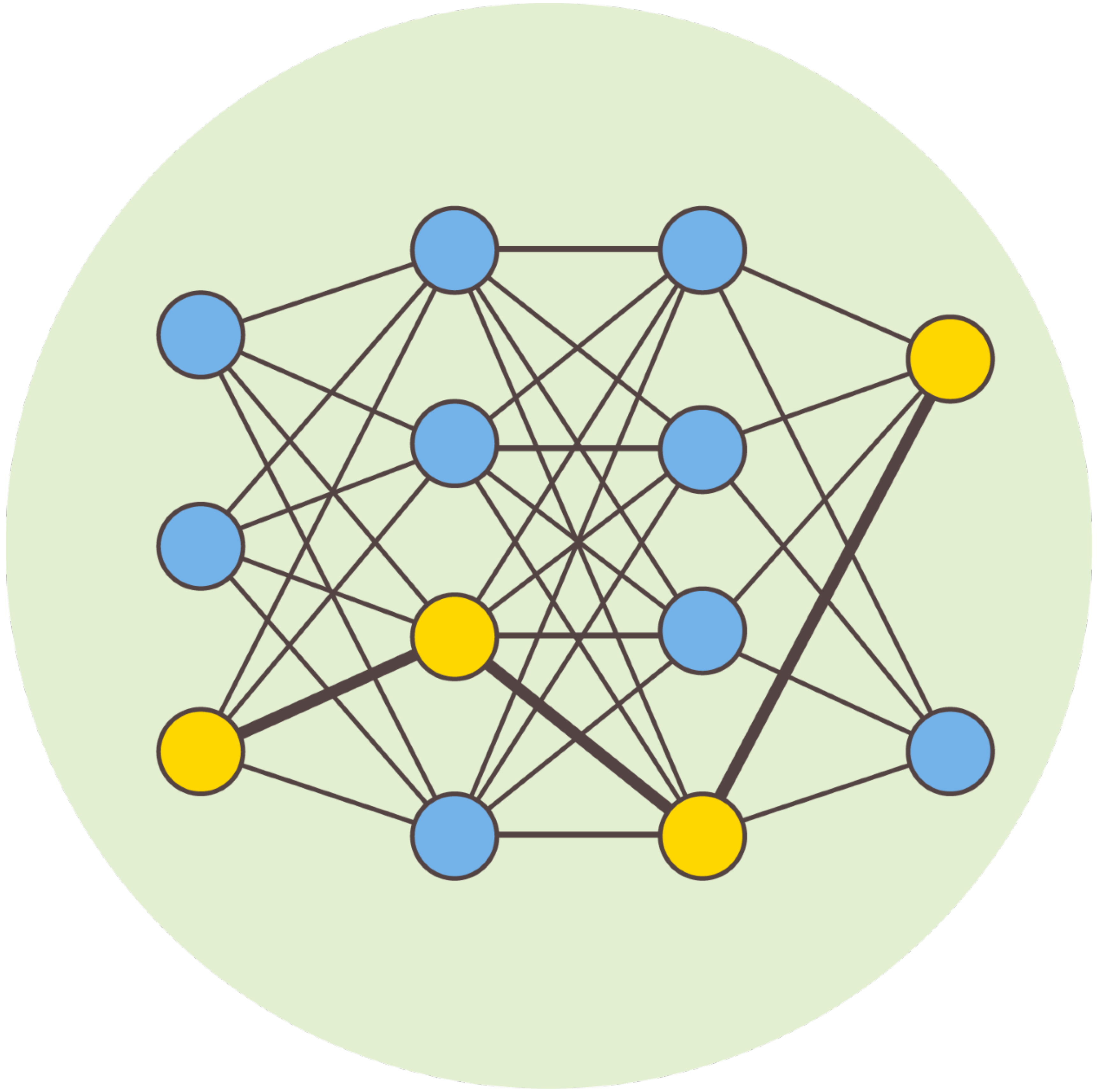
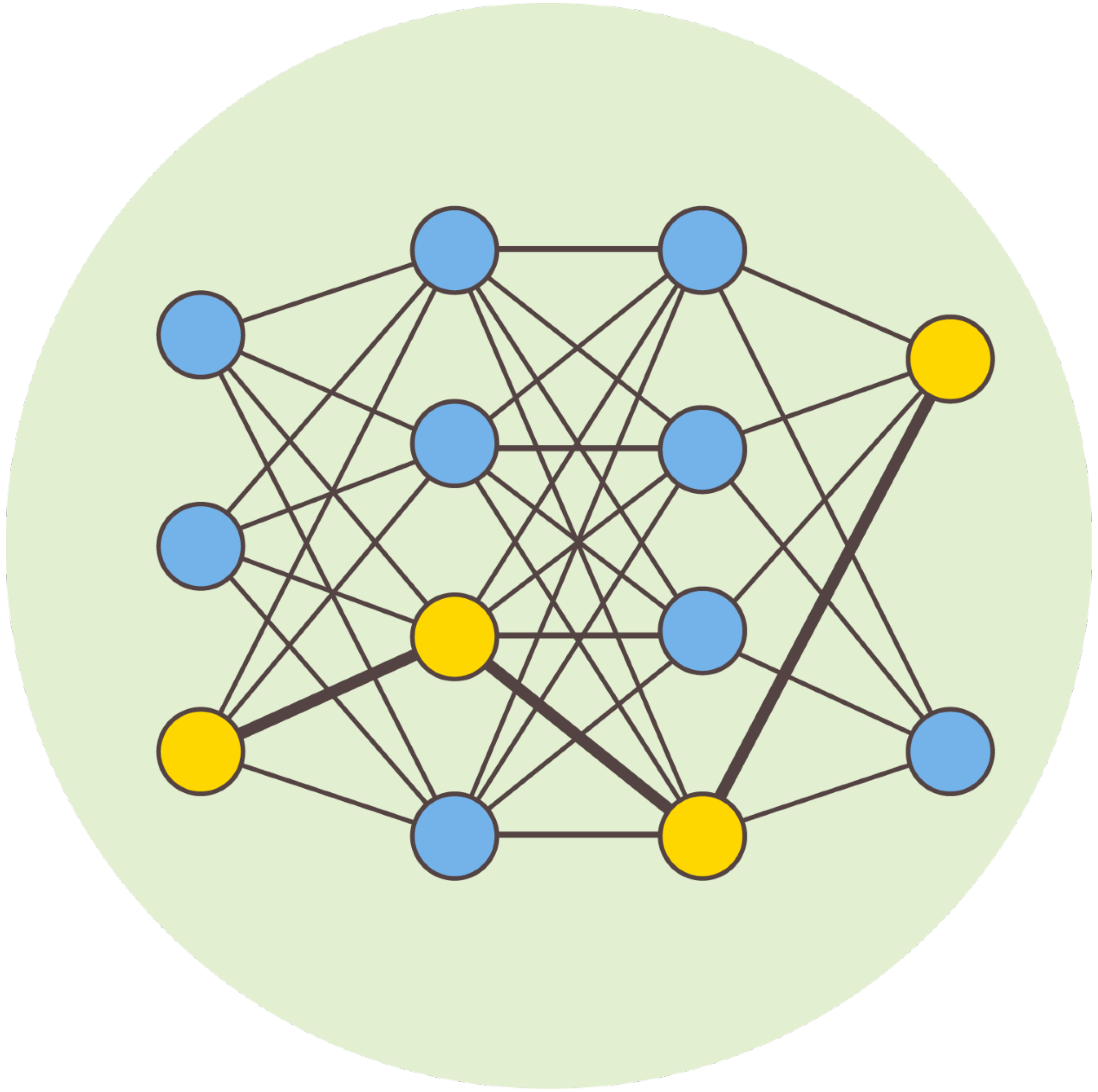


