚逐次合併
- 最終合併為一個大的群組

Given:

A set X of objects $\{x_1,\ldots,x_n\}$

A distance function $dist(c_1,c_2)$

for $i = 1$ to n

  $c_i = \{x_i\}$

end for

$C = \{c_1,\ldots,c_n\}$

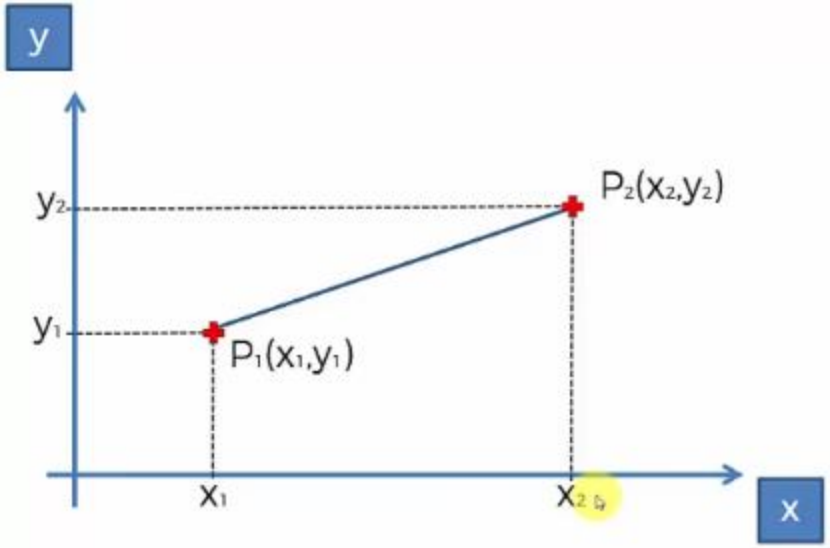
$l = n+1$

while C.size > 1 **do**

- (c_{min1},c_{min2}) = minimum $dist(c_i,c_j)$ for all c_i,c_j in C
- remove c_{min1} and c_{min2} from C
- add $\{c_{min1},c_{min2}\}$ to C
- $l = l + 1$

end while

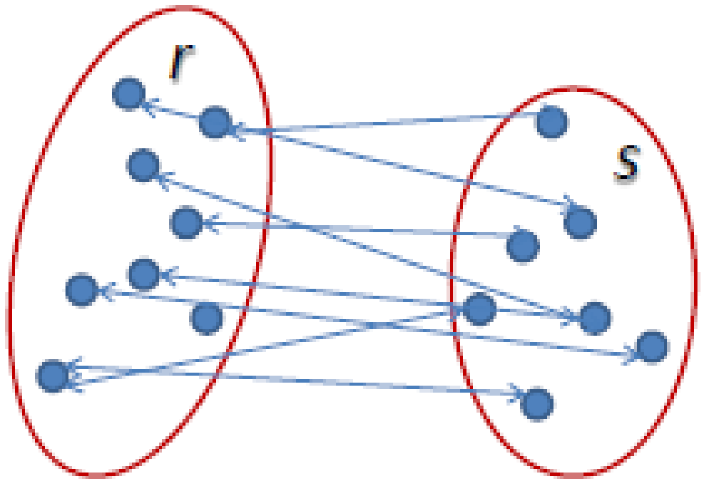
定義點之間的距離



Euclidean Distance between P_1 and $P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

歐式距離
曼哈頓距離
余弦距離
…

定義群之間的距離



- 單一連結聚合演算法

$$d(C_i, C_j) = \min_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} d(\mathbf{a}, \mathbf{b})$$

- 完整連結聚合演算法

$$d(C_i, C_j) = \max_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} d(\mathbf{a}, \mathbf{b})$$

- 平均連結聚合演算法

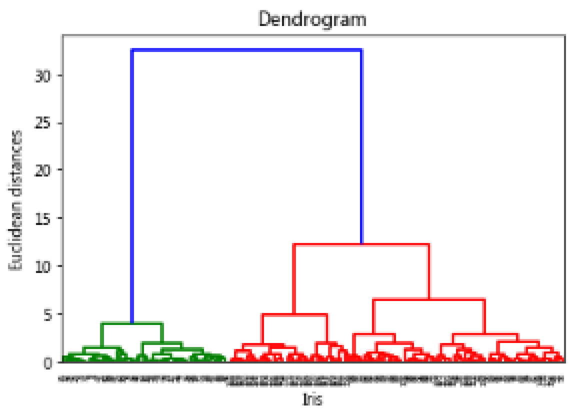
$$d(C_i, C_j) = \sum_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} \frac{d(\mathbf{a}, \mathbf{b})}{|C_i||C_j|},$$

- 沃德法

$$d(C_i, C_j) = \sum_{\mathbf{a} \in C_i \cup C_j} \|\mathbf{a} - \mu\|,$$

使用scipy 繪製樹狀圖

```
from sklearn.datasets import load_iris
iris = load_iris()

import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
dendrogram = sch.dendrogram(sch.linkage(iris.data, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Iris')
plt.ylabel('Euclidean distances')
plt.show()
```

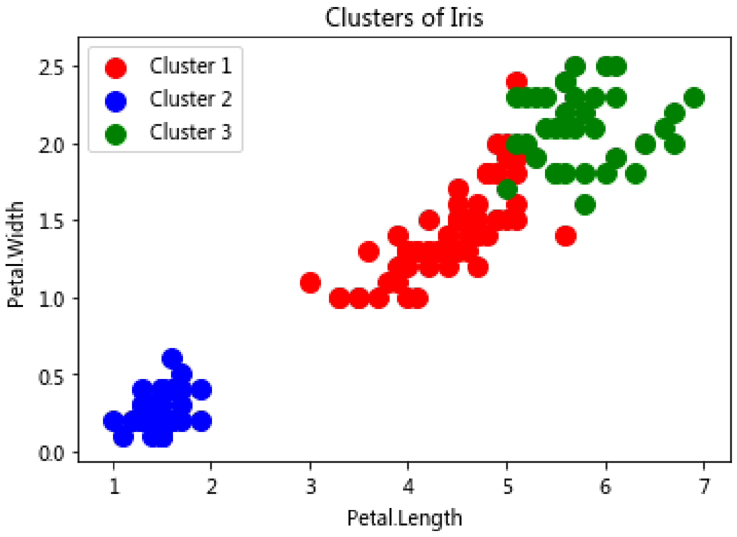


使用sklearn 分群

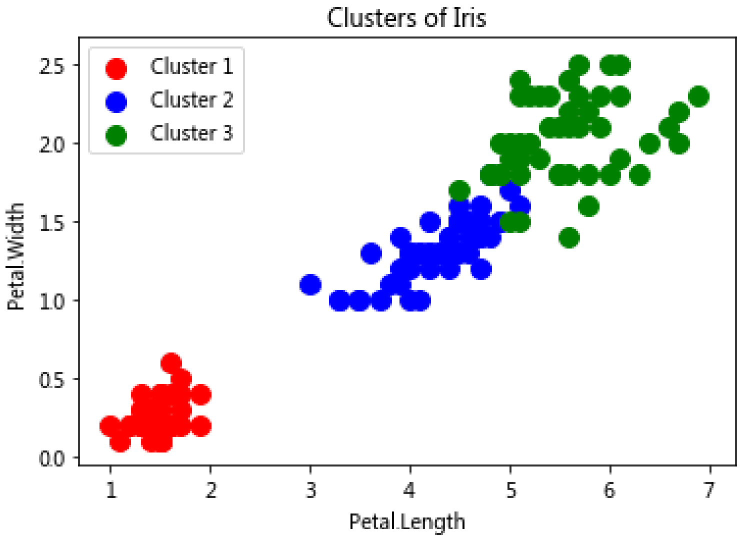
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(iris.data)

plt.scatter(iris.data[y_hc == 0, 2], iris.data[y_hc == 0, 3], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(iris.data[y_hc == 1, 2], iris.data[y_hc == 1, 3], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(iris.data[y_hc == 2, 2], iris.data[y_hc == 2, 3], s = 100, c = 'green', label = 'Cluster 3')

