KHAI PHÁ DỮ LIỆU

Trường Đại học Nha Trang
Khoa Công nghệ thông tin
Bộ môn Hệ thống thông tin
Giáo viên: TS.Nguyễn Khắc Cường

CHỦ ĐỀ 4

PHÂN LỚP
(kNN)

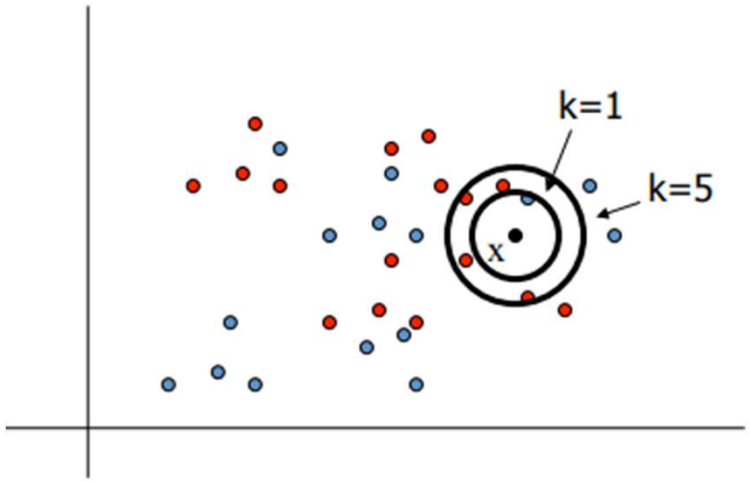
kNN

- Giới thiệu
  - kNN = k-Nearest Neighbor
  - Là một trong các giải thuật density estimation
- Idea:
  - Quá trình học:
    - không thật sự học gì cả, chỉ nhớ phân bố (sự sắp xếp phi tuyến của data space) của các training data (nên gọi là lazy algorithm)
  - Classification:
    - Xét khoảng cách từ data đang xét (data mới) đến k learning data gần nhất
    - Dựa vào các giá trị khoảng cách đó

    → đoán được class của data mới = class của các training data gần nhất có số lượng nhiều nhất

kNN

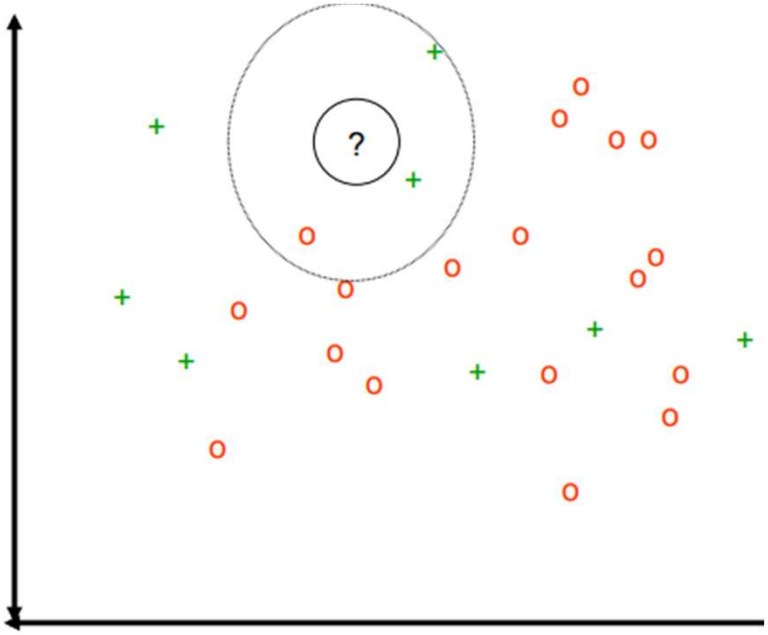
- Classification:
  - VD:



- Ưu điểm:
  - Quá trình học rất nhanh (???)
- Nhược điểm:
  - Quá trình classification chậm
  - Dễ cho kết quả sai lầm khi các training data có phân bố có sự khác biệt không rõ ràng