Python 機器學習與深度學習實作

分類與迴歸

Burgeoning Course



Python 機器學習與深度學習實作

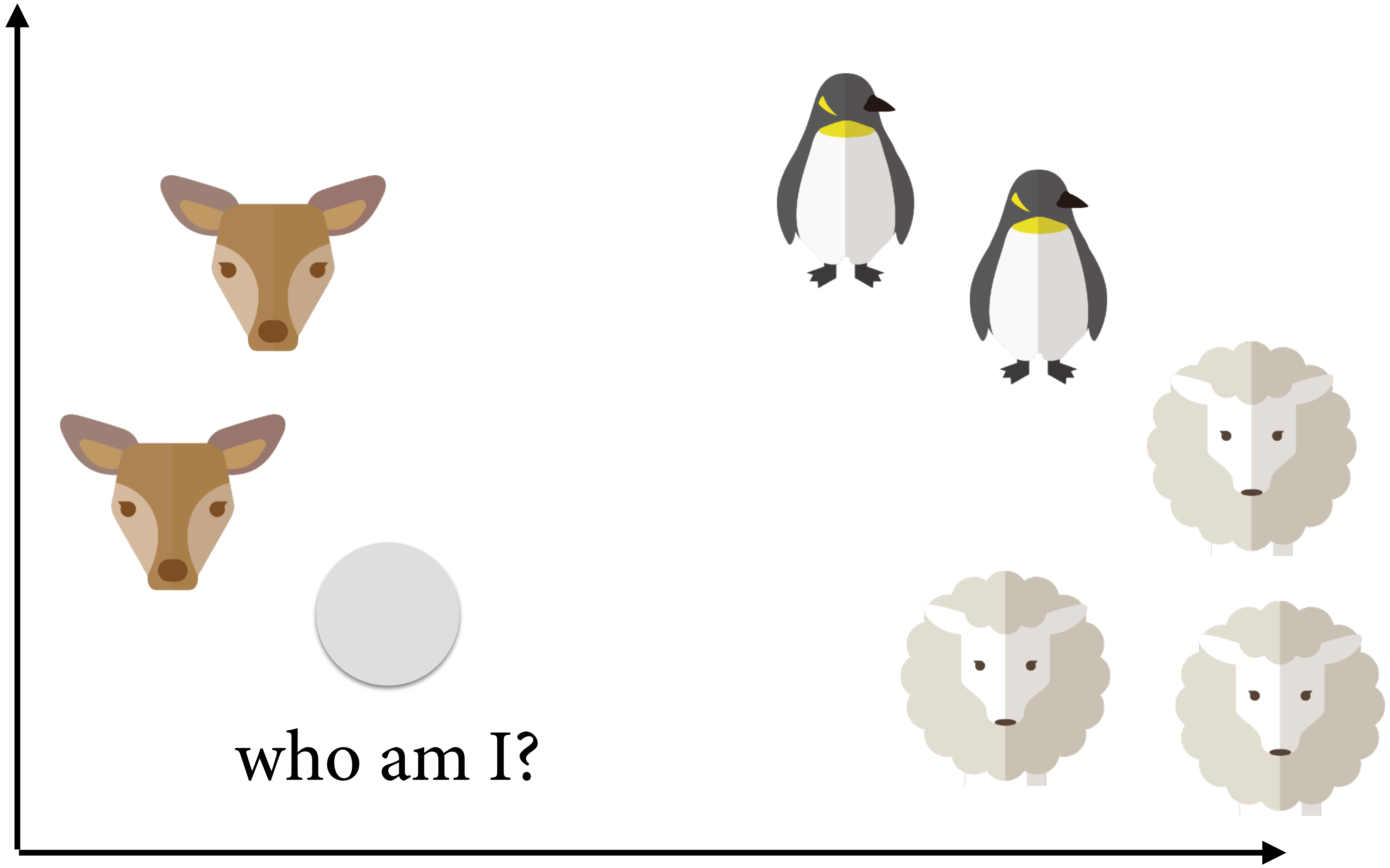
K最近鄰

K最近鄰

- 最簡單、直觀又有效的分類演算法
- 原理：物以類聚
- K：由最近的K個點決定類別

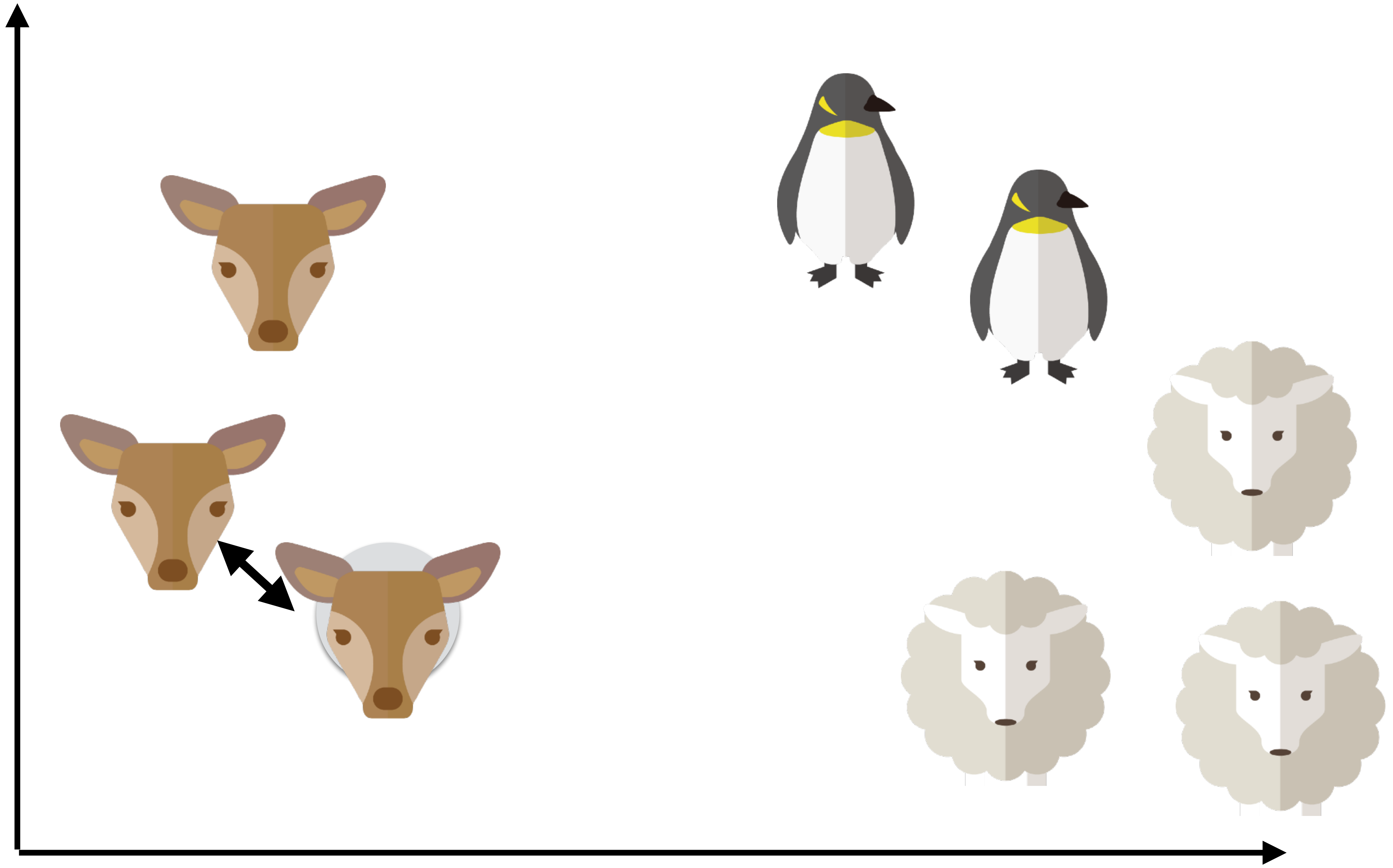
K最近鄰

- K = 1



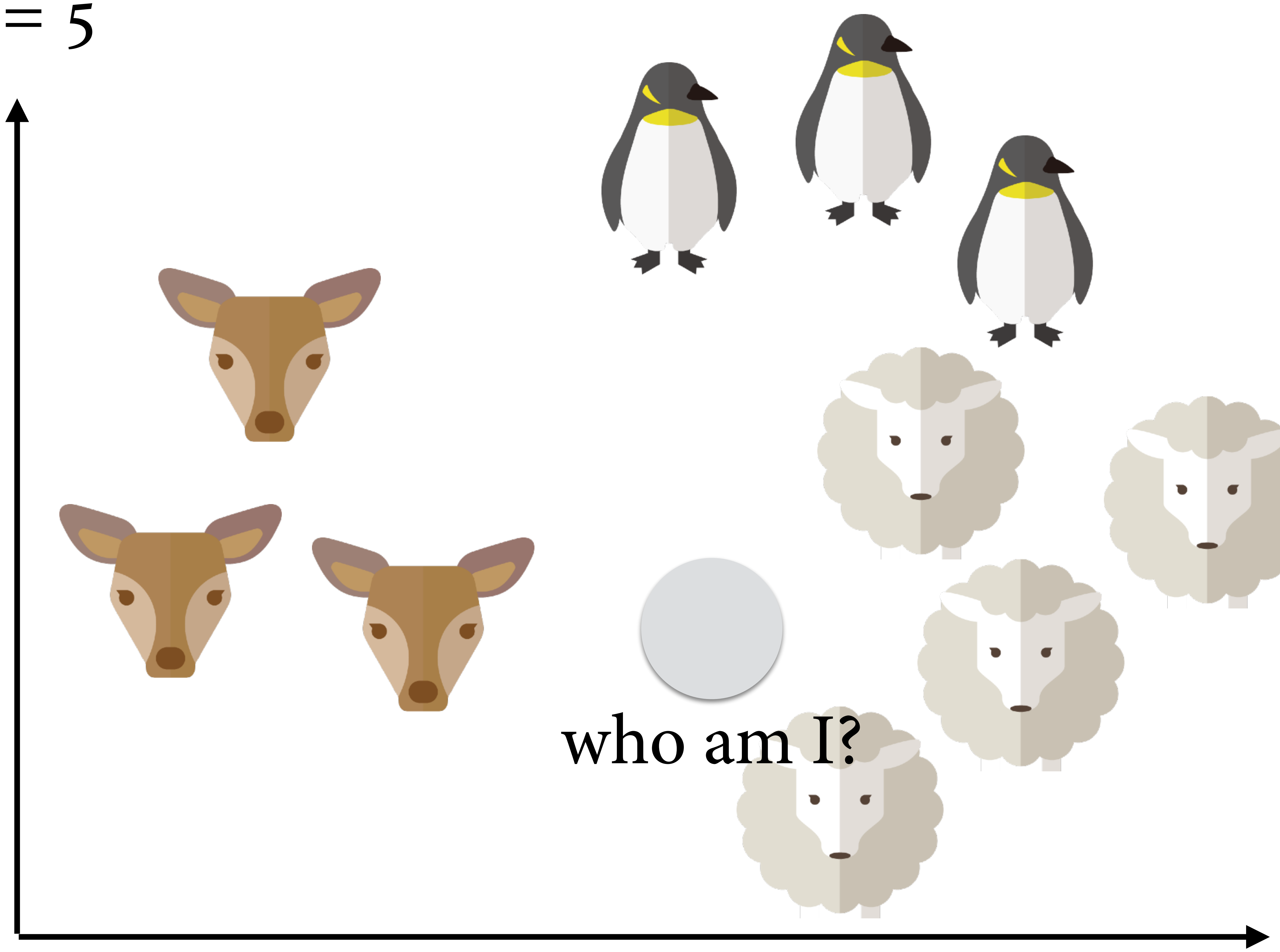
K最近鄰

- K = 1



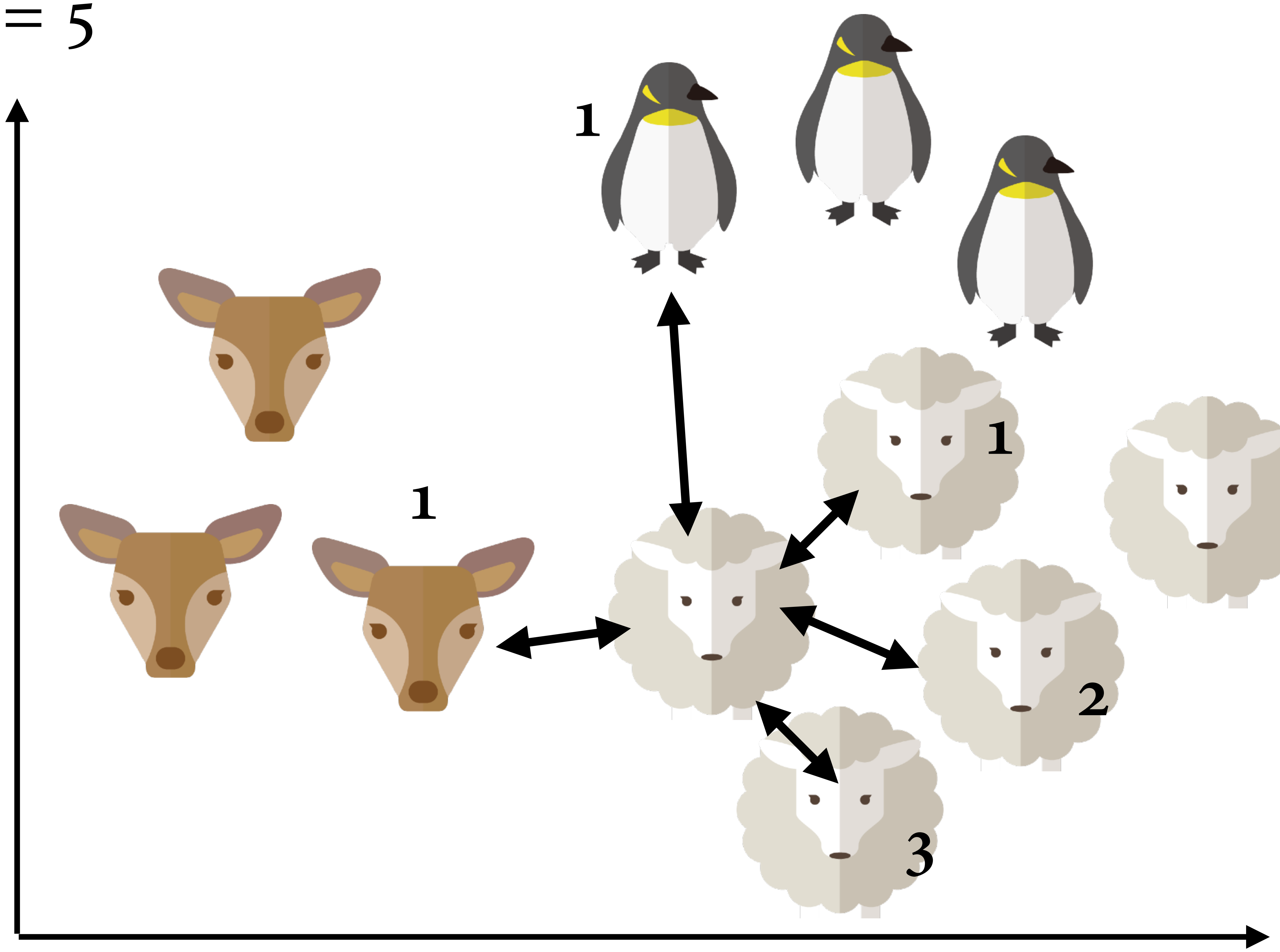
K最近鄰

- K = 5



K最近鄰

- K = 5



優缺點分析

- 缺點：
  - 若分類資料筆數差異大，容易被歸類到資料量較多的類別
  - 每次都要計算與全部點之距離，計算量大耗時
- 優點：
  - 簡單有效
  - 異常值影響不大

KNN 分類

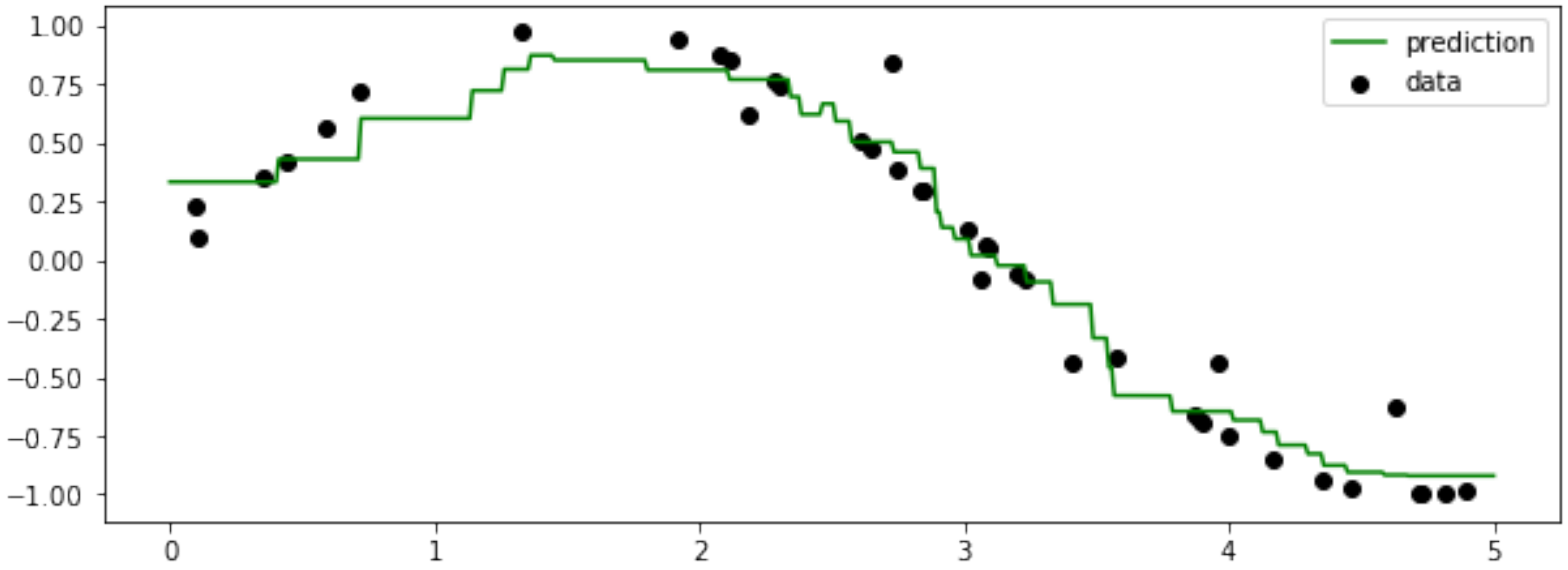
- *class* `sklearn.neighbors.`**`KNeighborsClassifier`**(*n_neighbors=5*, *weights=’uniform’*, *algorithm=’auto’*, *leaf_size=30*, *p=2*, *metric=’minkowski’*, *metric_params=None*, *n_jobs=1*, ***kwargs*)
  - *n_neighbors:* 根據最近多少個鄰居*(K)*來決定類別
  - *weights: ‘uniform’* 最近*k*個鄰居的權重一樣來決定類別，*’distance’* 最近*k*個鄰居的權重根據距離成反比決定類別

- *class* `sklearn.neighbors.`**`RadiusNeighborsClassifier`**(*radius=1.0*, *weights=’uniform’*, *algorithm=’auto’*, *leaf_size=30*, *p=2*, *metric=’minkowski’*, *outlier_label=None*, *metric_params=None*, ***kwargs*)
  - raidus*:* 根據方圓距離內鄰居*(K)*來決定類別
  - *weights: ‘uniform’* 方圓距離內鄰居的權重一樣來決定類別，*’distance’* 方圓距離內鄰居的權重根據距離成反比決定類別

KNN 迴歸

- *class* `sklearn.neighbors.`**`KNeighborsRegressor`**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=1*, ***kwargs)*

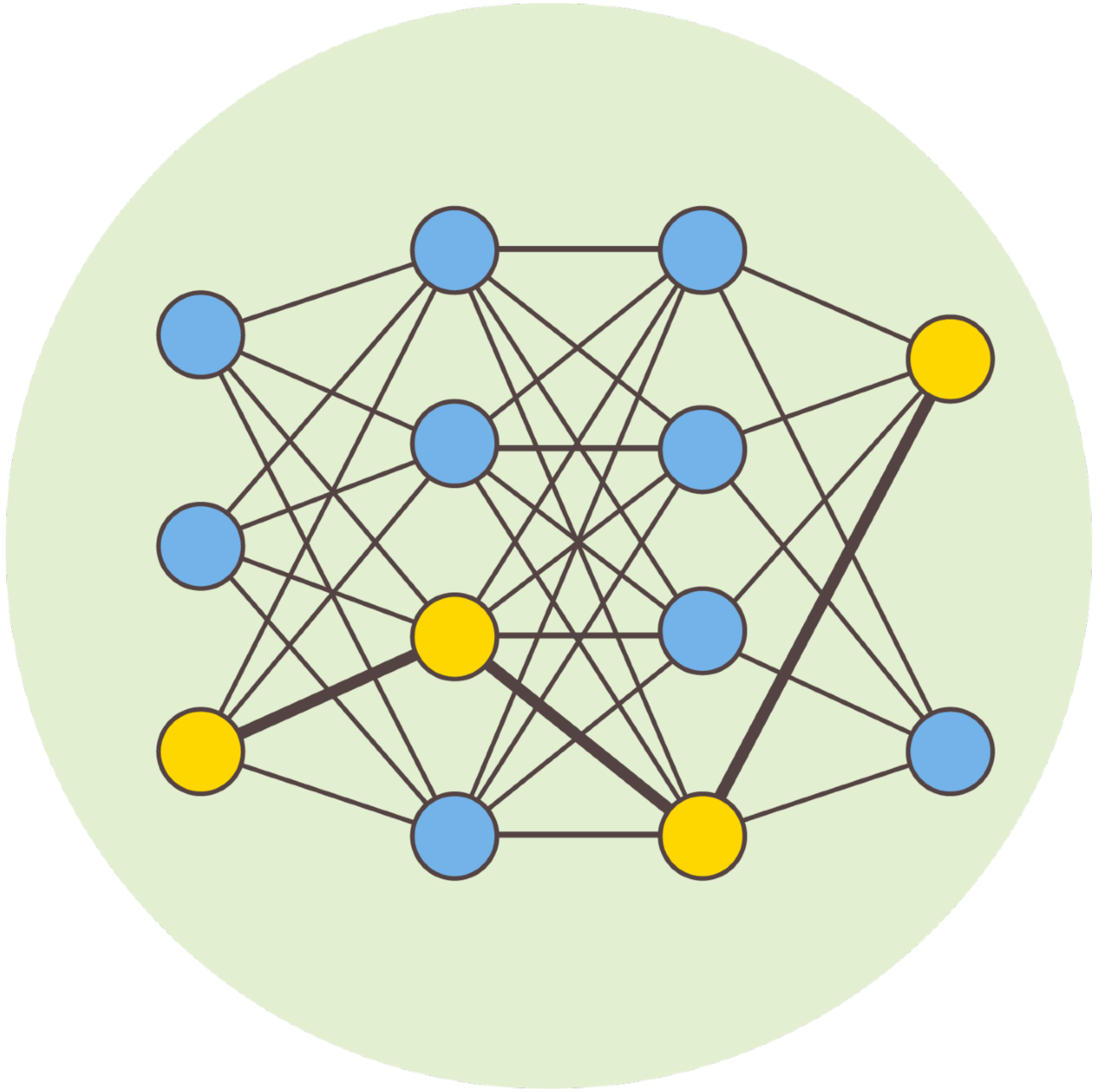
    - *n_neighbors:* 根據最近多少個鄰居(*K*)決定數值
    - *weights: 'uniform'* 最近*k*個鄰居的權重一樣來做平均，*'distance'* 最近*k*個鄰居的權重根據距離成反比做加權平均



KNN 迴歸

- *class* `sklearn.neighbors.RadiusNeighborsRegressor`(*radius=1.0*, *weights=’uniform’*, *algorithm=’auto’*, *leaf_size=30*, *p=2*, *metric=’minkowski’*, *metric_params=None*, ***kwargs)*

  - *radius:* 根據方圓多少距離以內的鄰居計算數值
  - *weights: ‘uniform’* 方圓內的權重一樣來做平均，*’distance’* 方圓內鄰居的權重根據距離成反比做加權平均