plt.title('Clusters of Iris')
plt.xlabel('Petal.Length')
plt.ylabel('Petal.Width')
plt.legend()
plt.show()
```

比較分群跟實際結果

分群結果



實際結果



階層式分群的優點/缺點

- 優點
  - 可以產生視覺化分群結果
  - 可以等結構產生後，再進行分群
  - 不用一開始決定要分多少群

- 缺點
  - 計算速度緩慢(採用遞迴式聚合或分裂)

K-Means 分群

- 最小化誤差函數 (Within cluster sum of squares by cluster)
  - 將資料分為k 群
  - 所有數據點 x_j 到其對應群中心 C_i 的距離總合是最小的

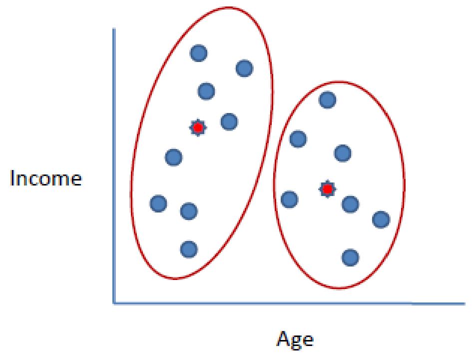
number of clusters

number of cases

centroid for cluster *j*

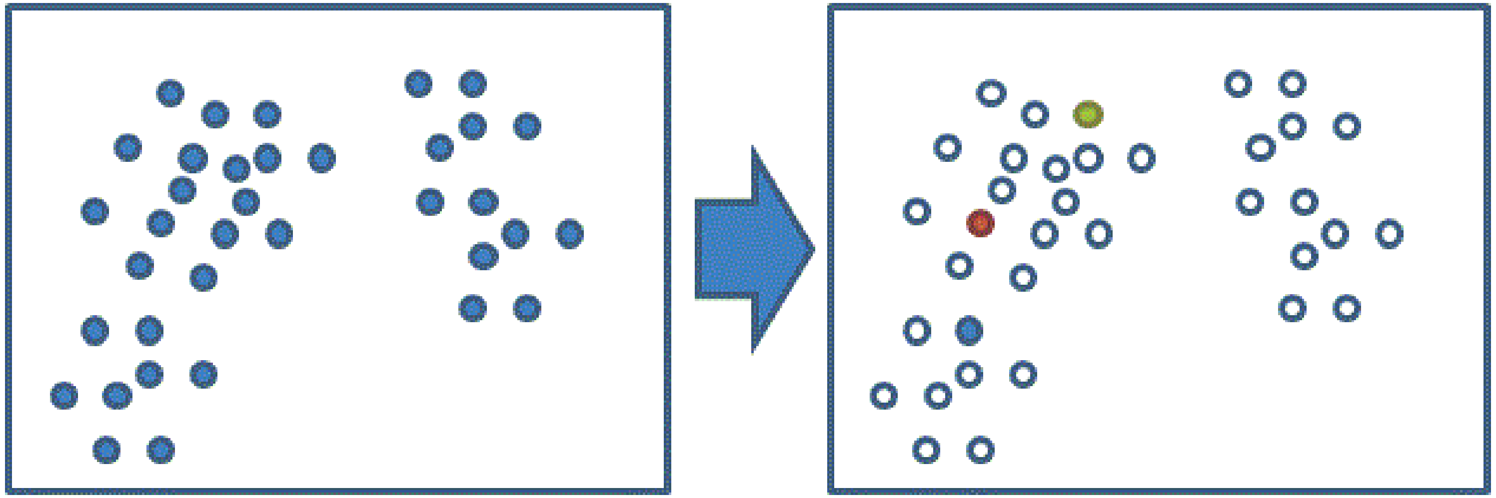
case *i*

$$\text{objective function} \leftarrow J = \sum_{j=1}^{k}\sum_{i=1}^{n} \underbrace{\left\| x_i^{(j)} - c_j \right\|^2}_{\text{Distance function}}$$



1. 隨機選取資料組中的k筆群中心

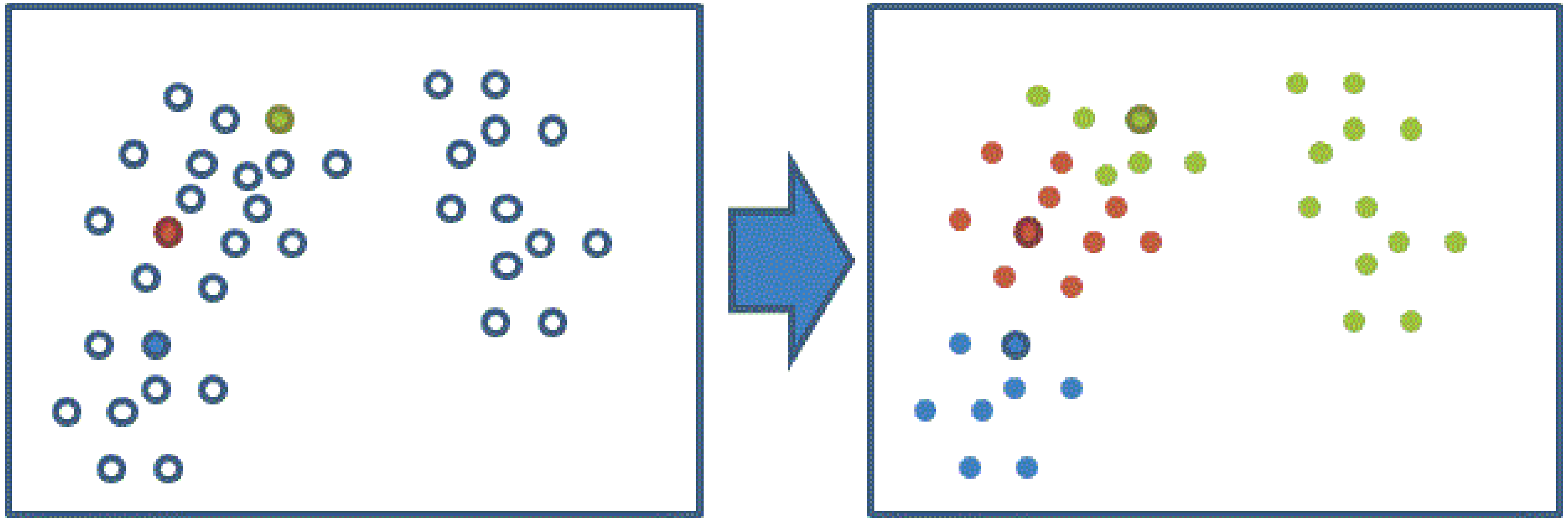
- 隨機選取資料組中的k筆資料當作初始群中心u_1~u_k



初始群中心設定的不好可能導致不好的結果

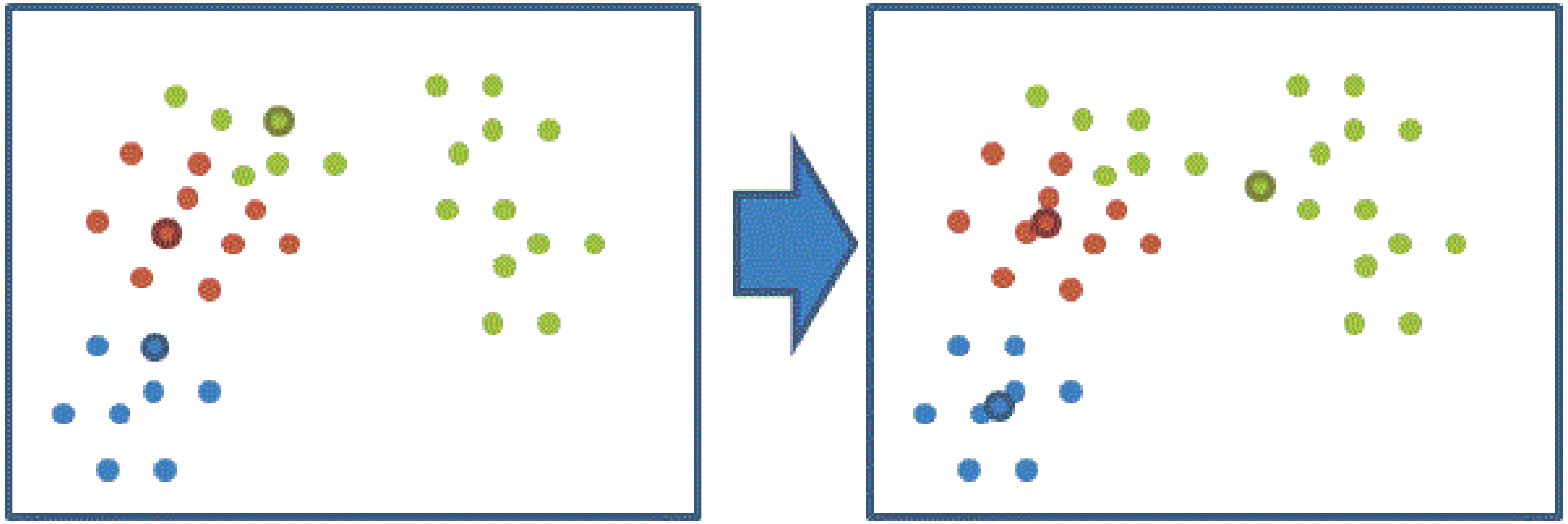
2. 計算每個資料xi 對應到最短距離的群中心

- 計算每個資料xi 對應到最短距離的群中心 **(固定 u_i 求解所屬群 S_i)**



3.利用目前得到的分類重新計算群中心

- 利用目前得到的分類重新計算群中心 **(固定 S_i 求解群中心 u_i)**



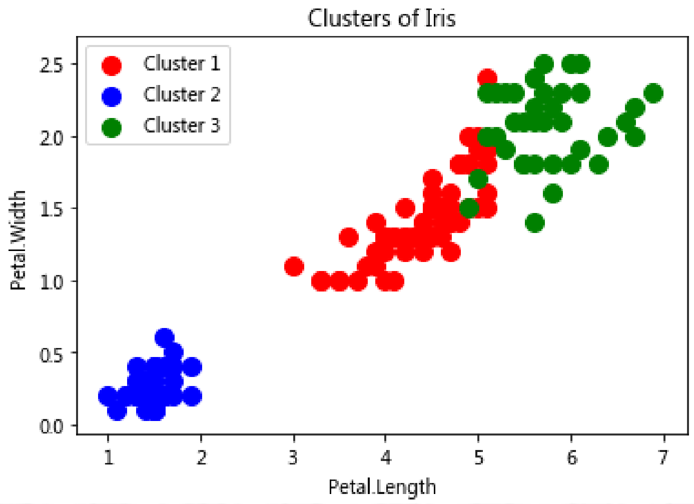
重複step 2,3直到收斂
(達到最大反覆運算次數 or 群心中移動距離很小)

使用Kmeans 分群

```
from sklearn.datasets import load_iris
iris = load_iris()

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 123)
y_kmeans = kmeans.fit_predict(iris.data)
y_kmeans
```

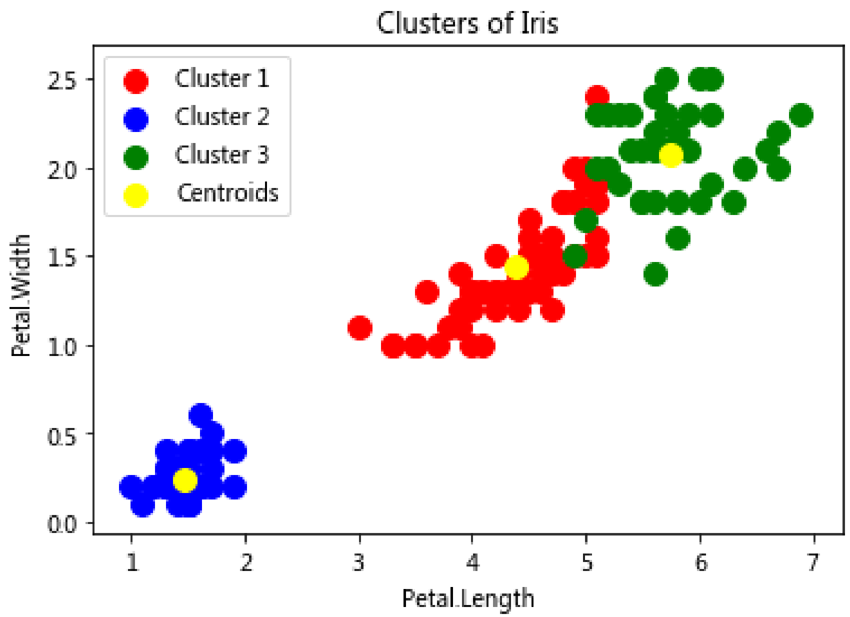
產生分群後的結果