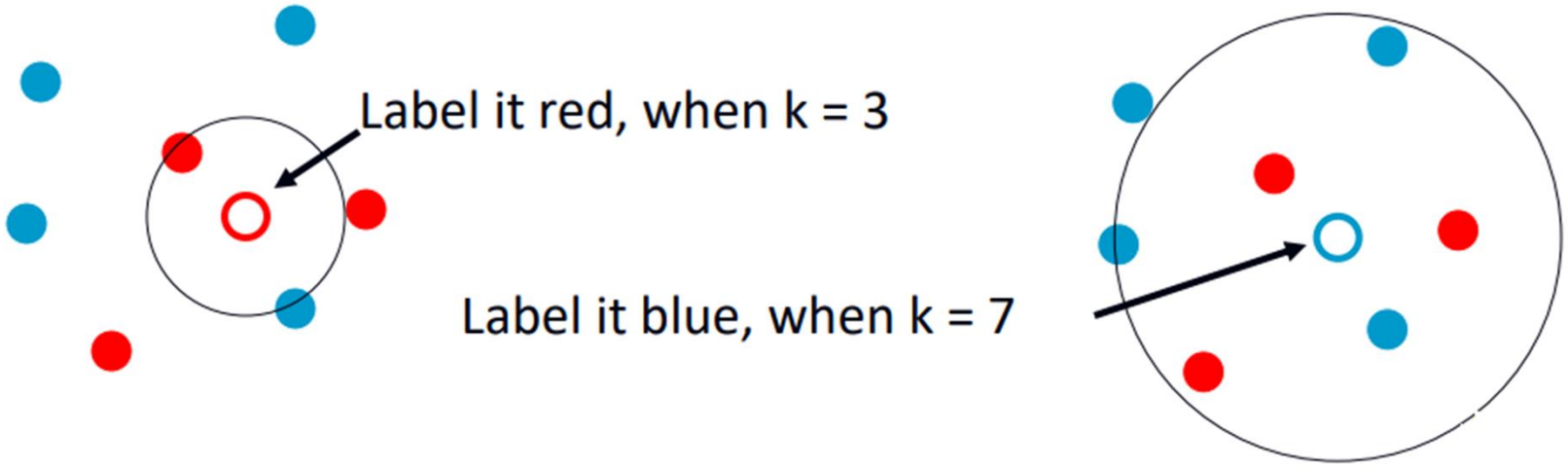
kNN

- Nhược điểm:
  - Dễ cho kết quả sai lầm khi các training data có phân bố không rõ ràng



kNN

- Nhược điểm:
  - Khó chọn k
  - Tăng k
    - giảm sai số (nhưng classification chậm hơn)
    - cũng dễ gây ra bias



kNN

- Nhược điểm:
  - Khi training data có dimension càng cao → sai số khi tính khoảng cách càng cao
- Các công thức tính khoảng cách phổ biến:

$$d_{euclidean} = \sqrt{\sum_{i=1}^{n}\left(\vec{x}_i - \vec{y}_i\right)^2} \qquad d_{manhattan} = \sum_{i=1}^{n}\left|\vec{x}_i - \vec{y}_i\right|$$

$$d_{minkowski} = \left(\sum_{i=1}^{n}\left|\vec{x}_i - \vec{y}_i\right|^p\right)^{\frac{1}{p}}$$

- Thực tế:
  - Có thể áp dụng các phép tính normalization cho training data và new data trước khi áp dụng các công thức tính khoảng cách → có thể tăng tốc độ training, classification và accuracy

kNN

- Codes: Iris dataset

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()
```

```
irisData
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
```

kNN

- Codes:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```

```
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

kNN

- Codes:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=7)

knn = KNeighborsClassifier(n_neighbors=9)

knn.fit(X_train, y_train)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

```
0.9
```

kNN

- Codes:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

```
0.9666666666666667
```

kNN

- Codes:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=5)

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

```
0.9333333333333333
```

kNN

- Codes:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Loading data
irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

# Calculate the accuracy of the model
print(knn.score(X_test, y_test))
```

```
1.0
```

Q / A