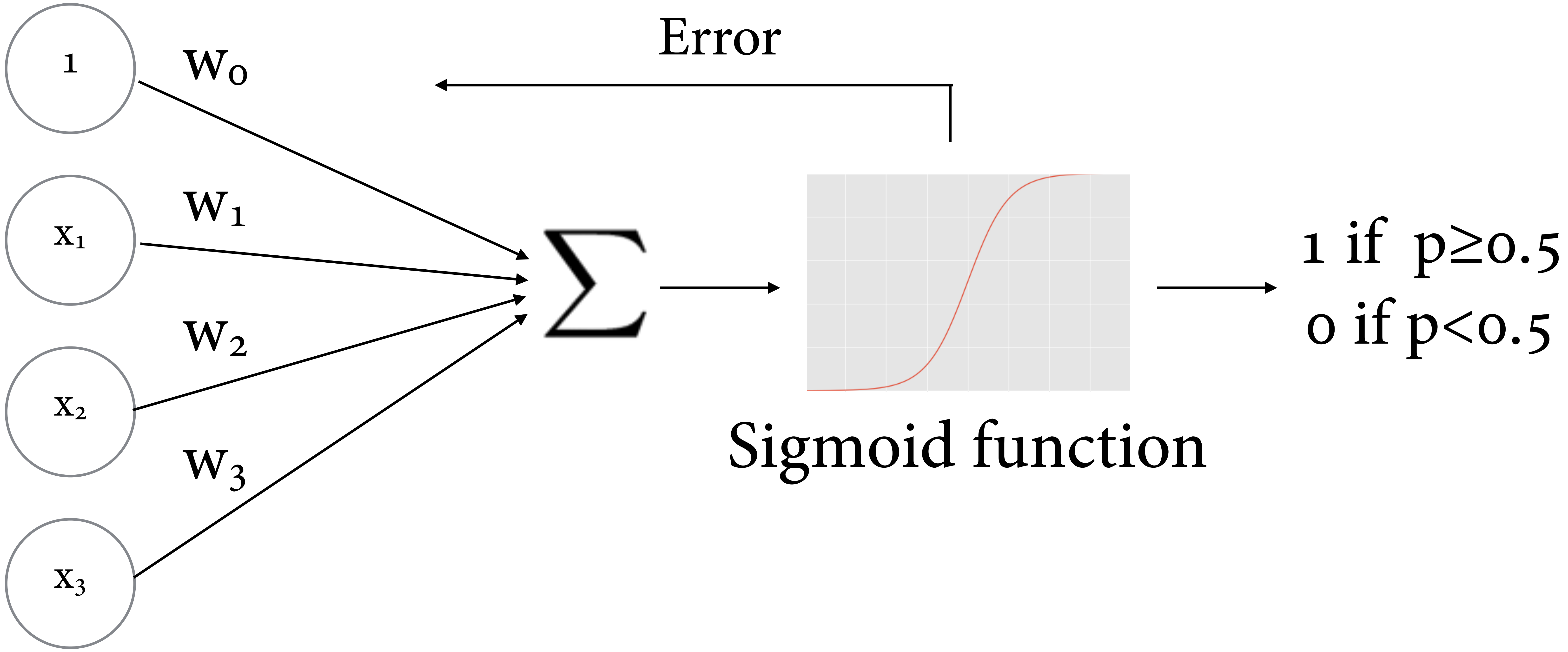


Python 機器學習與深度學習實作

羅吉斯迴歸

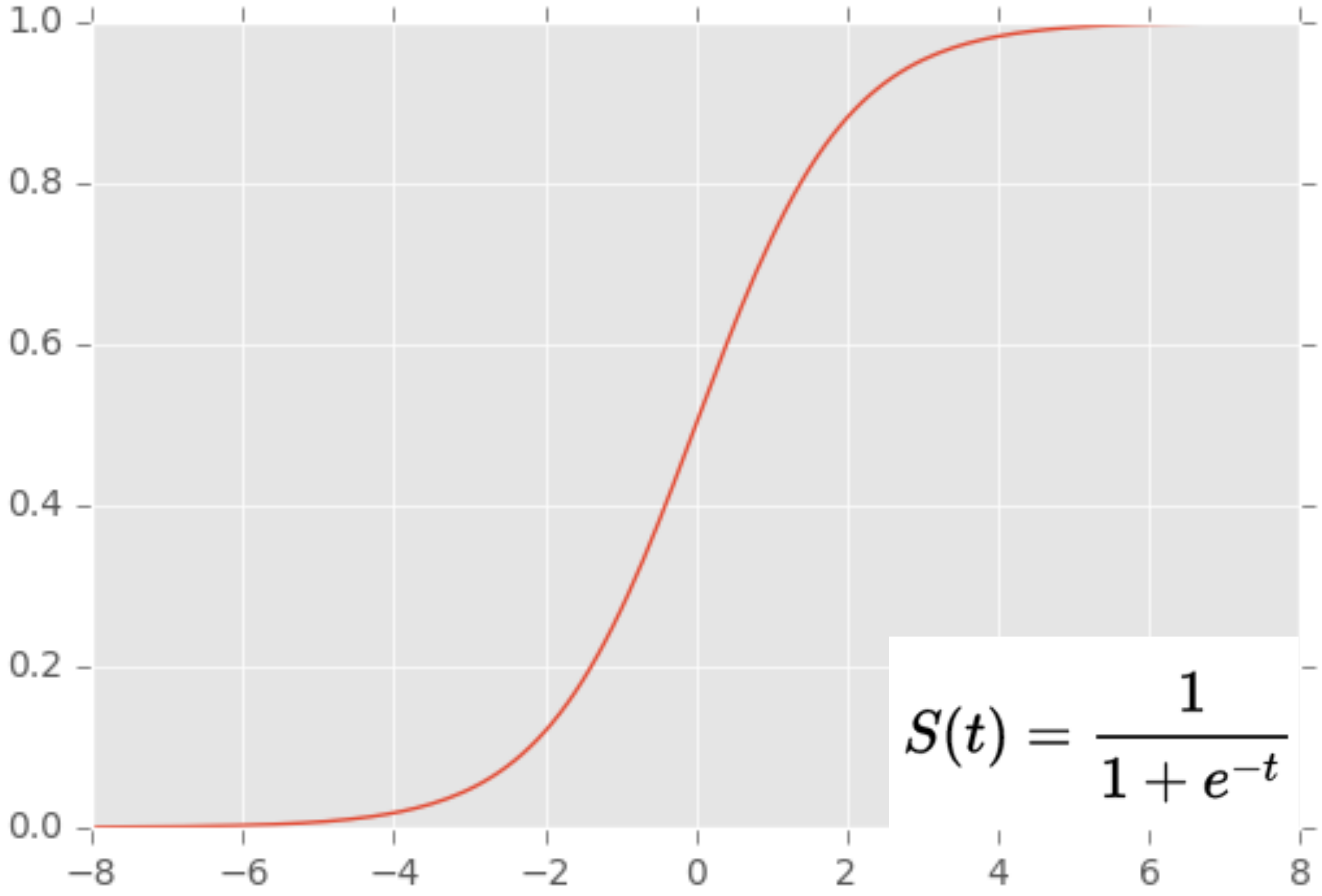
羅吉斯迴歸（Logistic Regression）

- 雖然名為迴歸，但常用於分類（二元或多類別）



Logistic Function

- Logistic function / Sigmoid function

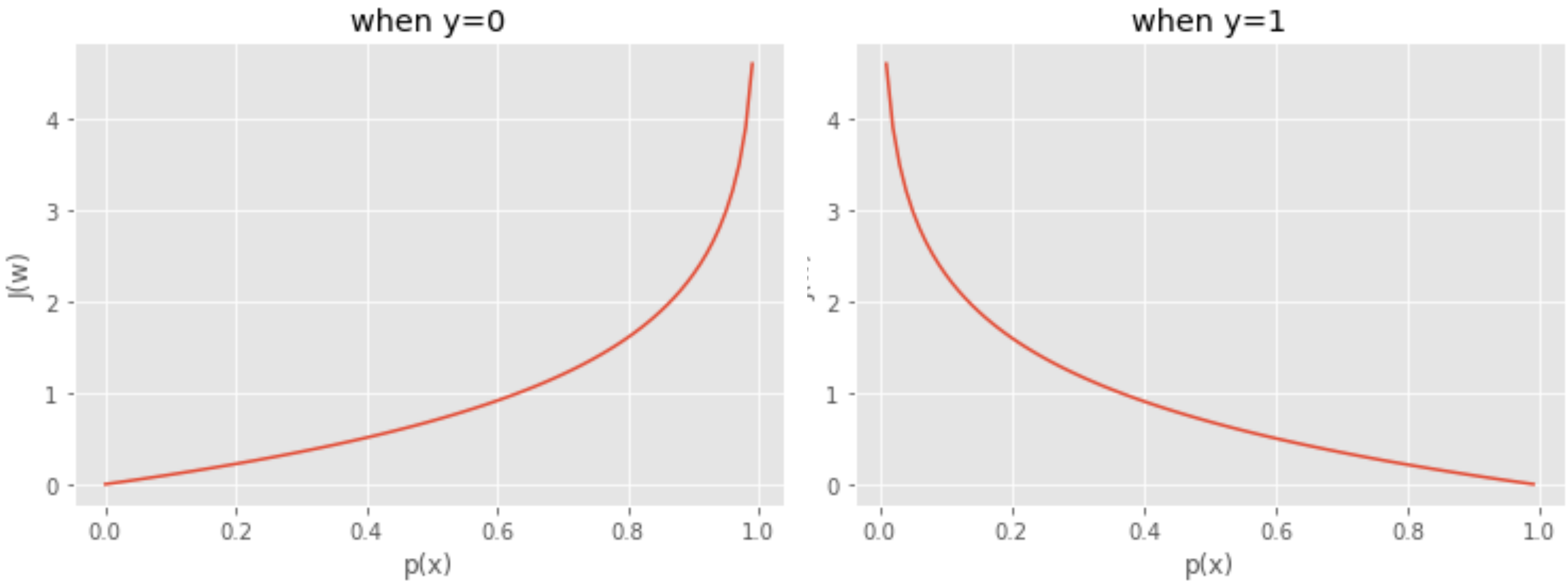


$$p(x) = \frac{1}{1+e^{-w^T x}}$$

Cost Function

$$J(w) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)}\log p(x^{(i)}) + (1-y^{(i)})\log(1-p(x^{(i)}))\right)$$

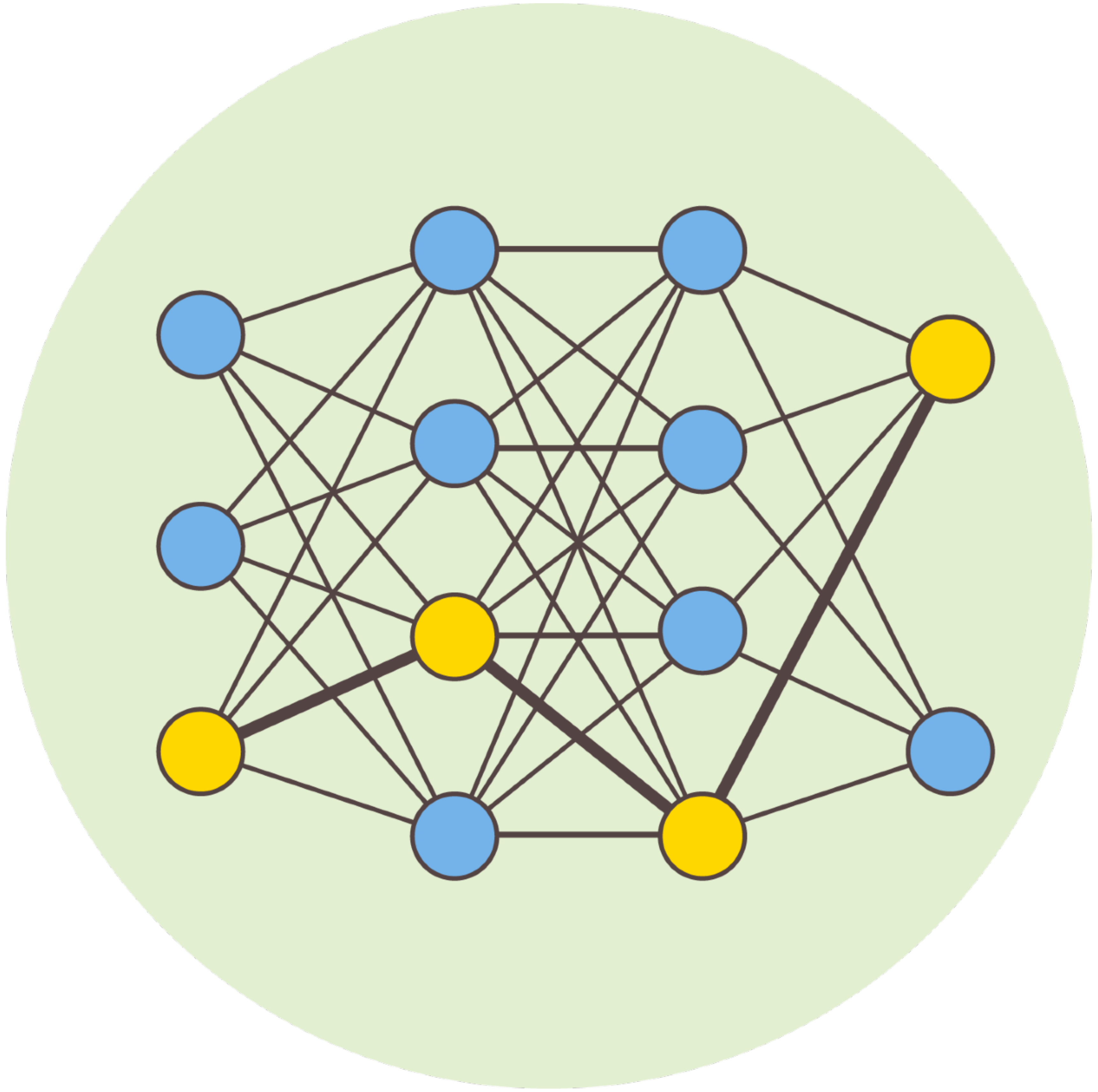
$$J(w) = -\log(1-p(x)) \qquad J(w) = -\log(p(x))$$



多類別分類

- 多類別分類，使用One-vs-Rest（OvR）
  - e.g. A, B, C三類，分別計算是A的機率、是B的機率、是C的機率

```
array([[ 0.009,  0.401,  0.59 ],
       [ 0.008,  0.436,  0.555],
       [ 0.009,  0.585,  0.406],
       [ 0.76 ,  0.137,  0.103],
       [ 0.007,  0.505,  0.488],
       [ 0.   ,  0.399,  0.601],
       [ 0.018,  0.496,  0.487],
       [ 0.004,  0.419,  0.577],
       [ 0.864,  0.088,  0.048],
```