```
import matplotlib.pyplot as plt
plt.scatter(iris.data[y_kmeans == 0, 2], iris.data[y_kmeans == 0, 3], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(iris.data[y_kmeans == 1, 2], iris.data[y_kmeans == 1, 3], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(iris.data[y_kmeans == 2, 2], iris.data[y_kmeans == 2, 3], s = 100, c = 'green', label = 'Cluster 3')

plt.title('Clusters of Iris')
plt.xlabel('Petal.Length')
plt.ylabel('Petal.Width')
plt.legend()
plt.show()
```



增加中心點

plt.scatter(kmeans.cluster_centers_[:, 2], kmeans.cluster_centers_[:, 3], s = 100, c = ‘yellow’, label = ‘Centroids')



客戶屬性分群

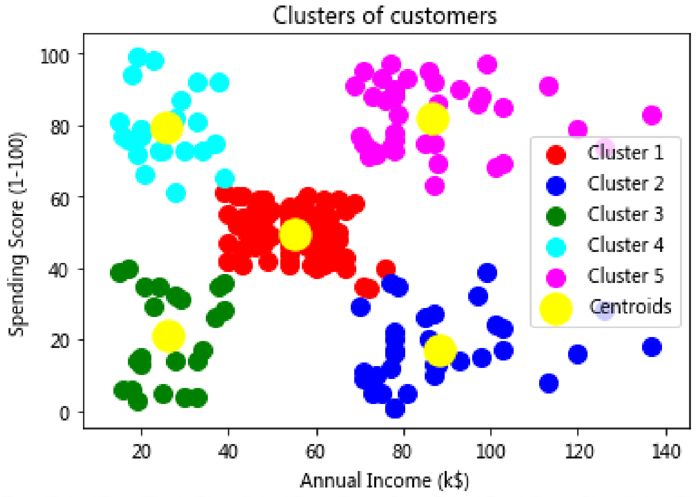
```
dataset = pd.read_csv('data/customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

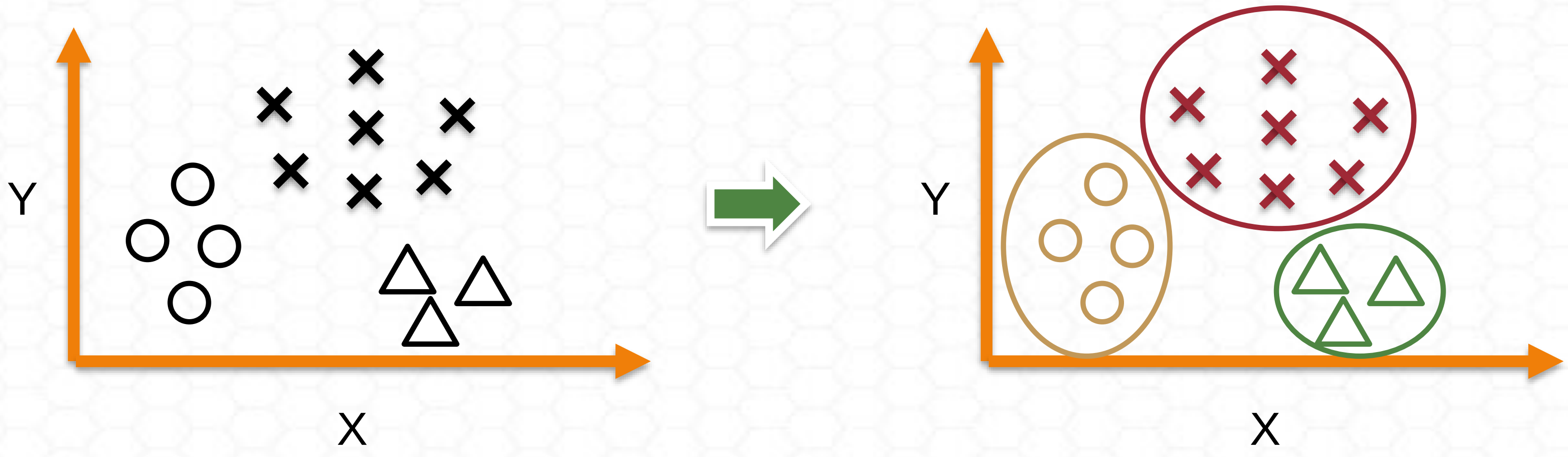
繪製客戶屬性分佈

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow',
label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



該將數據分成幾群?

分兩群好？　還是分三群好？

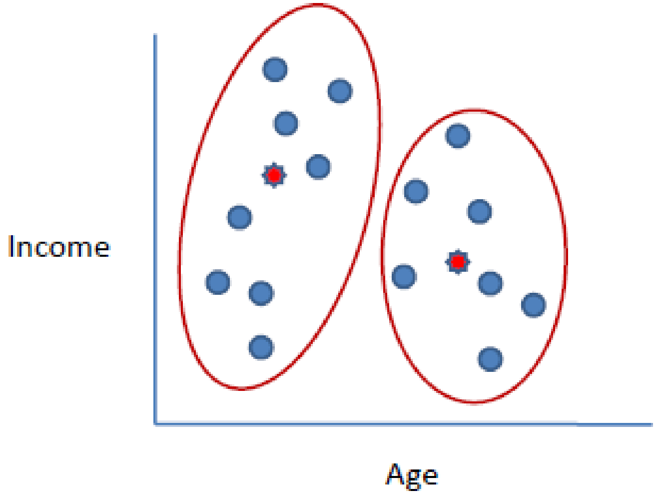


根據WCSS 決定

number of clusters　number of cases　centroid for clust

case i

objective function $\leftarrow$ $$J = \sum_{j=1}^{k}\sum_{i=1}^{n}\underbrace{\left\|x_i^{(j)} - c_j\right\|^2}_{\text{Distance function}}$$

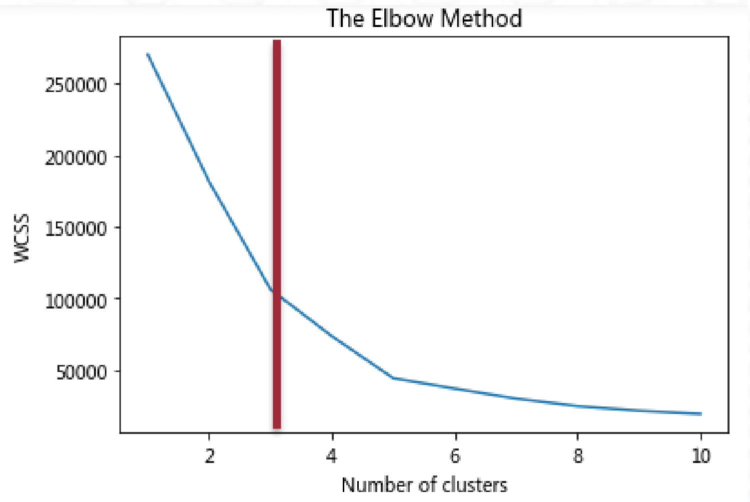


找出WCSS

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
kmeans.inertia_
```

找到坡度改變的地方

```
import matplotlib.pyplot as plt
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



評估分群效果

- 定義:
  - 群內之間點的平均距離 (群內的點平均距離越小)
  - 群間之間的平均距離 (群間點的平均距離越大)

$$\text{Silhouette(x)} = \frac{b(x) - a(x)}{\max([b(x), a(x)])}$$

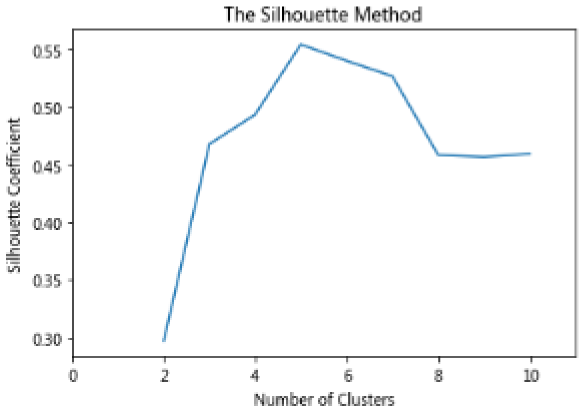
- a(x) 為 x 距離群內其他點的平均距離
- b(x) 是 x 距離其他群內點之間的最小平均距離

計算 Silhouette

```
from sklearn import metrics
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X, y_kmeans))
```

計算Silhouette Coefficient

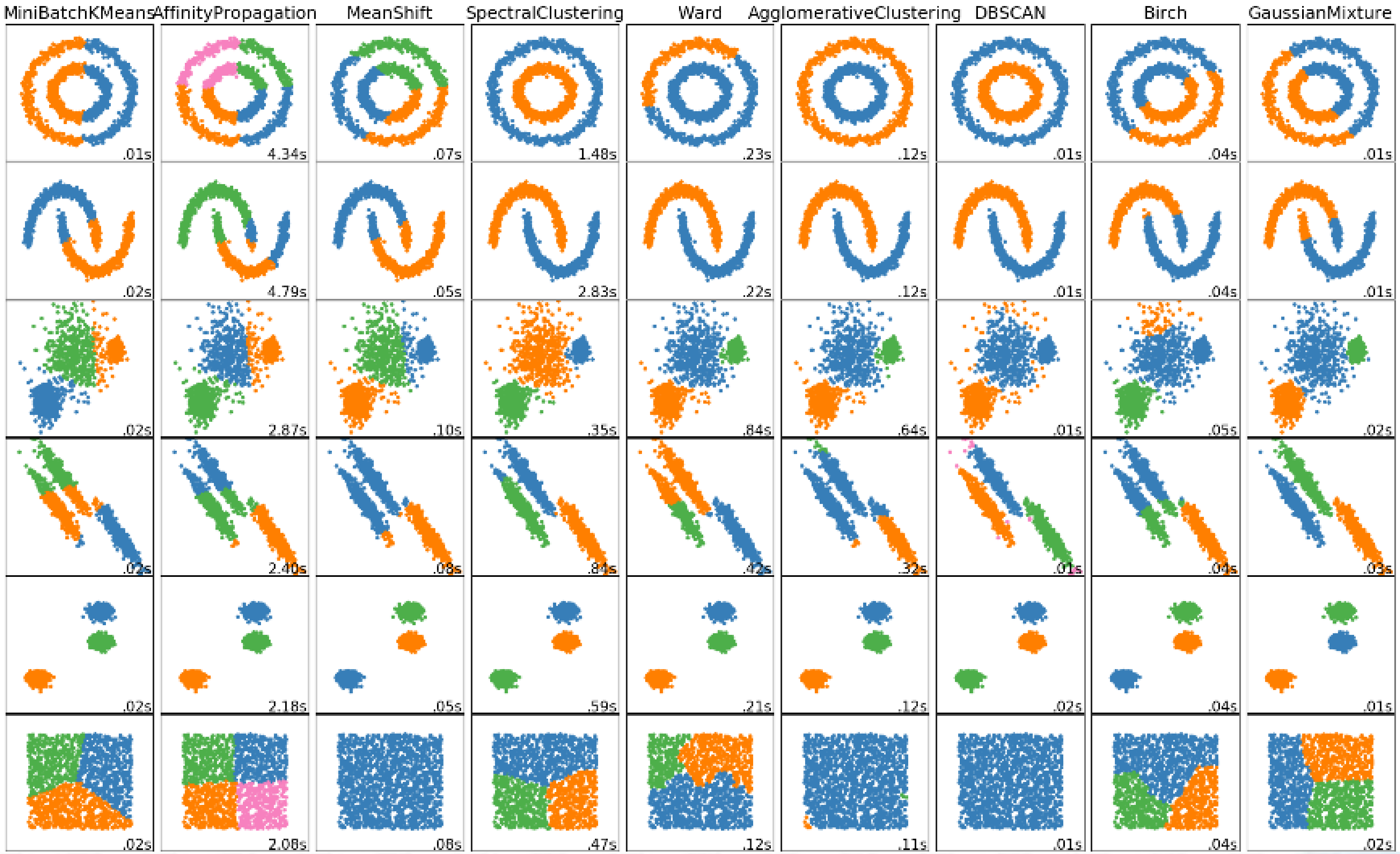
```
import matplotlib.pyplot as plt
sil = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    y_kmeans = kmeans.fit_predict(X)
    sil.append(metrics.silhouette_score(X, y_kmeans))
plt.plot(range(2, 11), sil)
plt.xlim([0,11])
plt.title('The Silhouette Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Coefficient')
plt.show()
```