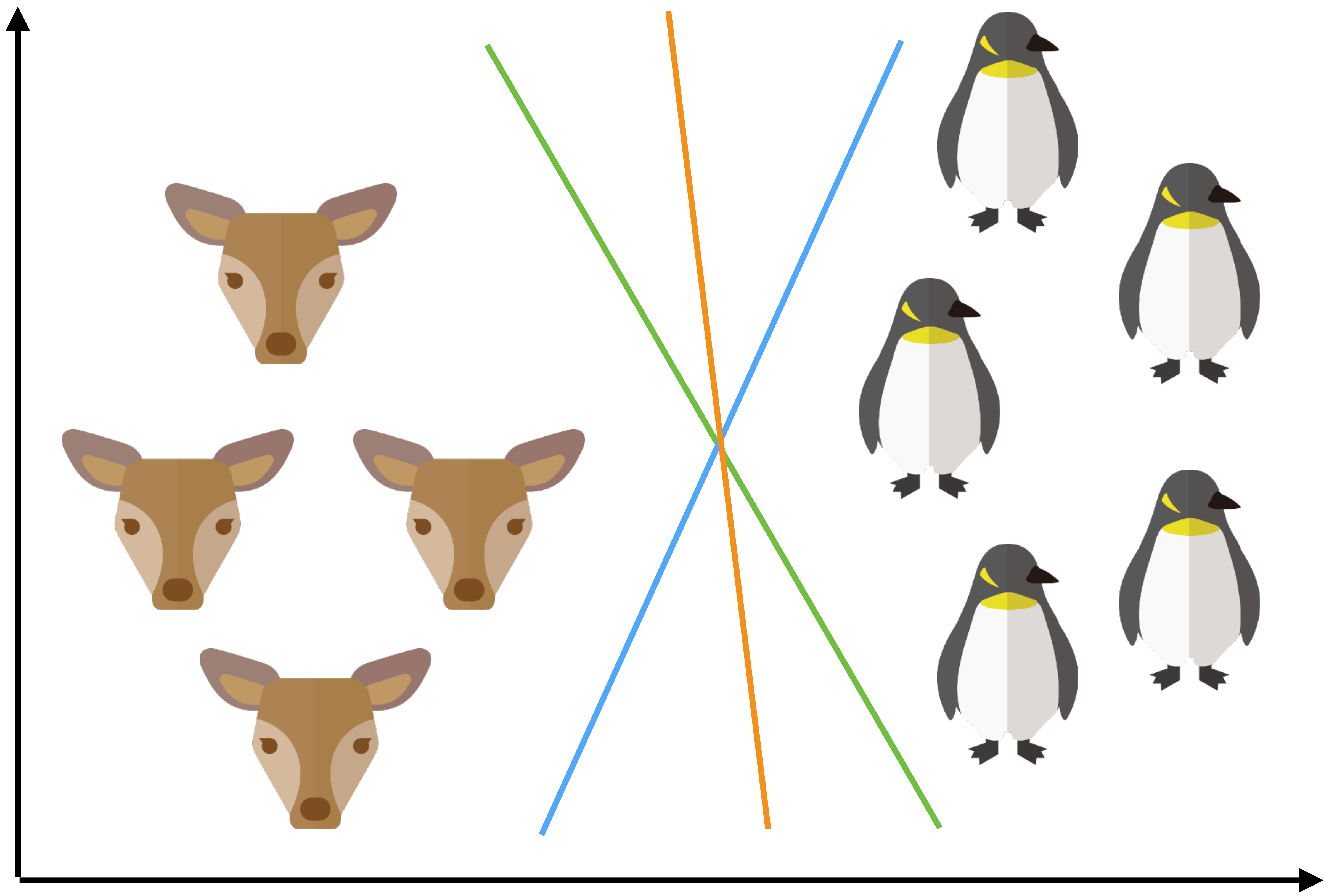


Python 機器學習與深度學習實作

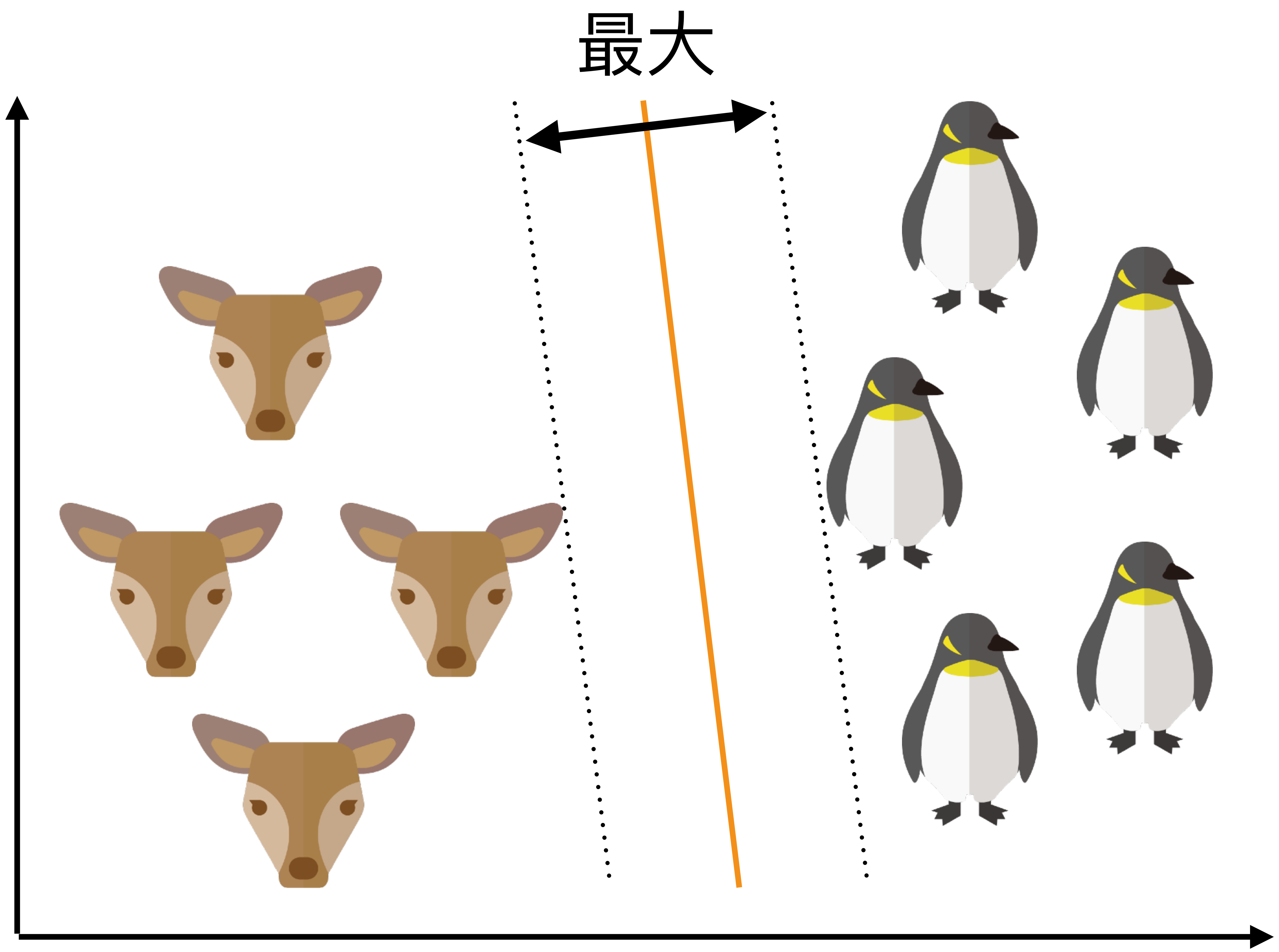
支持向量機與決策邊界

Burgeoning Course

哪一條分類線最好？

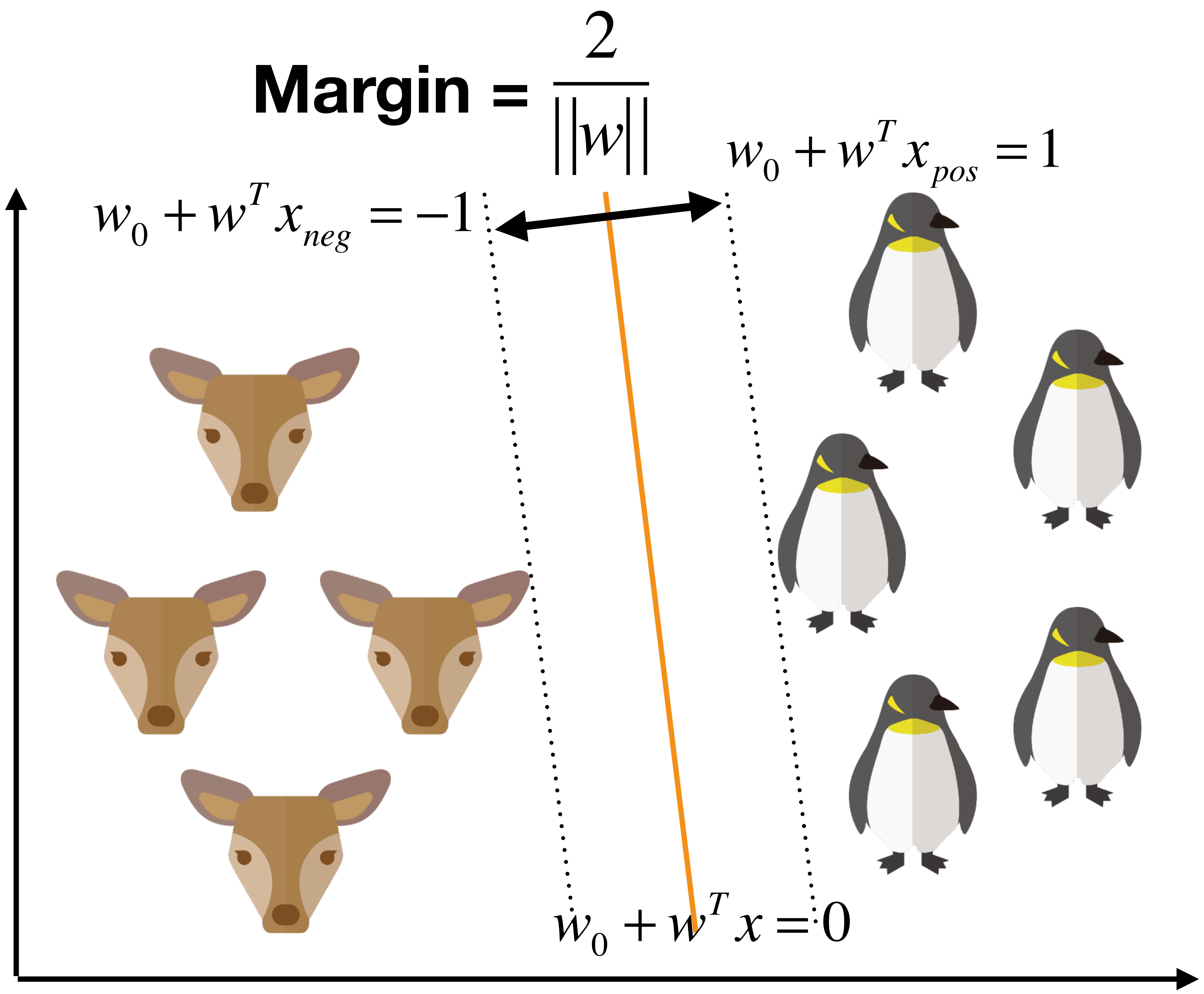


SVM says…this one



- 目標：最大化邊界 (margin)
- 直觀理由：最大化的邊界，通常可以獲得較小的誤差

SVM



定義：$||w|| = \sqrt{\sum_{i=1}^{m} w^{(i)2}}$

$$w^T(x_{pos} - x_{neg}) = 2$$

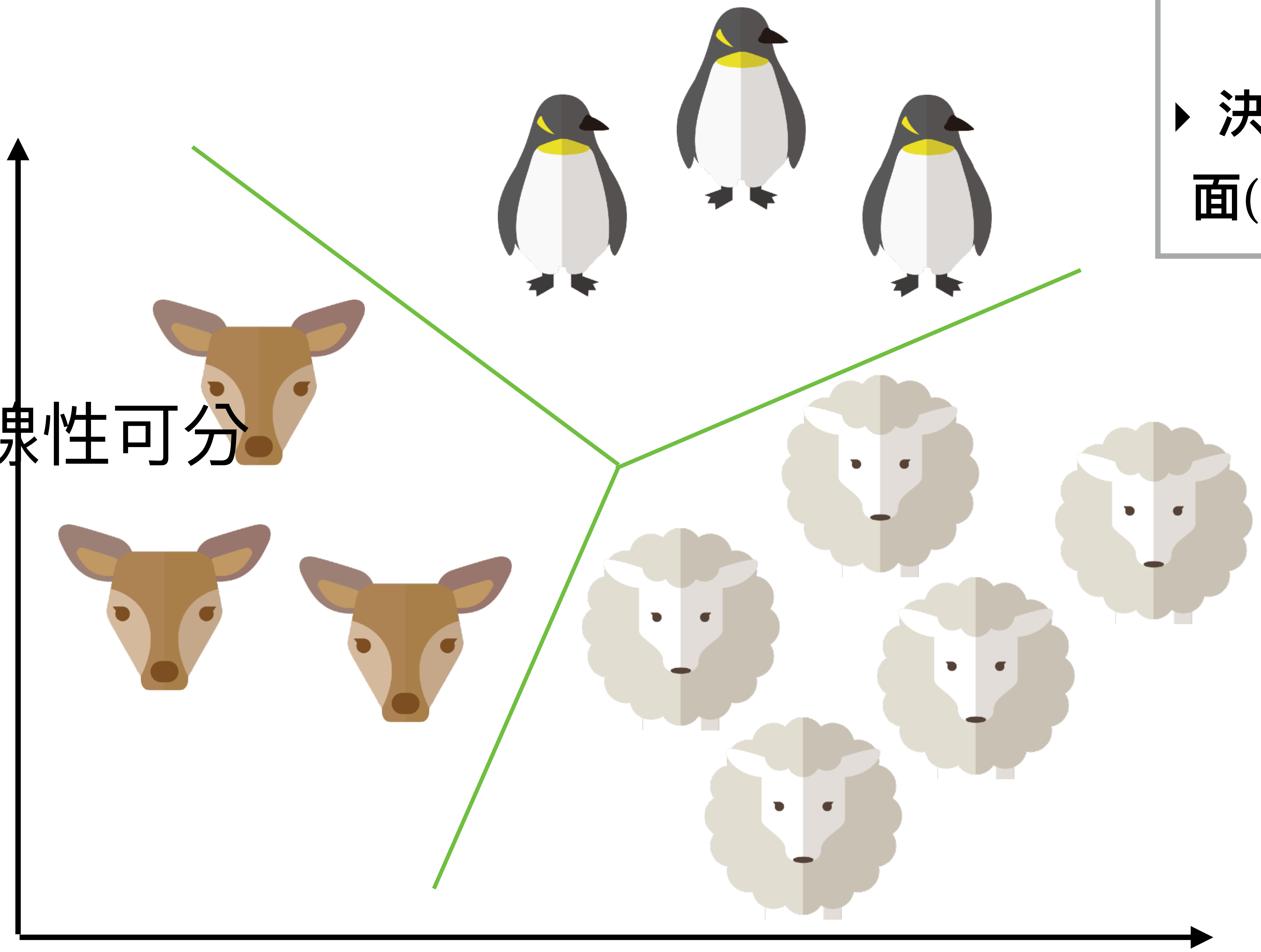
$$\frac{w^T(x_{pos} - x_{neg})}{||w||} = \frac{2}{||w||}$$

決策邊界 (Decision Boundary)

Notes

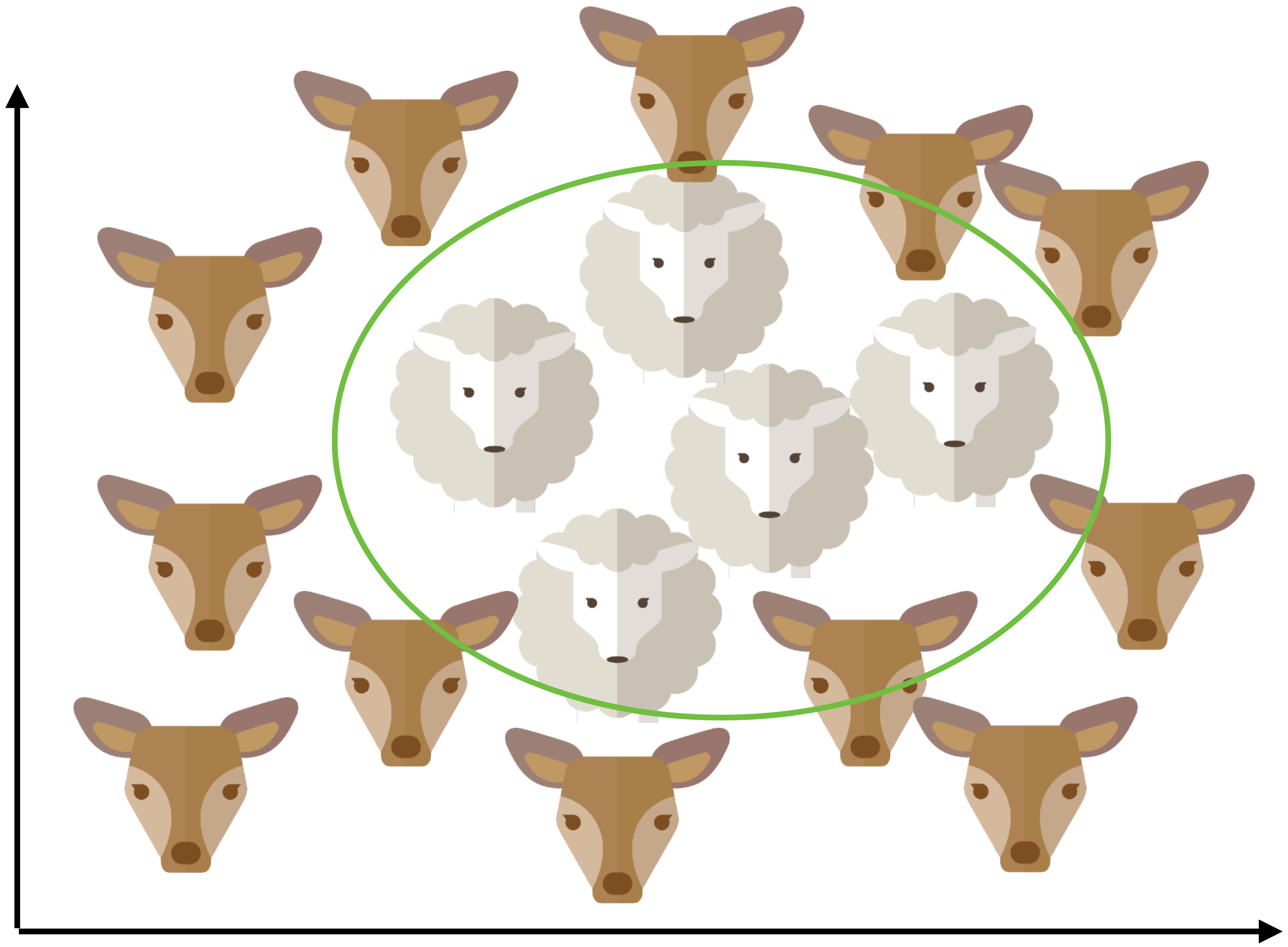
- 決策邊界(Decision Boundary) 又稱為分離超平面(separating hyperplane)

- 線性可分



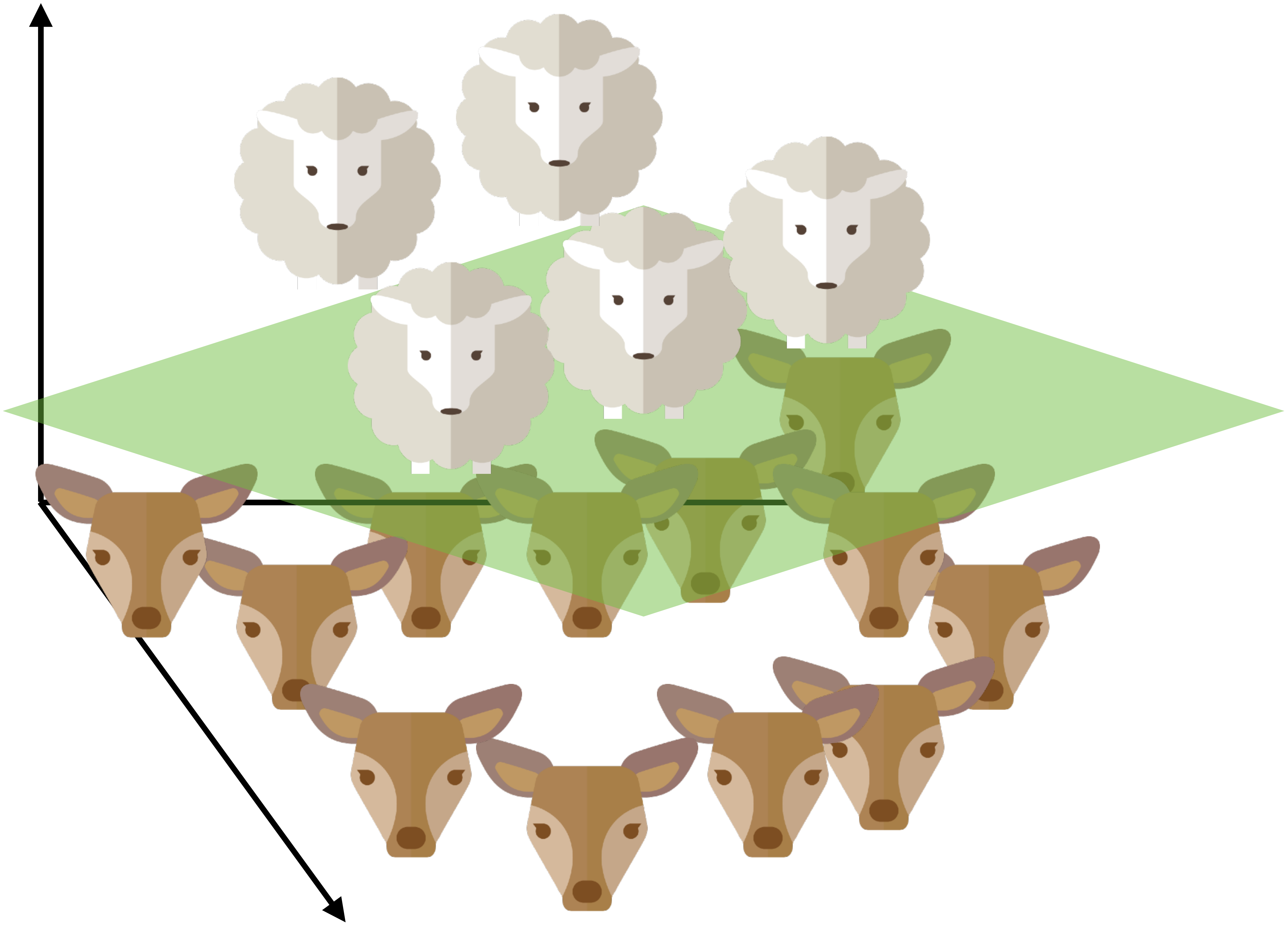
SVM問題

- 線性不可分 => 非線性邊界



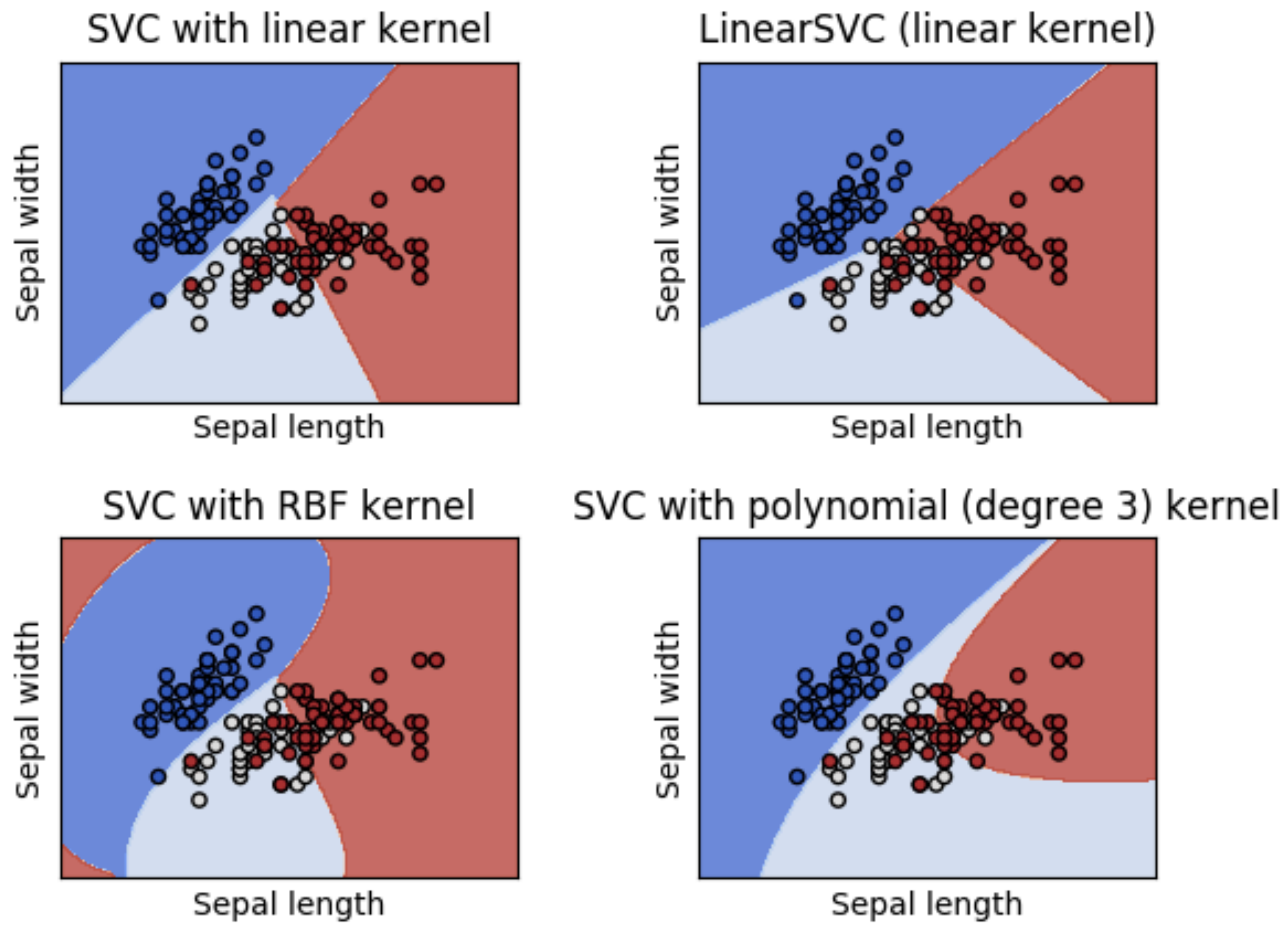
使用核(Kernel)技巧

- 轉換到高維空間



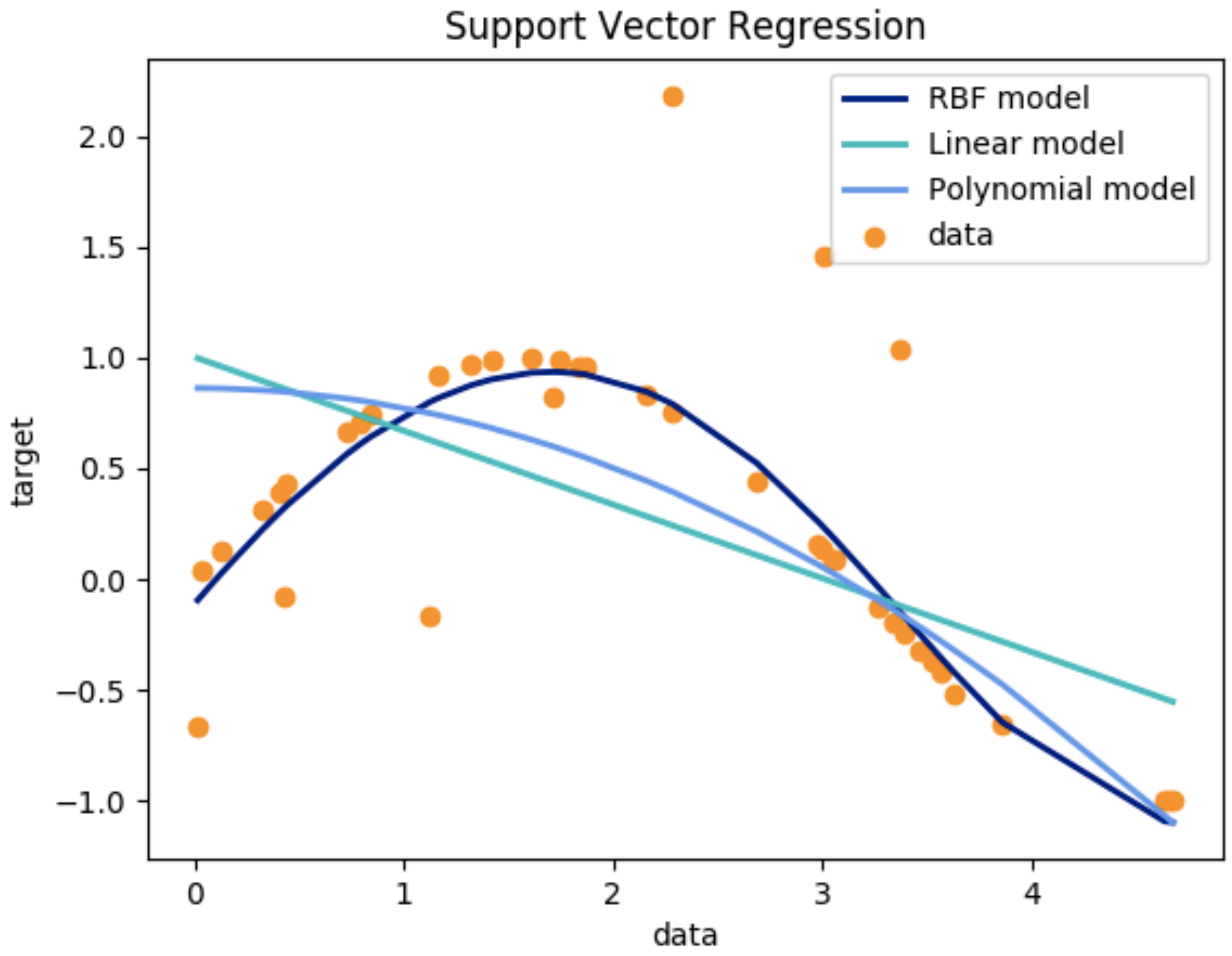
SVM 分類

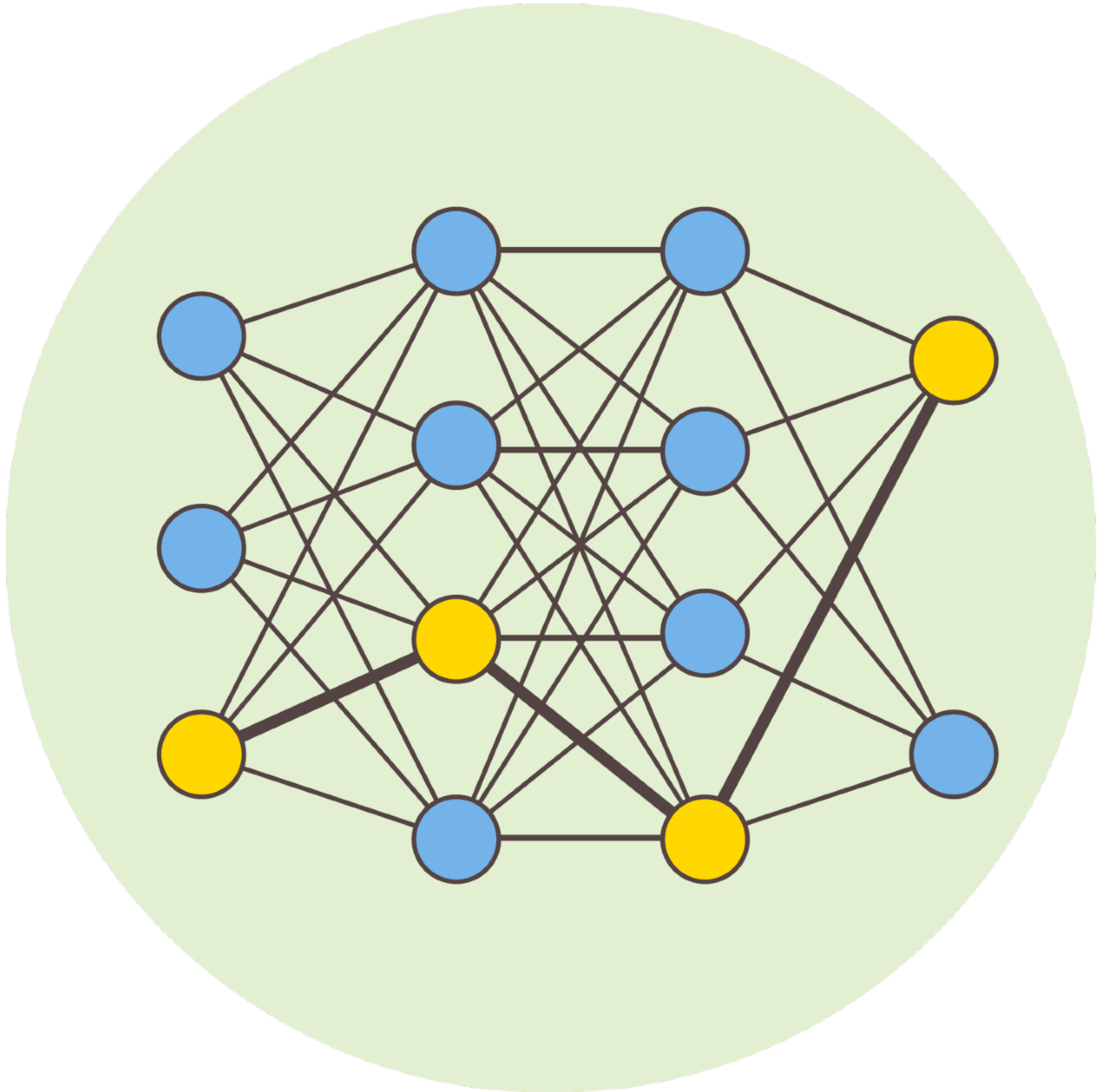
- *class* `sklearn.svm.SVC`(*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *random_state=None*)
  - C: 對於錯誤分類的懲罰
  - kernel: 'rbf' 徑向基函數核(Radius basis function kernel)，'linear' 線性，'ploy' 多項式(非線性) with degree