同時比較不同分群方法

```
# ward
ward = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_ward = ward.fit_predict(X)

#complete
complete = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'complete')
y_complete = complete.fit_predict(X)

# kmeans
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

for est, title in zip([y_ward,y_complete, y_kmeans], ['ward', 'complete', 'kmeans']):
    print(title, metrics.silhouette_score(X, est))
```

各種分群方法


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
Birch
GaussianMixture
.01s
4.34s
.07s
1.48s
.23s
.12s
.01s
.04s
.01s
.02s
4.79s
.05s
2.83s
.22s
.12s
.01s
.04s
.01s
.02s
2.87s
.10s
.35s
.84s
.64s
.01s
.05s
.02s
.02s
2.40s
.08s
.84s
.42s
.32s
.01s
.04s
.03s
.02s
2.18s
.05s
.59s
.21s
.12s
.02s
.04s
.01s
.02s
2.08s
.08s
.47s
.12s
.11s
.01s
.04s
.02s

文章分群 (K-MEANS)

處理新聞文字內容

- 取得新聞內容

```
import pandas
news = pandas.read_excel('20150628news.xlsx')
```

- 將新聞斷詞

```
import jieba
titles = []
corpus = []
for rec in news.iterrows():
    titles.append(rec[1]['title'])
    corpus.append(' '.join(jieba.cut(rec[1]['description'])))
```

產生詞頻矩陣

- 利用CountVectorizer 產生詞頻矩陣

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
word = vectorizer.get_feature_names()
```

- 計算餘弦相似度

```
from sklearn.metrics.pairwise import cosine_similarity
n_cosine_similarities = cosine_similarity(X)
```

使用kmeans 進行分群

- 將資料分為四群

```
from sklearn import cluster
km = cluster.KMeans(n_clusters=4, init='k-means++', random_state=42)
c = km.fit_predict(n_cosine_similarities)
```

- 檢視分群結果

```
import numpy as np
titles_ary = np.array(titles)
titles_ary[c == 3]
```

```
羅志祥哭了 蔡依林讚表現很好
蔡依林淚奪金曲 錦榮傳訊恭喜
陳奕迅、張惠妹稱王封后 蔡依林抱回最大獎
…
```

評估分群效果

- 定義:
  - 群內之間點的平均距離 (群內的點平均距離越小)
  - 群間之間的平均距離 (群間點的平均距離越大)

$$\text{Silhouette}(\mathrm{x}) = \frac{b(x) - a(x)}{\max([b(x), a(x)])}$$

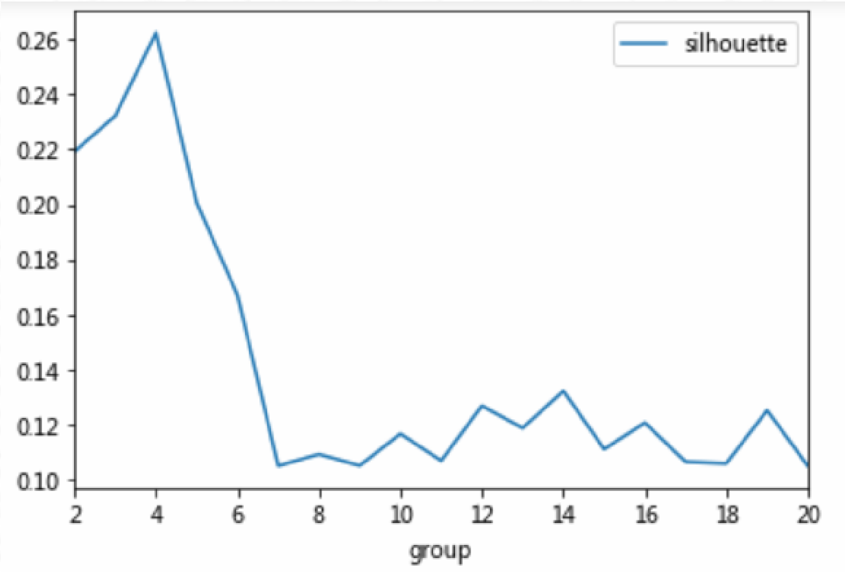
- a(x) 為 x 距離群內其他點的平均距離
- b(x) 是 x 距離其他群內點之間的最小平均距離

計算 Silhouette

```python
from sklearn.metrics import silhouette_score

sil_ary = []
for k in range(2,21):
    km = cluster.KMeans(n_clusters = k, init='k-means++', random_state=42)
    c = km.fit_predict(n_cosine_similarities)
    sil_ary.append({'group':k,
                    'silhouette':silhouette_score(n_cosine_similarities, labels=c)})
```

繪製 Silhouette 圖

```
% pylab inline
import pandas
sil_df = pandas.DataFrame(sil_ary)
sil_df.plot(kind = 'line', x='group', y='silhouette')
```



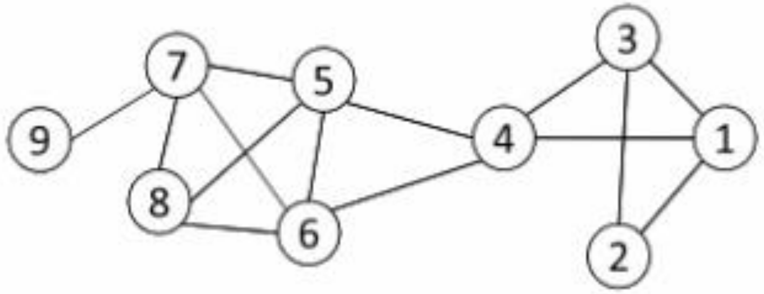
試著評估其他資料集?

文章分群 (社群偵測法)

網路圖

- 利用點(Node)與邊(Edge)的網路架構表示資料間的關係。在文本分析中可以用點描述文章，邊描述兩文章之間相似(e.g. 相似度高於0.5)

  - Graph Representation

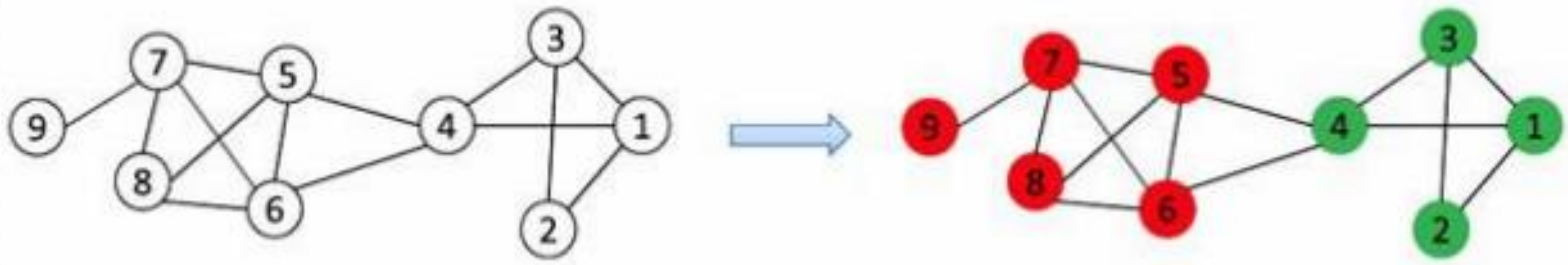


  - Matrix Representation

Node	1	2	3	4	5	6	7	8	9
1	-	1	1	1	0	0	0	0	0
2	1	-	1	0	0	0	0	0	0
3	1	1	-	1	0	0	0	0	0
4	1	0	1	-	1	1	0	0	0
5	0	0	0	1	-	1	1	1	0
6	0	0	0	1	1	-	1	1	0
7	0	0	0	0	1	1	-	1	1
8	0	0	0	0	1	1	1	-	0
9	0	0	0	0	0	0	1	0	-

社群架構 (Community Structure)

- 社群代表互動較為頻繁(互相連通)的點，可用做偵測交友圈，商品推薦或找尋文章主題用。



如何定義互動頻繁?

模組度 (Modularity Class)

- 模組度是評估一個社群網路好壞的方法，它的物理定義是社區內節點的連邊數與隨機情況下邊數的差距

$$Q = \frac{1}{2m}\sum_{c}[\Sigma in - \frac{(\Sigma tot)^2}{2m}] = \sum_{c}[e_c - {a_c}^2]$$

- $\sum in$ 代表社群c內邊權重和
- $\sum tot$ 代表與社群c內節點相連的邊權重和

代表社群c內部邊權重和與所有與社區c節點相連的邊權重和的差距

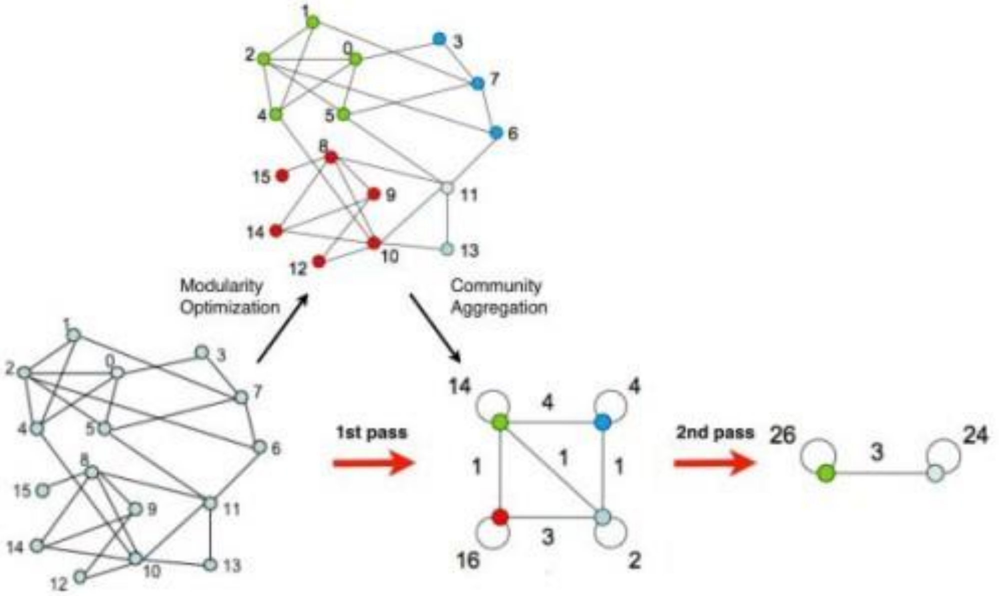
Louvain 演算法

- 找出完全互相連通的社群(Clique) 是一個NP-Complete 的問題
- Louvain 是啟發式(Heuristic) 貪婪 (Greedy) 演算法，希望能找出最大模組度(Modularity)的社群
  - 尋找所有擁有最高模組度的小社群
  - 聚合在同社群中的節點，建立一個新社群
  - 反覆聚合直到達到最大模組度

Louvain 演算法

Require: $G = (V, E)$, l^* a level threshold
Ensure: $\mathscr{P}$ a partition