SVM 迴歸

- *class* `sklearn.svm.SVR`(*kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1*)
  - C: 對於錯誤分類的懲罰
  - kernel: 'rbf' 徑向基函數核(Radius basis function kernel)，'linear' 線性，'ploy' 多項式(非線性) with degree





Python 機器學習與深度學習實作

決策樹與特徵選擇

如何判斷好的特徵？

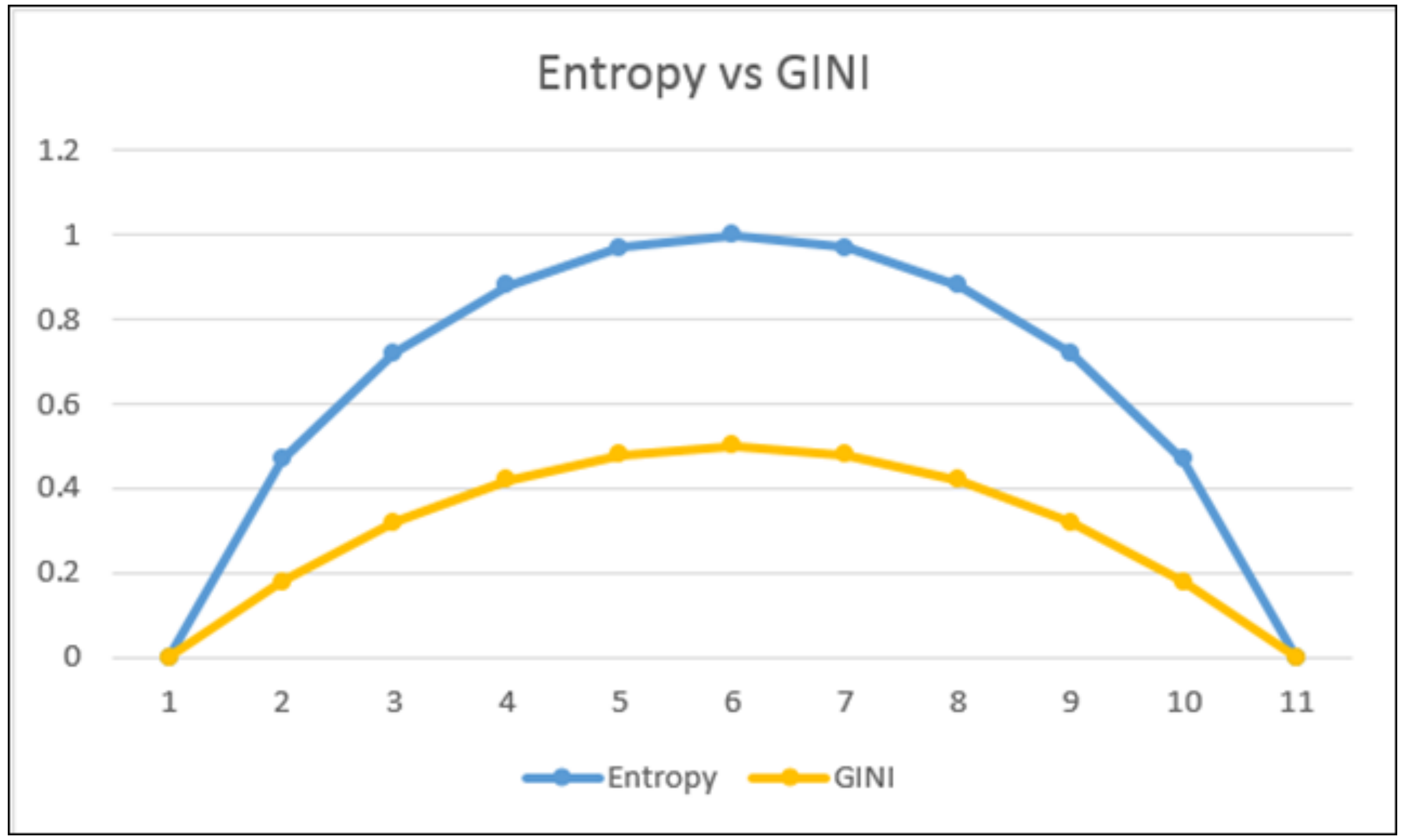
- Domain Knowledge / Know-How

- 特徵是否能將資料有效區隔為不同群體？切分後的子群體純度多高？（純度越高越好）

  - e.g. 蘑菇的氣味、顏色較形狀能區隔出有毒或無毒蘑菇

度量

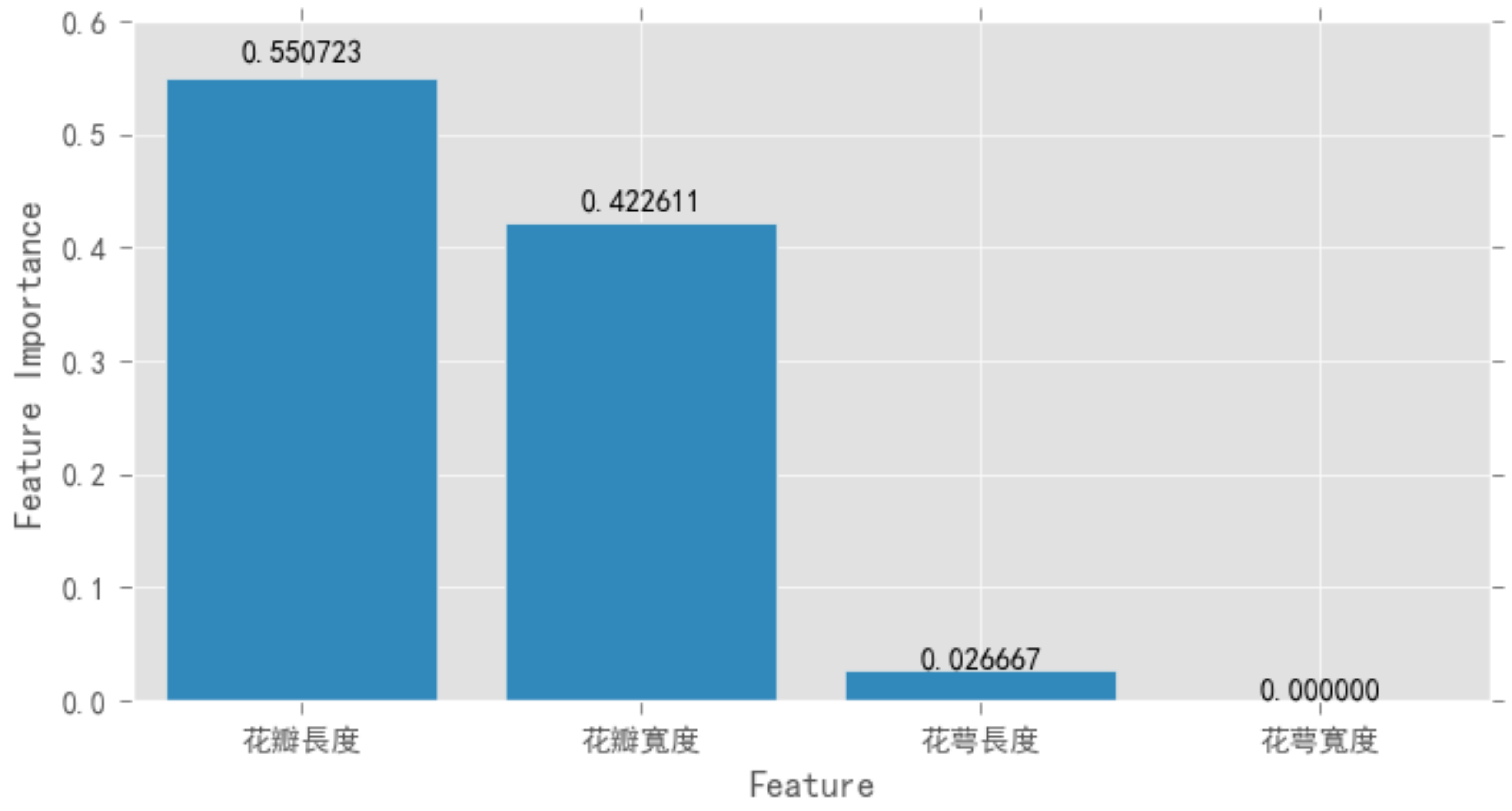
- 熵（Entropy, I_E）
  - $I_E = - \sum_{i} p_j * \log_2 p_j$
- 吉尼不純度（Gini Impurity, I_G）
  - $I_G = 1 - \sum_{i} p_j^2$
- 實務上效果差不多
  - e.g. 一個群體包含20%毒菇、80%非毒菇
  - Entropy = - 0.2 * log₂(0.2) - 0.8 * log₂(0.8) = 0.72
  - Gini = 1 - (0.2² + 0.8²) = 0.32



(Source from: https://abhyast.wordpress.com/)

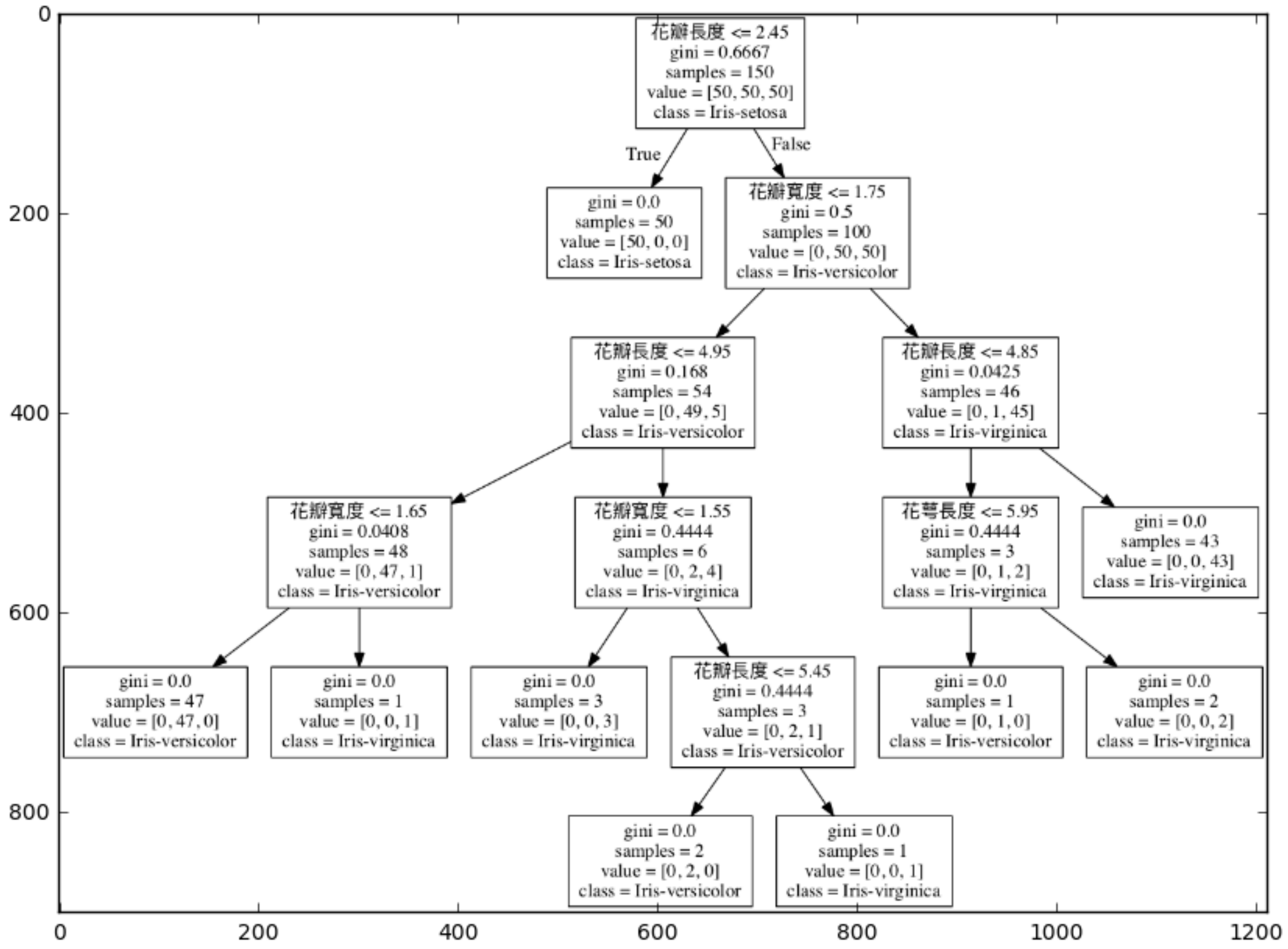
資訊增益 (Information Gain, IG)

- $IG = I_{E \text{ or } G}(\text{parent}) - \sum_{j} p(c_j) * I_{E \text{ or } G}(\text{children})$
- 決策分類樹演算法依據，節點產生的IG越高越好
- 判斷特徵重要性



決策分類樹與迴歸樹

- 能將決策判斷邏輯視覺化，最易理解、具說服力的演算法



Notes

- 決策分類樹
  - sklearn.tree.DecisionTreeClassifier
- 迴歸樹
  - sklearn.tree.DecisionTreeRegressor

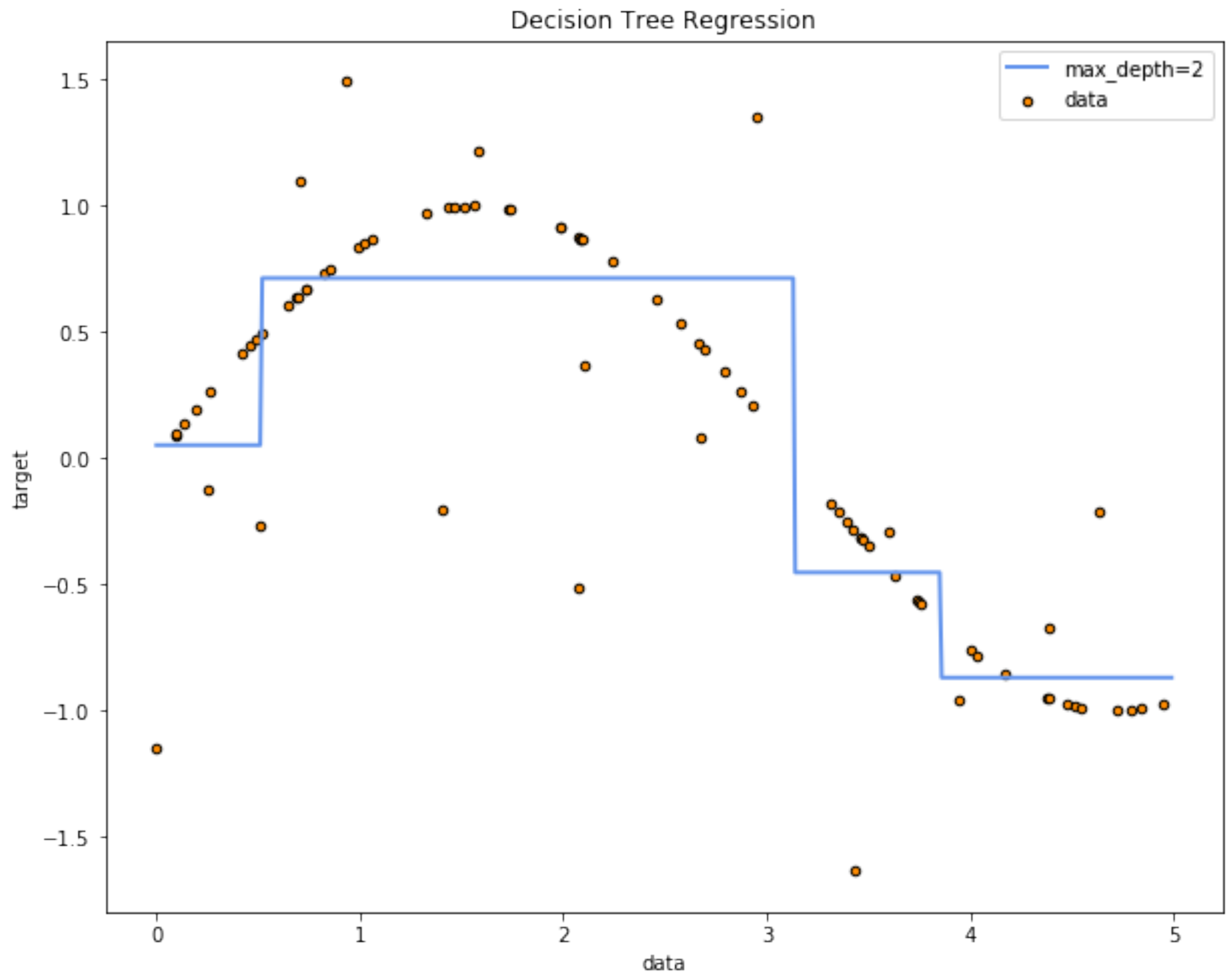
決策樹演算法

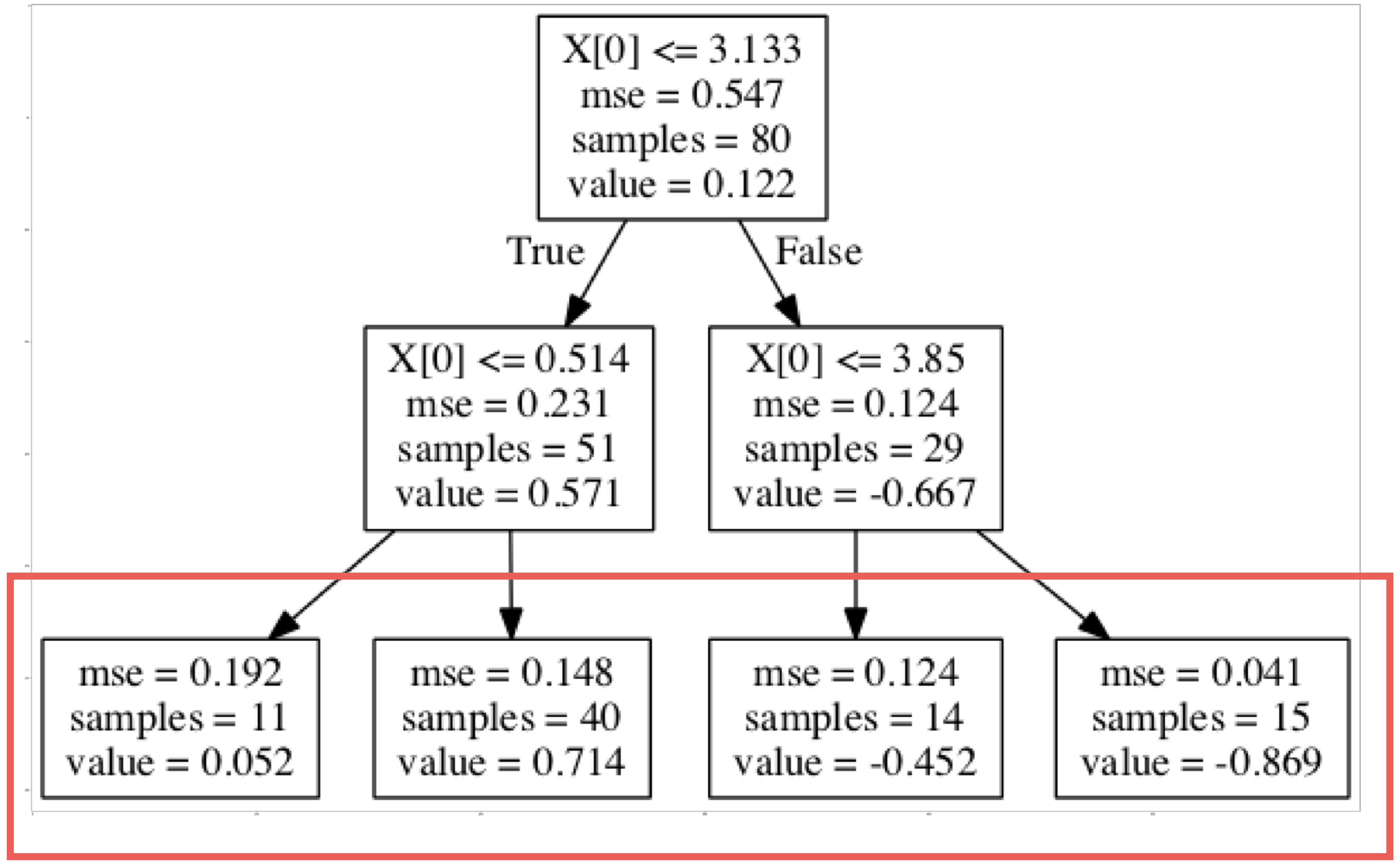
- 決策分類樹：(不能做迴歸)
  - ID3 (Iterative Dichotomiser 3) (只接受離散數值)
  - C4.5 (改進ID3) (接受連續值)
  - C5.0 (商用C4.5)
- CART (Classification and Regression Trees)

Notes

- Scikie-Learn 採用最佳化版本的CART演算法

迴歸樹


Decision Tree Regression
max_depth=2
data
target
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5


X[0] <= 3.133
mse = 0.547
samples = 80
value = 0.122
True
False
X[0] <= 0.514
mse = 0.231
samples = 51
value = 0.571
X[0] <= 3.85
mse = 0.124
samples = 29
value = -0.667
mse = 0.192
samples = 11
value = 0.052
mse = 0.148
samples = 40
value = 0.714
mse = 0.124
samples = 14
value = -0.452
mse = 0.041
samples = 15
value = -0.869

Overfitting問題

- scikit-learn 目前不提供修剪(pruning)演算法
- 建議直接設定決策樹生長最大深度：max_depth

決策分類樹(DecisionTreeClassifier)

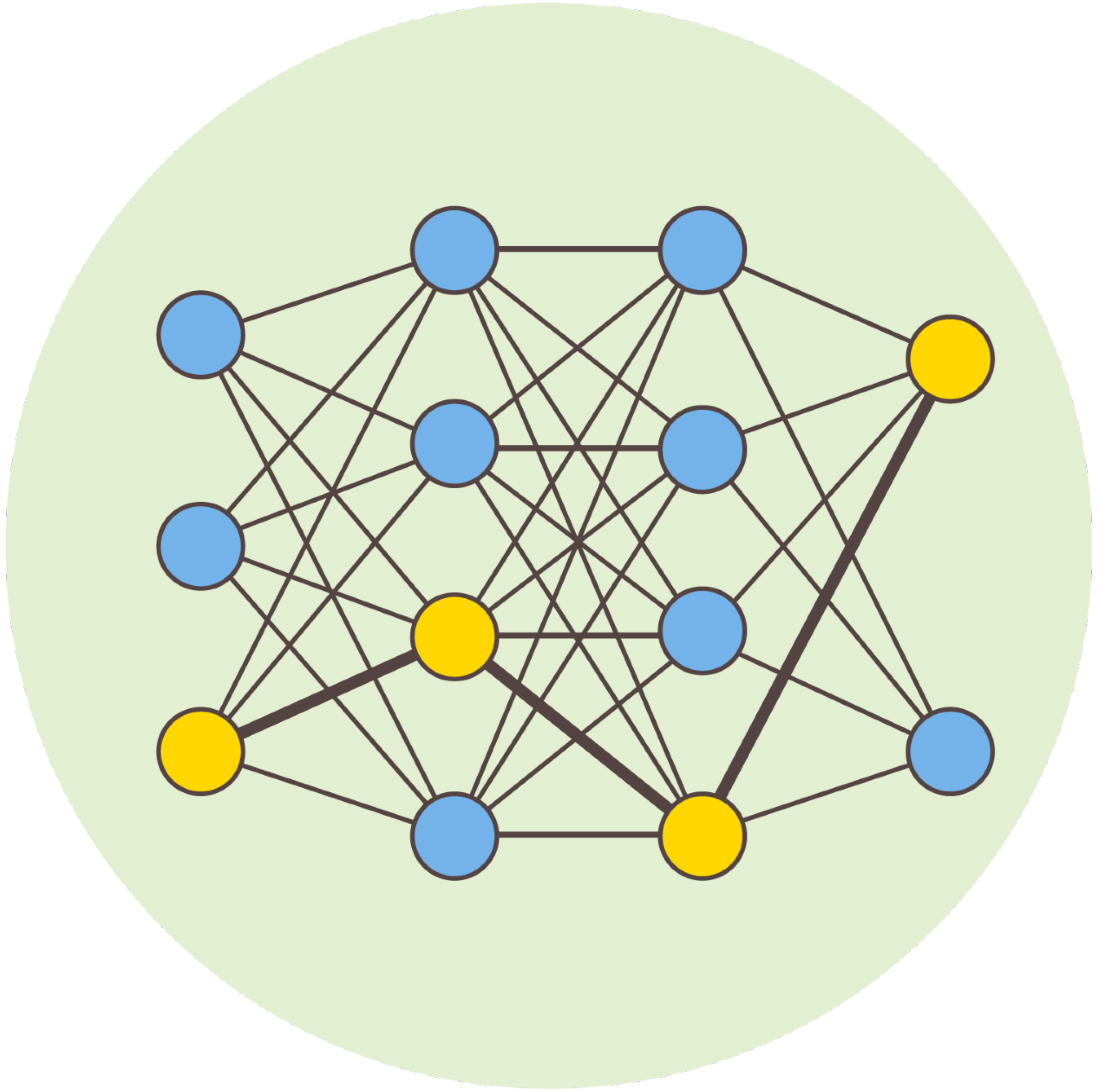
- *class* `sklearn.tree.`**`DecisionTreeClassifier`**(*criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *class_weight=None*, *presort=False*)
  - criterion: 'gini' or 'entropy'
  - *max_depth:* 樹的最大深度

- *attributes:*
  - *feature_importances_:* 特徵重要性
  - *features_names:* 特徵名稱(注意順序)
  - *class_names:* 分類名稱(注意順序)

迴歸樹(Regression Tree)

- *class* `sklearn.tree.DecisionTreeRegressor`(*criterion='**mse**', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort=False*)
  - max_depth: 樹的最大深度

- attributes
  - feature_importances_: 特徵重要性
  - *features_names: 特徵名稱(注意順序)*
  - *class_names: 分類名稱(注意順序)*



Python 機器學習與深度學習實作

單純貝式分類器

Burgeoning Course

條件機率Case

- 假設共有1000則評論，其中共有50則包含「雷到」這個詞，每則評論都被貼上正評或負評的標籤，其中45則是負評。

- 評論中有「雷到」這個詞，是負評的機率是90%

- P(負評|雷到) = P(負評∩雷到)/P(雷到) = (45/1000)/(50/1000)=0.9

$$\boxed{P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B)}}$$

貝氏定理

- 貝氏定理

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B|A)P(A) + P(B|A^C)P(A^C)} \qquad P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{\sum_j P(B|A_j)\,P(A_j)}$$

- e.g. 假設共有1000則評論，600則正評中有5則包含「雷到」，400則負評中有45則包含「雷到」，包含「雷到」是負評的機率是多少？
- 包含「雷到」(B)是負評(A)的機率 = 包含「雷到」且是負評的機率/(負評中包含「雷到」的機率+正評中包含「雷到」的機率)

$$P = \frac{\frac{45}{400}\cdot\frac{400}{1000}}{\frac{45}{400}\cdot\frac{400}{1000} + \frac{5}{600}\cdot\frac{600}{1000}} = \frac{\frac{45}{1000}}{\frac{45}{1000} + \frac{5}{1000}} = \frac{45}{50} = 0.9$$

單純貝氏分類器

- 比較分類機率的大小，機率較大者為分類結果

評論是正評的機率	**評論是負評的機率**
• P(正評 | A詞, B詞, C詞…)	• P(負評 | A詞, B詞, C詞…)
e.g. 評論包含“雷到”和“還好”是正評的機率：	e.g. 評論包含“雷到”和“還好”是負評的機率：
P(正評 | 雷到, 還好)	P(負評 | 雷到, 還好)
= P(**正評**) * P(**雷到, 還好 | 正評**) / P(雷到, 還好)	= P(**負評**) * P(**雷到, 還好 | 負評**) / P(雷到, 還好)
分子 = P(正評) * P(雷到 | 正評) * P(還好 | 正評)	**分子** = P(負評) * P(雷到 | 負評) * P(還好 | 負評)
(假設每個詞出現為**獨立事件**)	(假設每個詞出現為**獨立事件**)

分母一樣，只需比較分子大小即可判斷分類

Notes

- **若A與B為獨立事件，則：P(A∩B)=P(A)*P(B)**

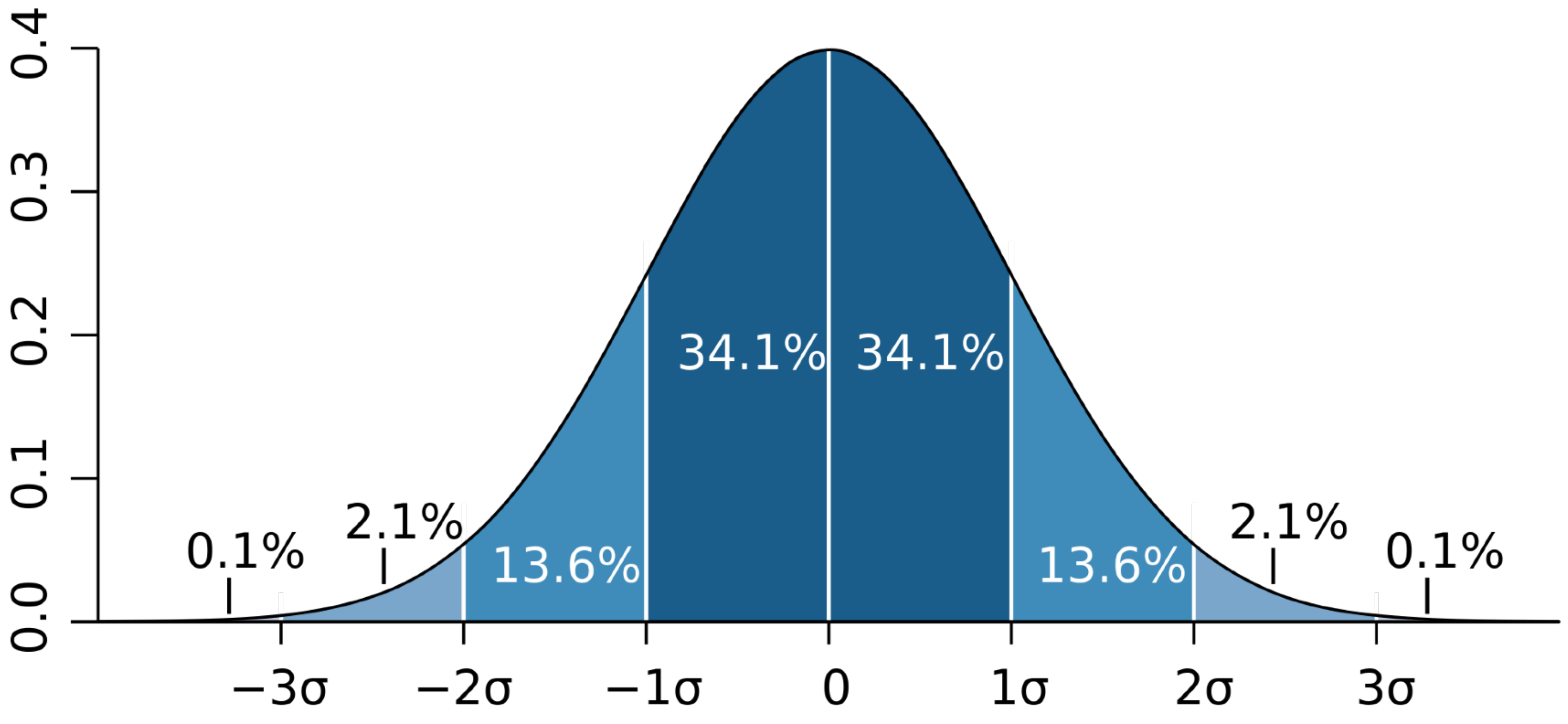
問題

1. 假設偏離現實，字詞通常互相影響
    - 最大值後驗判定(maximum a posteriori, MAP)
    - 最後只會用來選擇是正評或是負評，看誰大就是誰
2. 詞越多乘積起來越接近0（下溢問題，underflow）
    - 轉成：exp(log(p1) * log(p2) * …)
3. 若評論中不存在這個詞，則P=0
    - 設為一個很小的值，如：0.0001
    - 偽計數值k，如：假裝有看到1則(k=1)