```
l ← 0; G_0 ← G
repeat
    l ← l + 1
    Initialize a partition 𝒫_l of G_l(V_l, E_l)
    // First phase: Partition update
    repeat
        Nodes in a random permutation
        for all Nodes: v ∈ V_l do
            Move from σ_v to one selected σ_v' (v' is a neighbor of v)
        end for
    until no more change increases modularity
    // Second phase: Construct a new meta graph
    Replace each community by a node
    Replace connections between a pair of communities by one weighted edge
until 𝒫_l is not updated or l = l*.
Return 𝒫 corresponding to the roots of the hierarchical tree.
```



處理新聞文字內容

■ 取得新聞內容

```python
import pandas
news = pandas.read_excel('news.xlsx')
```

■ 將新聞斷詞

```python
import jieba
titles = []
corpus = []
for rec in news.iterrows():
    titles.append(rec[1]['title'])
    corpus.append(' '.join(jieba.cut(rec[1]['content'])))
```

產生詞頻矩陣

■ 利用CountVectorizer 產生詞頻矩陣

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
word = vectorizer.get_feature_names()
```

■ 計算餘弦相似度

```
from sklearn.metrics.pairwise import cosine_similarity
n_cosine_similarities = cosine_similarity(X)
```

使用Louvain 演算法偵測社群

- 建立網路圖

```
import networkx as nx
m = (n_cosine_similarities >= 0.5).astype(int)
G = nx.from_numpy_matrix(m)
```

- 偵測社群

```
import community
comm = community.best_partition(G)
cluster_ary = np.array(list(comm.values()) )
```

安裝 Python Louvain
pip install python-louvain

列出最多文章數的社群

- 建立標題的numpy array

```
titles_ary  = np.array(titles)
```

- 找出前10文章數的社群

```
from collections import Counter
c = Counter(cluster_ary)
for group, cnt in c.most_common(10):
    articles = titles_ary[cluster_ary == group]
    for news in articles:
        print(news)
```

DBSCAN

密度為基礎分群法（DBSCAN）

- **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise

- 以密度為基礎的分群方式，用密度的概念剔除不屬於所有分群資料的雜訊點

DBSCAN示意圖

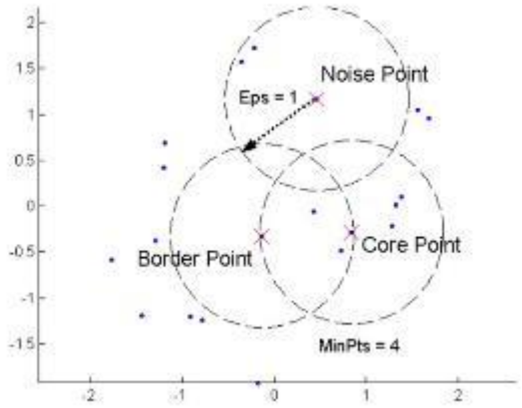
1. Eps = Density = 以資料點為圓心所設的半徑長度

2. Core Point（核心點）：以核心點為半徑所圍繞出來的範圍

3. Border Point（ 邊界點）：被某個核心點包含

4. Noise Point（雜訊點）：不屬於核心點，也不屬於邊界點，即為雜訊點

5. 密度相連：如果兩個核心點互為邊界點的話，則可把兩個核心點合併在同一個群組中



DBSCAN演算法

1. 將所有的點做過一次搜尋，找出**核心點、邊界點、雜訊點**
2. 移除所有雜訊點
3. SET「當前群集編號」= 0
4. FOR 1 到 最後一個核心點 do
5. IF 這個核心點並沒有被貼上群組編號 則
6. 「當前群集編號」的變數 + 1
7. 把「當前群集編號」給這個被抽出的核心點
8. END
9. FOR 這個核心點在密度相連後所有可以包含的點 do
10. IF 這個點還沒有被貼上任何群組編號 則
11. 把這個點貼上「當前變數的編號」
12. END
13. END FORLOOP
14. END FORLOOP

與K-means 比較

- 優點
  - 與K-means方法相比，DBSCAN不需要事先知道K
  - 與K-means方法相比，DBSCAN可以找到任意形狀
  - DBSCAN能夠識別出雜訊點
  - DBSCAN對於資料庫中樣本的順序不敏感

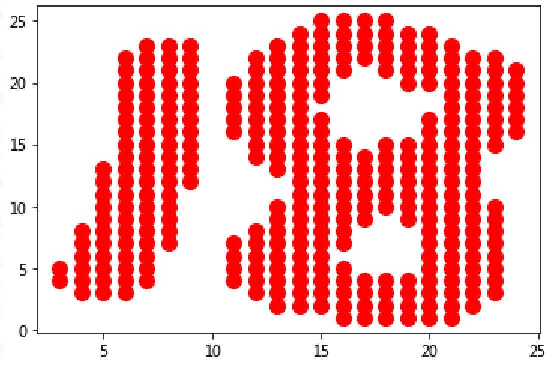
- 缺點
  - DBScan不適合反映高維度資料。
  - DBScan不適合反映已變化資料的密度

將影像讀取成numpy array

```python
import numpy as np
from PIL import Image
img = Image.open('data/handwriting.png')
img2 = img.rotate(-90).convert("L")
imgarr = np.array(img2)
```

呈現影像資料

```
from sklearn.preprocessing import binarize
imagedata = np.where(1- binarize(imgarr, 0) == 1)

import matplotlib.pyplot as plt
plt.scatter(imagedata[0], imagedata[1], s = 100, c = 'red', label = 'Cluster 1')
plt.show()
```

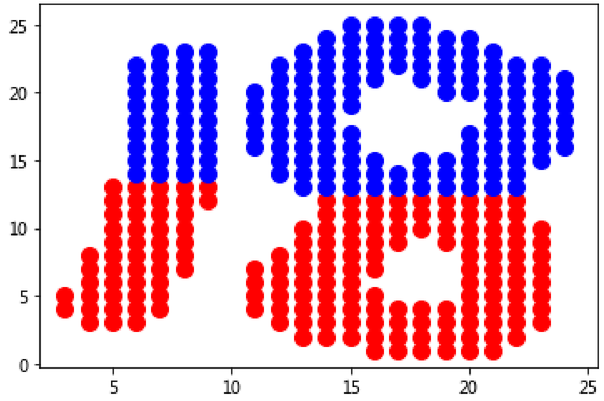


使用KMeans 分群

```
from sklearn.cluster import KMeans
X =np.column_stack([imagedata[0],imagedata[1]])
kmeans = KMeans(n_clusters = 2, init = 'k-means++', random_state
= 42)
y_kmeans = kmeans.fit_predict(X)
```

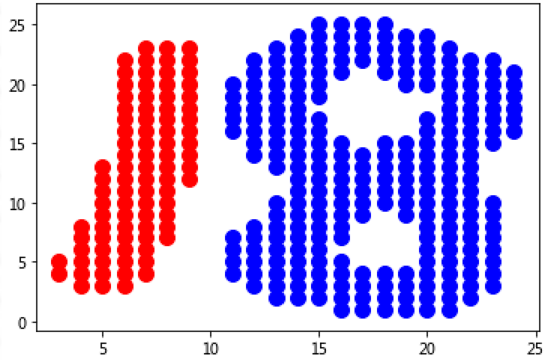
呈現分群結果

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c =
'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c =
'blue', label = 'Cluster 2')
plt.show()
```



使用DBSCAN 分群

```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=1, min_samples=3)
y_dbs = dbs.fit_predict(b)

plt.scatter(X[y_dbs == 0, 0], X[y_dbs == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_dbs == 1, 0], X[y_dbs == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.show()
```



特徵篩選與降低維度

降低維度

- 影響事情發展的因素是多元性的;但不同因素之間會互相影響(共線性)，或相重迭;進而影響到統計結果的真實性
  - 使用降低維度來降低訊息重迭
  - 使用降低維度來減少工作量
  - 找出一個互不相關的綜合指標來反映原本資料所含大部分的訊息

降低維度的應用

- 透過產生一有最大變異數的欄位線性組合，可用來降低原本問題的維度與複雜度
  - e.g. 濃縮用到的特徵，編纂成一個新指標
  - 產生經濟指標
- 去掉不必要的變數
  - e.g.減少工作量，還有避免誤用指標
  - 變數排序與篩選

降低維度的方法

- 選擇特徵 (Feature Selection)
  - 從原有的特徵中挑選出最佳的部分特徵
  - 能夠簡化分類器的計算
  - 幫助瞭解分類問題的因果關係
- 抽取特徵 (Feature Extraction)
  - 將資料群由高維度的空間中投影到低維度的空間
  - 找出一組基底向量（ base ）來進行線性座標轉換
  - 使得轉換後的座標，能夠符合某一些特性

移除低變異數的特徵

```
import pandas
from sklearn.feature_selection import VarianceThreshold
df = pandas.read_csv('data/customer_behavior.csv')
X = df[['bachelor','gender', 'age', 'salary']]
sel = VarianceThreshold()
X_val = sel.fit_transform(X)

names = df.columns[sel.get_support()]
```

單變數特徵選擇 (Univariate Feature Selection)

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X = df[['bachelor','gender', 'age', 'salary']]
y = df['purchased'].values
clf = SelectKBest(chi2, k=2)
clf.fit(X,y)
print(clf.scores_)

X_new = clf.fit_transform(X,y)
print(X_new)
```