Gaussian Naive Bayes

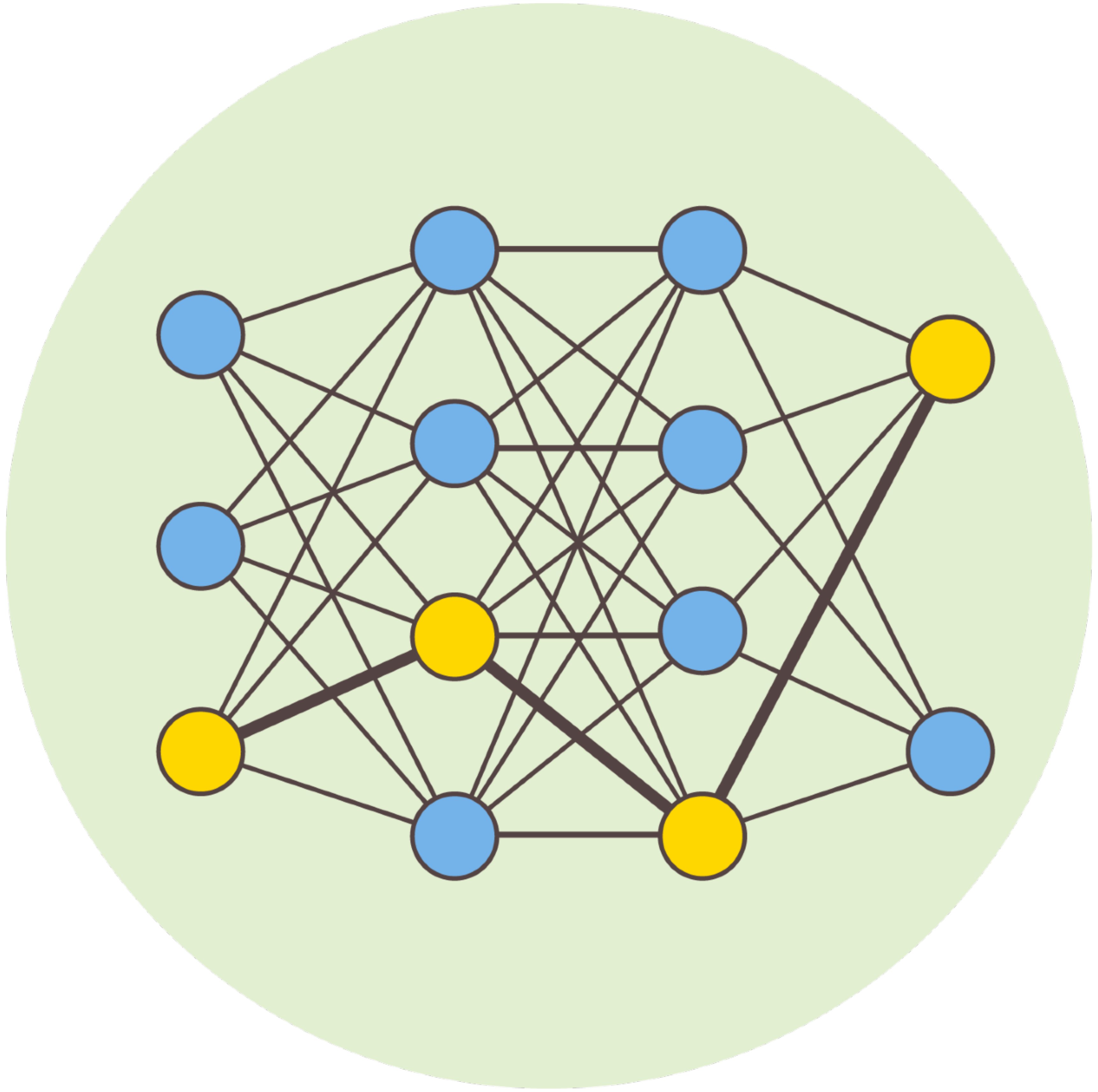
- *class* `sklearn.naive_bayes.`**`GaussianNB`**(*priors=None*)
- 假設資料是高斯分佈(常態分佈)

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma^2_y}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma^2_y}\right)$$



Note:

- scikit-learn 還提供另外兩種機率分佈的貝式分類器：MultinomialNB、BernoulliNB



Python 機器學習與深度學習實作

分類模型評估

分類效果評估

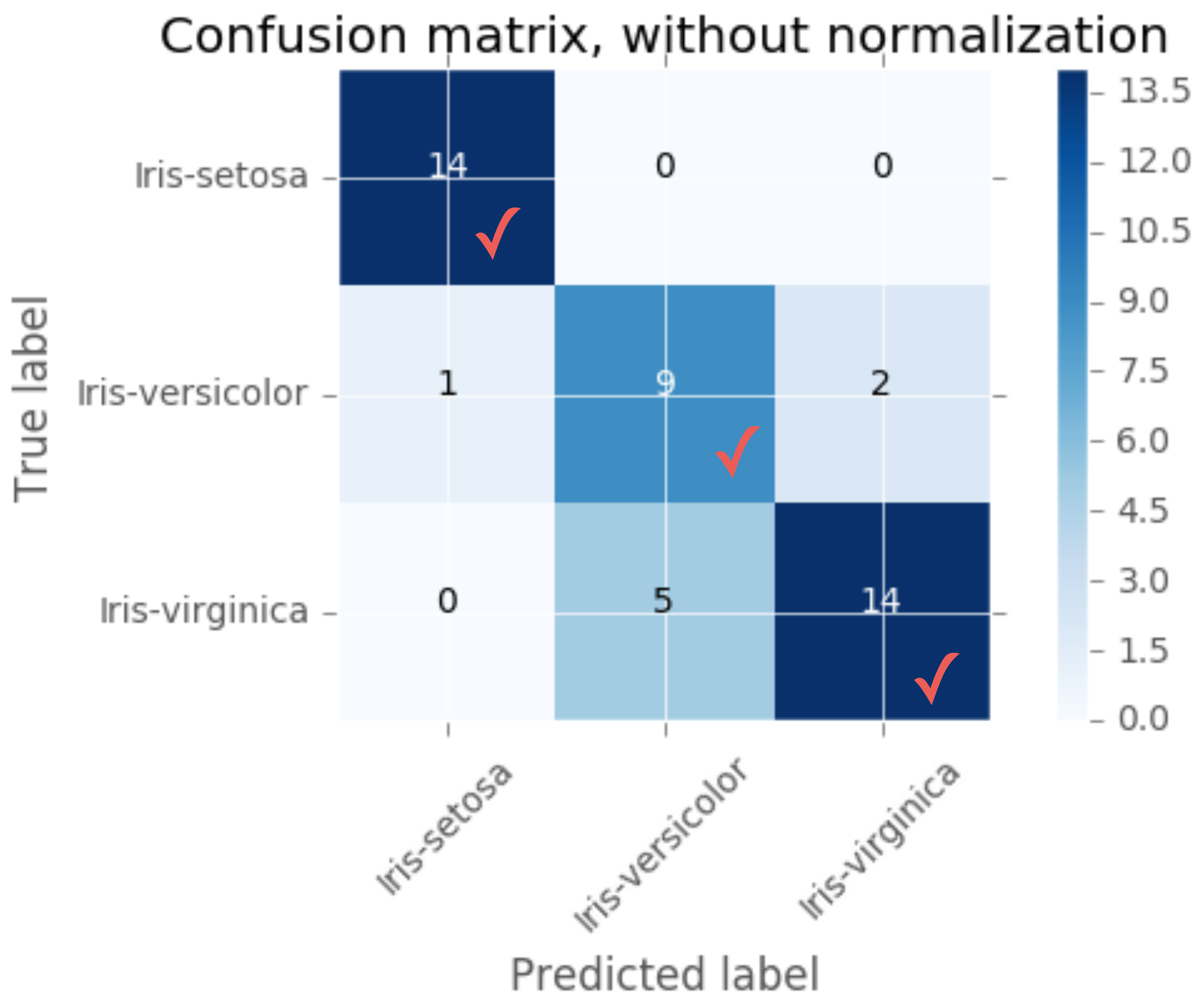
- 混淆矩陣（Confusion Matrix）

	Predited 0	Predicted 1
True 0	True negatives (TN)	False positives (FP)
True 1	False negatives (FN)	True positives (TP)

- 正確率(Accuracy): A = (TN + TP) / (TN + FN + FP + TP)
- **精確率**(Precision): P = TP / (TP + FP)
- **召回率**(Recall): R = TP / (TP + FN)
- F1 score = 2PR / (P+R) (P, R的調和平均)

分類效果評估

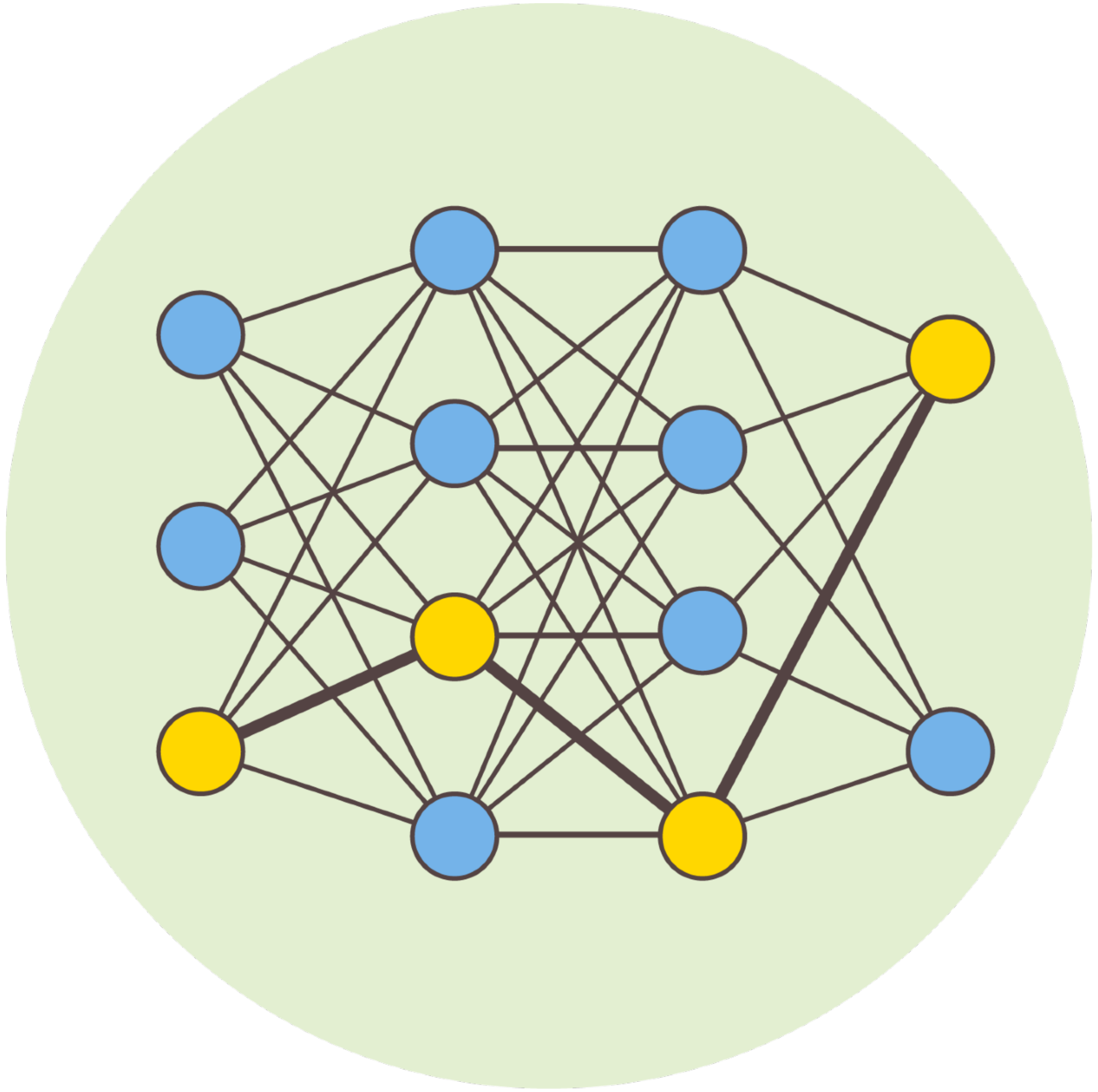
- 混淆矩陣（Confusion Matrix） - 多類別



	precision	recall	f1-score
Iris-setosa	0.93	1.00	0.97
Iris-versicolor	0.64	0.75	0.69
Iris-virginica	0.88	0.74	0.80

e.g. Iris-versicolor (變色鳶尾花)

- Precision = 9/(9+5) = 0.64
  - 預測14個變色鳶尾花，9個命中
- Recall = 9/(1+9+2) = 0.75
  - 有12個變色鳶尾花，找回了9個



Python 機器學習與深度學習實作

超參數調校

超參數調校

- 「網格搜尋」（Grid Search）是透過「暴力法」把所有超參數跑過一遍，再看何者的訓練結果最好，提供最佳超參數的結果。
- 搭配 K 折交叉驗證法（k-fold cross-validation）一起使用。
- 若超參數量大、範圍大，相當消耗運算資源。可使用「隨機搜尋」（Randomized Search）隨機抽樣超參數組合，降低運算資源。
- 不保證每次運算出的最佳超參數一樣，也不保證在「測試資料集」獲得最佳結果，但是很好的超參數參考依據。

範例1：網格搜尋

In [4]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

n_neighbors = [i for i in range(1,11,1)]
weights = ['uniform','distance']

hyperparameters = dict(n_neighbors=n_neighbors, weights=weights)

model = KNeighborsClassifier()
knn = GridSearchCV(model, hyperparameters, cv=5, verbose=0)

best_model = knn.fit(X_train_std, y_train.values.ravel())

# 查看最好的超參數
print('n_neighbors: ', best_model.best_estimator_.get_params()['n_neighbors'])
print('weights: ', best_model.best_estimator_.get_params()['weights'])
print('所有超參數: ', best_model.best_estimator_.get_params())
```

超參數範圍設定

模型選擇和訓練

查看超參數結果

```
n_neighbors:  9
weights:  uniform
所有超參數:  {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_n
eighbors': 9, 'p': 2, 'weights': 'uniform'}
```

範例2：隨機搜尋

In [7]:

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
import numpy as np

C = np.linspace(0.1,10,50)
kernel = ['linear', 'poly', 'rbf', 'sigmoid']

hyperparameters = dict(C=C, kernel=kernel)

model = SVC()
svc = RandomizedSearchCV(model, hyperparameters, cv=5, iid=False)
best_model = svc.fit(X_train_std, y_train.values.ravel())

# 查看最好的超參數
print('C: ', best_model.best_estimator_.get_params()['C'])
print('kernel: ', best_model.best_estimator_.get_params()['kernel'])
print('所有超參數: ', best_model.best_estimator_.get_params())
```

超參數範圍設定

模型選擇和訓練

查看超參數結果

```
C:  6.161224489795918
kernel:  rbf
所有超參數:  {'C': 6.161224489795918, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shap
e': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': Non
e, 'shrinking': True, 'tol': 0.001, 'verbose': False}
```