選擇最佳兩個特徵

另外可以選用
Regression: f_regression
Classfication: chi2, f_classif

暴力法選擇特徵

1 input → x1 x2 x3 x4 x5

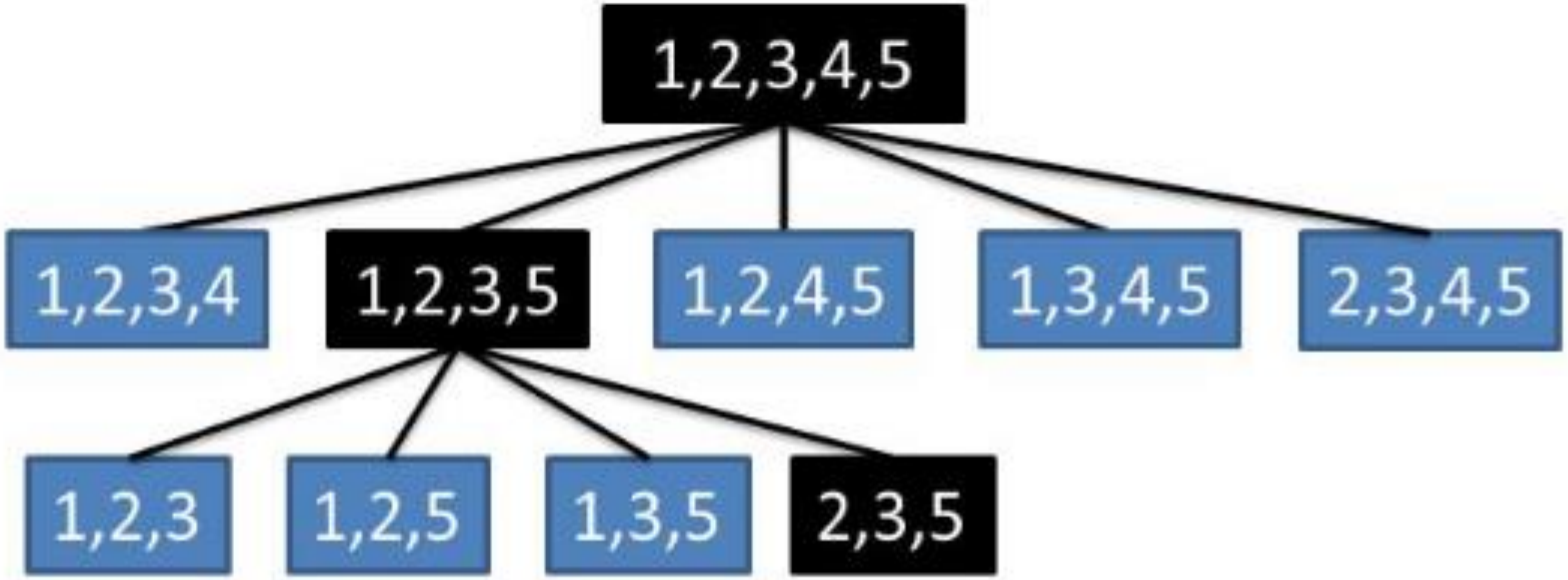
2 inputs → x1, x2 | x1, x3 | x1, x4 | x1, x5 | x2, x3 ...

3 inputs → x1, x2, x3 | x1, x2, x4 | x1, x2, x5 | x1, x3, x4 ...

4 inputs → x1, x2, x3, x4 | x1, x2, x3, x5 | x1, x2, x4, x5 ...

Recursive feature elimination

- 逐步剔除特徵以找到最好的特徵組合



Recursive feature elimination

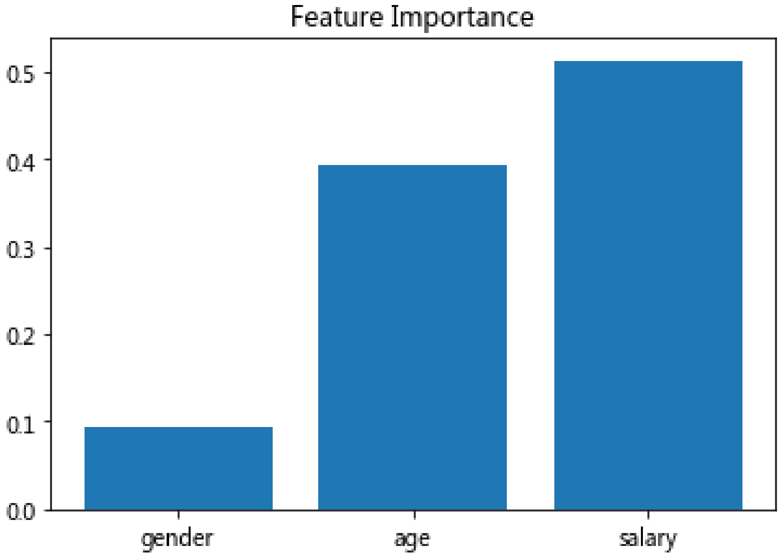
```
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

clf = SVC(kernel='linear')

rfe = RFE(clf, n_features_to_select=1)
rfe.fit(X_val,y)

for x in rfe.ranking_:
    print(names[x-1], rfe.ranking_[x-1])
```

使用隨機森林篩選變數

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10,
random_state=123)

clf.fit(X_val, y)

names, clf.feature_importances_
for feature in zip(names, clf.feature_importances_):
    print(feature)
```

視覺化呈現 Feature Importance

```
import matplotlib.pyplot as plt
plt.title('Feature Importance')
plt.bar(range(0, len(names)), clf.feature_importances_)
plt.xticks(range(0,len(names)), names)
plt.show()
```



抽取特徵

- 抽取特徵 (Feature Extraction)
  - 將資料群由高維度的空間中投影到低維度的空間
  - 找出一組基底向量（base）來進行線性座標轉換
  - 使得轉換後的座標，能夠符合某一些特性

主成分分析的歷史

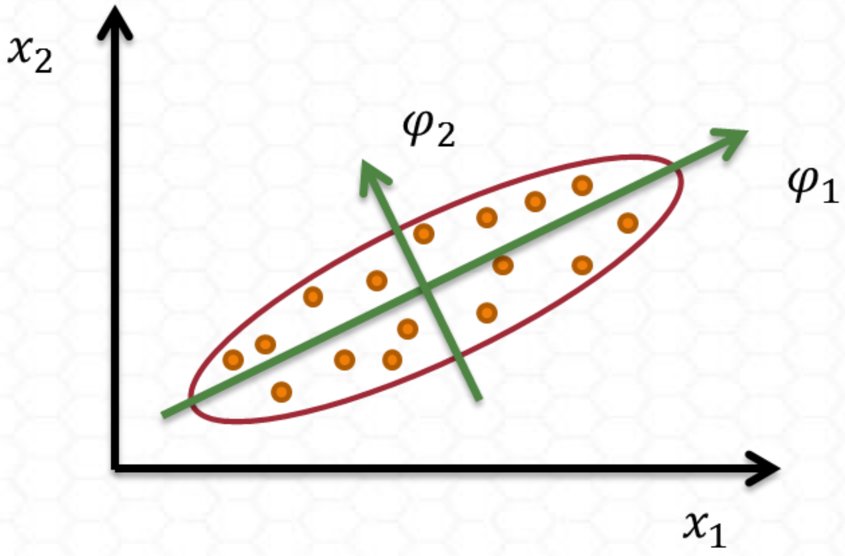
- Stone 於1974 年對美國1929 ~ 1938 年經濟資料的研究
  - 透過降低維度找到三個主成分(解釋度達97.4%)，以總結17個變數
    - F1 總收入
    - F2 總收入變化率
    - F3 經濟發展趨勢
  - 主成分保留了原始變數大部分的訊息
  - 主成分個數少於原變數的個數
  - 主成分之間互不相關
  - 每個主成分都是原始變數的線性組合

主成分分析模型

- X1...Xp為p維向量，主成分分析可將p 個觀測量透過縣性組合轉換為p個新指標
  - $F_1 = u_{11}X_1 + u_{12} X_2 + \ldots u_{1p}X_p$
  - $F_2 = u_{21}X_1 + u_{22} X_2 + \ldots u_{2p}X_p$
  - $F_p = u_{p1}X_1 + u_{p2} X_2 + \ldots u_{pp}X_p$
- 滿足條件如下
  - 主成分係數平方和為1$u_{i1}{}^2 + u_{i2}{}^2 + \cdots u_{ip}{}^2 = 1$
  - 主成分之間相互獨立 cov(Fi,Fj) = 0, $i \neq j$
  - 主成分的方差依重要性遞減
    - $Var(F_1) \geq Var(F_2) \ldots \geq Var(F_p)$

主成分分析目的

- 簡化變數
  - 主成分的個數小於原始變數的個數
  - 主成分盡可能反映原來變數的訊息

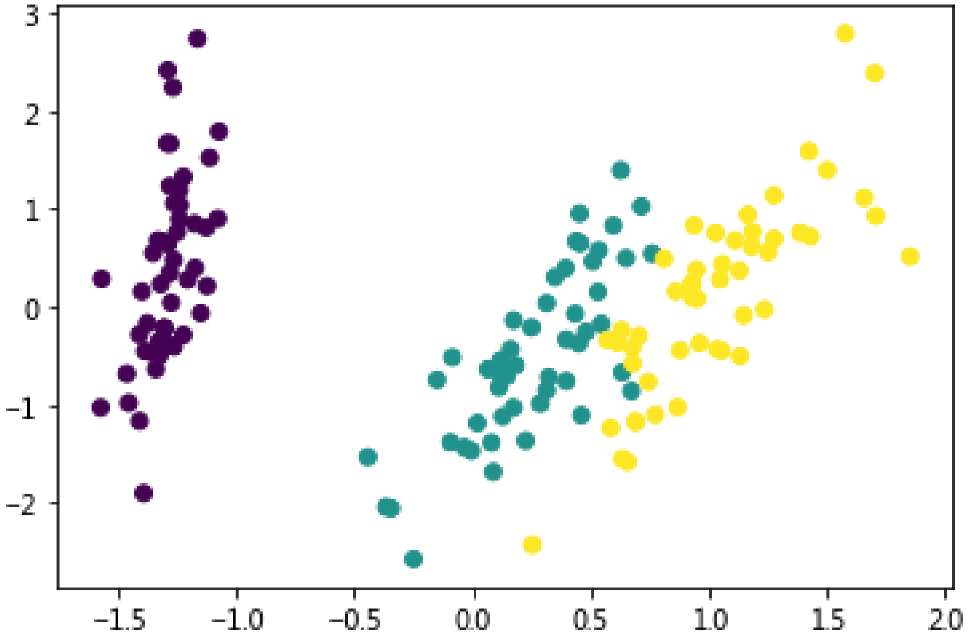


主成分分析

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

X_reduced = pca.transform(X)
X_reduced.shape
```

根據主成分繪製散佈圖

```
from matplotlib import pyplot as plt
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y)
plt.show()
```



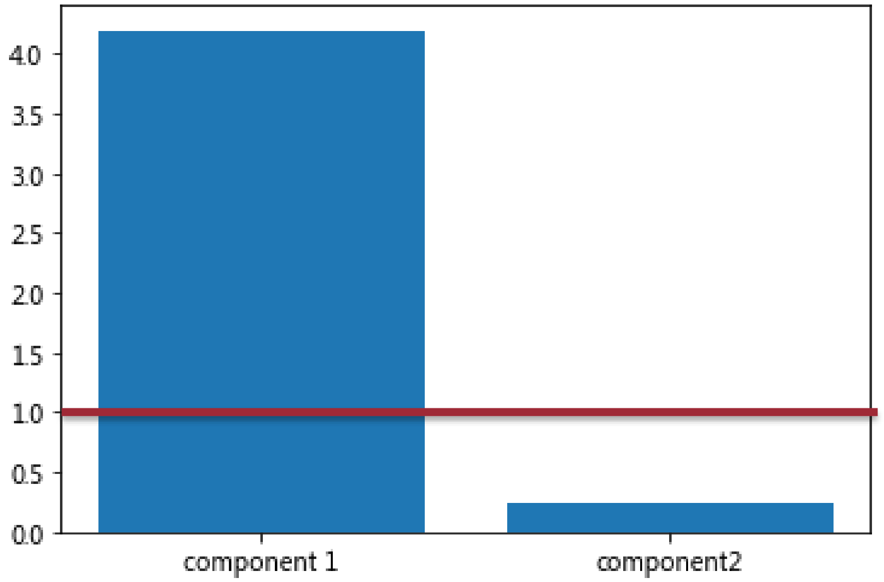
主成分組成

```python
for component in pca.components_:
    print(" + ".join("%.3f x %s" % (value, name)
                     for value, name in zip(component, iris.feature_names)))
```

```
0.362 x sepal length (cm) + -0.082 x sepal width (cm) + 0.857 x petal length (cm) + 0.359 x petal width (cm)
0.657 x sepal length (cm) + 0.730 x sepal width (cm) + -0.176 x petal length (cm) + -0.075 x petal width (cm)
```

變異數解釋量

```
plt.bar(range(0,2), pca.explained_variance_)
plt.xticks(range(0,2), ['component 1', 'component2'])
plt.show()
```



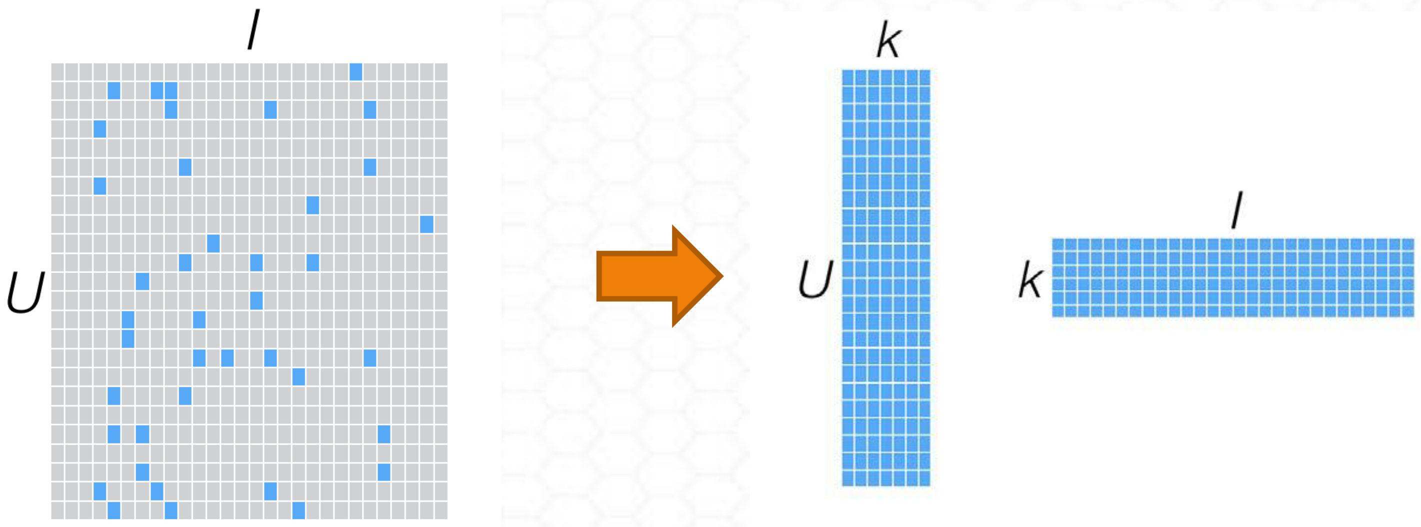
奇異值分解(SVD)

- 對特定資料集合做拆解(Matrix Factorization)，以便找出相對數量少卻富含重要資訊的要素組成新資料集合，並以此來近似原先的資料集合

- 目的:
  - 達到以簡馭繁 (A low-dimensional representation of a high-dimensional matrix)
  - 減少雜訊

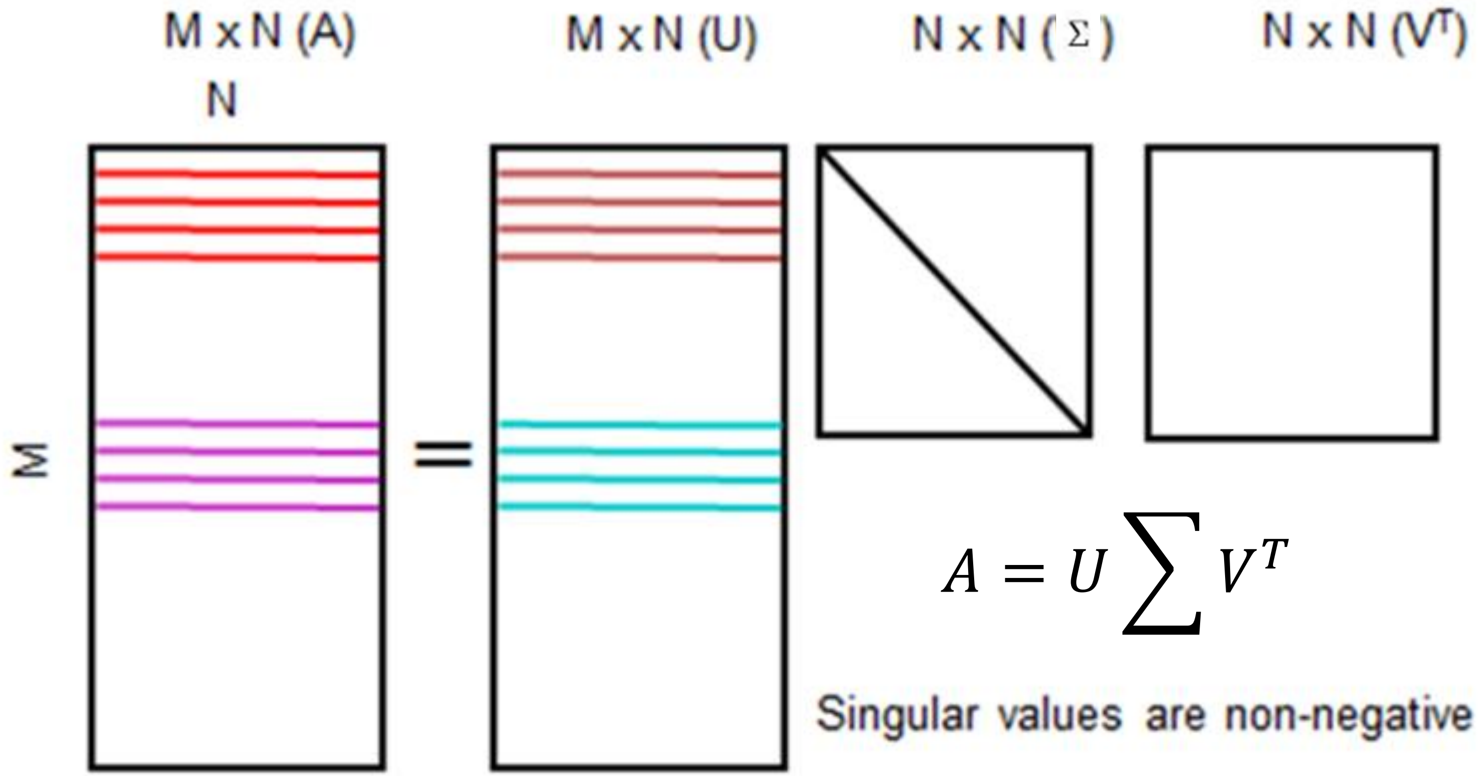
分解、保存矩陣

- 推薦系統
- 檔主題查找
- 降低維度



稀疏矩陣 (低秩矩陣)

奇異值分解(SVD)概念



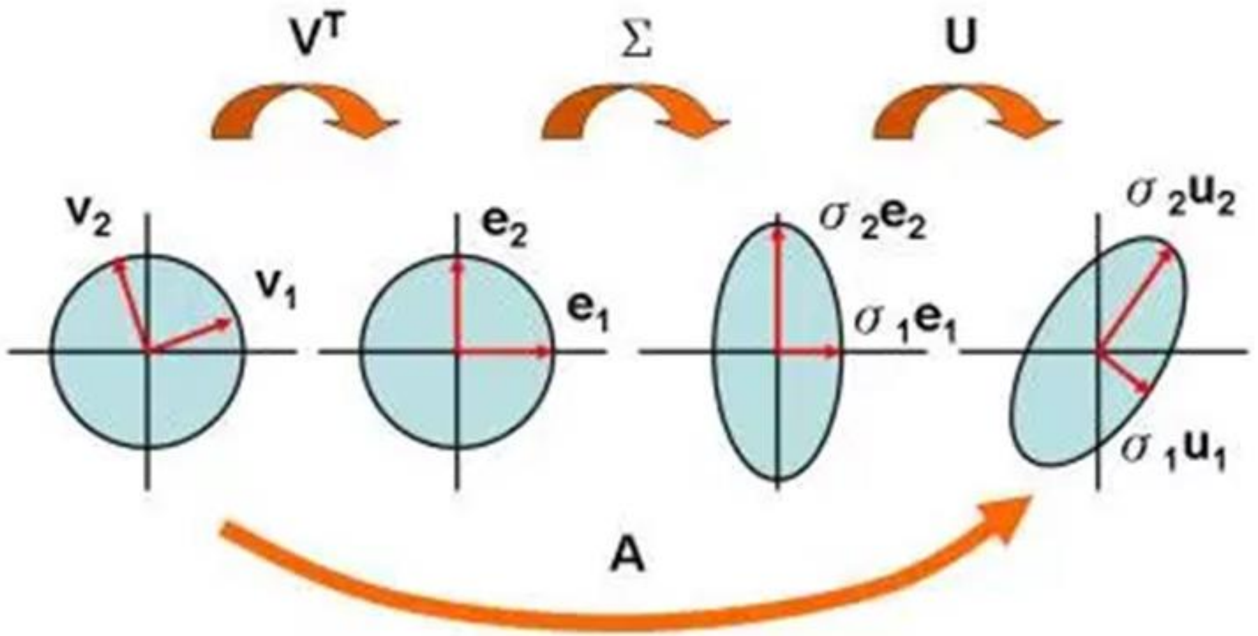
$$A = U \sum V^T$$

Singular values are non-negative

類似PCA，唯PCA 作用於協方差矩陣，SVD 用於一般矩陣

矩陣分解變換

- 旋轉 V^T，伸縮 Σ ，再旋轉 U



使用SVD 做矩陣還原

```
from scipy.linalg import svd
U, S, V = svd(X, full_matrices=False)
U.shape, S.shape, V.shape
np.diag(S)
np.dot(U.dot(np.diag(S)), V)
```

使用sklearn 的 TruncatedSVD

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(2)
X = svd.fit_transform(iris.data)
```

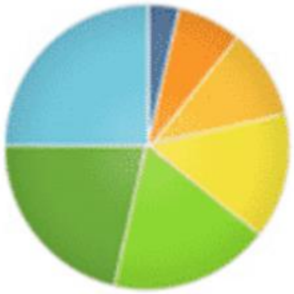
使用matplotlib 繪製視覺化結果

```
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=iris.target)
plt.xlabel('SVD1')
plt.ylabel('SVD2')
plt.title('SVD')
plt.show()
```

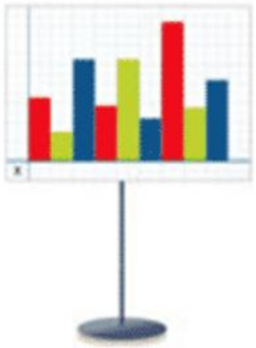
借貸俱樂部資料清理

Lending Club

Borrowers apply for loans.
Investors open an account.



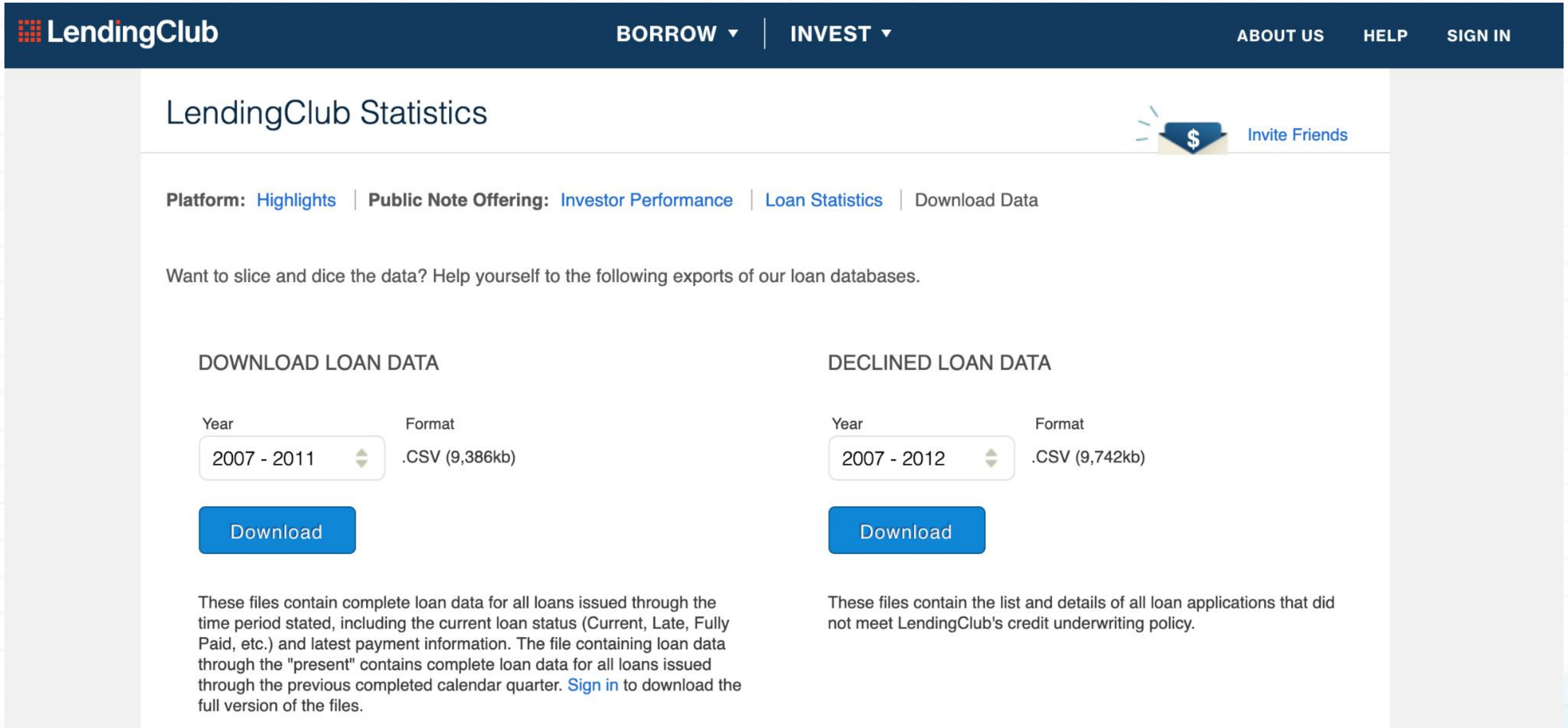
Borrowers get funded.
Investors build a portfolio.



Borrowers repay automatically.
Investors earn & reinvest.

借貸俱樂部資料

- https://www.lendingclub.com/info/download-data.action



資料清理

```
import pandas as pd

#讀取資料
dataset = pd.read_csv('LoanStats.csv')

#移除空白欄位
dataset = dataset.iloc[:,2:111]
empty_cols = [i for i in range(45,72)]
dataset = dataset.drop(dataset.columns[empty_cols],axis=1)
data_with_loanstatus_sliced = dataset[(dataset['loan_status']=="Fully Paid") | (dataset['loan_status']=="Charged Off")]

#轉換目標編碼
di = {"Fully Paid":0, "Charged Off":1}
Dataset_withBoolTarget= data_with_loanstatus_sliced.replace({"loan_status": di})
```

移除空白列與欄位

```
#移除空白列
dataset=Dataset_withBoolTarget.dropna(thresh = 340000,axis=1)
print("Current shape of dataset :",dataset.shape)

#移除欄位
del_col_names = ["delinq_2yrs",  "last_pymnt_d", "chargeoff_within_12_mths","delinq_amnt","emp_title", "term", "emp_title", "pymnt_plan","purpose","title", "zip_code", "verification_status", "dti","earliest_cr_line", "initial_list_status", "out_prncp",
"pymnt_plan", "num_tl_90g_dpd_24m", "num_tl_30dpd", "num_tl_120dpd_2m", "num_accts_ever_120_pd", "delinq_amnt",
"chargeoff_within_12_mths", "total_rec_late_fee", "out_prncp_inv", "issue_d"]
dataset = dataset.drop(labels = del_col_names, axis = 1)
print("Current shape of dataset :",dataset.shape)
```

篩選欄位

```
#篩選欄位
features = ['funded_amnt','emp_length','annual_inc','home_ownership','grade',
            "last_pymnt_amnt", "mort_acc", "pub_rec", "int_rate", "open_acc","num_actv_rev_tl",
            "mo_sin_rcnt_rev_tl_op","mo_sin_old_rev_tl_op","bc_util","bc_open_to_buy",
            "avg_cur_bal","acc_open_past_24mths",'loan_status'] #'sub_grade' #selecting final features #'addr_state''tax_liens',
Final_data = dataset[features] #19 features with target var
Final_data["int_rate"] = Final_data["int_rate"].apply(lambda x:float(x[:-1]) ) #reomving % sign, conv to float  - int_rate column
Final_data= Final_data.reset_index(drop=True)
print("Current shape of dataset :",Final_data.shape)
```

資料轉換

```
#Data encoding
Final_data['grade'] = Final_data['grade'].map({'A':7,'B':6,'C':5,'D':4,'E':3,'F':2,'G':1})
Final_data["home_ownership"] = Final_data["home_ownership"].map({"MORTGAGE":6,"RENT":5,"OWN":4,"OTHER":3,"NONE":2,"ANY":1})
Final_data["emp_length"] = Final_data["emp_length"].fillna('0')
Final_data["emp_length"] = Final_data["emp_length"].replace({'years':'','year':'',' ':'','<':'','\+':'','n/a':'0'}, regex = True)
Final_data["emp_length"] = Final_data["emp_length"].apply(lambda x:int(x))
print("Current shape of dataset :",Final_data.shape)
Final_data.head()
```

使用平均值填補遺失值

```
Final_data.fillna(Final_data.mean(),inplace = True)
print("Current shape of dataset :",Final_data.shape)
```

將特徵值標準化

```
from sklearn import preprocessing,metrics
scl = preprocessing.StandardScaler()
fields = Final_data.columns.values[:-1]
data_clean = pd.DataFrame(scl.fit_transform(Final_data[fields]),
columns = fields)
data_clean['loan_status'] = Final_data['loan_status']
data_clean['loan_status'].value_counts()
```

合併清理過後的資料

```
loanstatus_0 = data_clean[data_clean["loan_status"]==0]
loanstatus_1 = data_clean[data_clean["loan_status"]==1]
subset_of_loanstatus_0 = loanstatus_0.sample(n=5500)
subset_of_loanstatus_1 = loanstatus_1.sample(n=5500)
data_clean = pd.concat([subset_of_loanstatus_1,
subset_of_loanstatus_0])
data_clean = data_clean.sample(frac=1).reset_index(drop=True)
print("Current shape of dataset :",data_clean.shape)
data_clean.head()a
```

將資料區分為訓練與測試資料集

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_clean.iloc[:,:-1], data_clean.iloc[:,-1], test_size=0.2, random_state=42)
```

Recursive Feature Elimination

```
from sklearn import linear_model,svm
from sklearn.feature_selection import RFE
# create the RFE model and select 3 attributes
clf_LR = linear_model.LogisticRegression(C=1e30)
clf_LR.fit(X_train,y_train)
rfe = RFE(clf_LR, 10)
rfe = rfe.fit(data_clean.iloc[:,:-1].values, data_clean.iloc[:,-1].values)
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10, whiten=True)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print('Expected Variance is '+ str(explained_variance))
```

資料篩選

```
features = ['funded_amnt','annual_inc','grade',"last_pymnt_amnt", "int_rate", "mo_sin_rcnt_rev_tl_op","mo_sin_old_rev_tl_op","bc_util","bc_open_to_buy","acc_open_past_24mths","loan_status"]
X_train, X_test = X_train[features[:-1]], X_test[features[:-1]]
data_clean = data_clean[features]
print(X_train.shape)
print(data_clean.shape)
```

THANK YOU