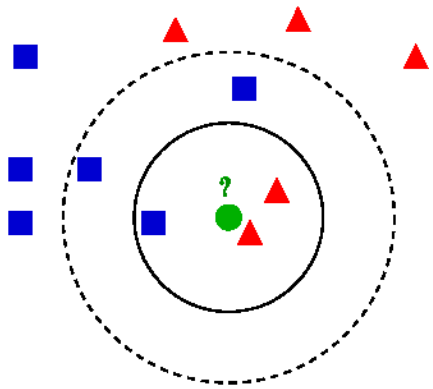
# Bài 6: MÔ HÌNH PHÂN LỚP *k* – lân cận

(*k-nearest neighbors model*)

## 1. Giới thiệu mô hình *k*-NN

**Phát biểu bài toán**: Cho tập dữ liệu $D = \{(X, y) \mid X \in R^{m \times n}, y \in \{c_i\}_{i=1}^{k}\}$ với $X$ là tập các điểm dữ liệu trong không gian $R^n$ và $y$ là vector chứa các nhãn lớp tương ứng của các điểm dữ liệu $x_i \in X$. Hình 1 minh họa áp dụng mô hình k-NN vào bài toán phân lớp.




*Hình 1: Phân lớp dữ liệu bằng mô hình k-NN (nguồn: wikipedia)*

Phép tính khoảng cách giữa các điểm là nền tảng của phương pháp k-NN, ví dụ:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (x_i^A - x_i^B)^2}$$

Nhãn lớp của điểm cần dự đoán sẽ được suy đoán từ ***k*** điểm dữ liệu gần nhất với nó theo cơ chế bỏ phiếu (nghĩa là trong *k* điểm lân cận, nhãn lớp nào chiếm đa số sẽ là nhãn lớp của điểm cần dự đoán).

Đặc điểm của mô hình k-NN:

- Quá trình xây dựng k-NN không trải qua bước huấn luyện (training). Nên đây có thể được xem là một thuật toán lười học – lazy learning;

- Việc chọn $k$ thường là một số lẻ và có chỉ số chính xác (accuracy) cao nhất. Mẹo kỹ thuật thông thường chọn giá trị của $k$ trong khoảng $1 \leq k \leq \frac{\sqrt{m}}{2}$ với $m$ là số điểm dữ liệu trong ;
- ***Mô hình k-NN còn có thể áp dụng cho bài toán hồi quy***.

## 2. Phân lớp với mô hình k-NN

Tập dữ liệu hoa diên vĩ (*Iris data set*) là tập dữ liệu phổ biến trong lĩnh vực học máy, được dùng trong các bài toán phân lớp. Tập dữ liệu này cung cấp các giá trị độ rộng của đài hoa (sepal width), độ dài của đài hoa (sepal length), độ rộng của cánh hoa (petal width) và độ dài của cánh hoa (petal length) nhằm xác định dòng hoa diên vĩ là *setosa, versicolor* hay *virginica?* (đây chính là các giá trị nhãn lớp. Thư viện sklearn cung cấp *mô hình KNeighborsClassifier* để hiện thực hóa mô hình *k*-lân cận.

*Bảng 1: Hướng dẫn sử dụng kNN với sklearn*

| STT | Ý nghĩa | Mã lệnh Python |
|---|---|---|
| 1 | Đọc tập dữ liệu hoa diên vĩ | `from sklearn import datasets`<br>`iris = datasets.load_iris()`<br>`X = iris.data`<br>`y = iris.target` |
| 2 | Huấn luyện mô hình kNN | `from sklearn.neighbors import KNeighborsClassifier`<br>`kNN = KNeighborsClassifier(n_neighbors=3)`<br>`kNN.fit(X_train, y_train)` |
| 3 | Sử dụng mô hình kNN để dự đoán | `y_hat = kNN.predict(X_test)` |
| 4 | Đánh giá độ chính xác | `from sklearn.metrics import accuracy_score`<br>`print('Độ chính xác của mô hình là: ',`<br>`accuracy_score(y_hat, y_test))` |
| 5 | Tìm giá trị k tối ưu bằng GridSearchCV | `from sklearn.model_selection import GridSearchCV`<br>`#B3: Xác định số lượng mẫu dữ liệu va k max`<br>`m = y_train.shape[0]`<br>`k_max = int(sqrt(m)/2)`<br>`print('k max: ', k_max)`<br><br>`#B4: Tạo lưới tham số với GridSearchCV`<br>`k_values = np.arange(start=1, stop = k_max + 1,`<br>`dtype=int)`<br>`print('Các giá trị k: ', k_values)`<br>`params = {'n_neighbors': k_values}`<br><br>`#B5: Khởi tạo và huấn luyện mô hình`<br>`kNN = KNeighborsClassifier()`<br>`kNN_grid = GridSearchCV(kNN, params, cv=3)`<br>`kNN grid.fit(X train, y train)`<br><br>`#B6: Thông báo giá trị k tối ưu` |

| | | |
|---|---|---|
| | | `print('Giá trị k tối ưu là: ', kNN_grid.best_params_)` |
| 6 | Dự đoán và đánh giá kết quả của GridSearchCV | `y_hat = kNN_grid.predict(X_test)`<br>`print('Độ chính xác dự đoán là: ', accuracy_score(y_hat, y_test))` |

### 3. Mở rộng: mô hình hồi quy k-NN

Tư tưởng của mô hình k-NN vẫn có thể áp dụng được cho bài toán hồi quy khi vector nhãn lớp – *y* - nhận giá trị liên tục. Thư viện sklearn cung cấp *mô hình KNeighborsRegressors* để thực hiện mô hình hồi quy theo tư tưởng của mô hình *k*-lân cận. Ví dụ được cung cấp ở phần còn lại của mục này.

```
import os
import numpy as np
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler



#Bước 1: Đọc dữ liệu
D = np.loadtxt(os.path.join('D:/data/hocmay', 'ex1data2.txt'), delimiter=',')
scaler = MinMaxScaler()
scaler.fit(D)
D = scaler.transform(D)
X, y = D[:, :-1], D[:, -1]

#B2: Phân chia train - test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=15,
                                                    test_size=0.30)

#B3: Xác định số lượng mẫu dữ liệu va k_max
m = y_train.shape[0]
k_max = int(sqrt(m)/2)
print('k max: ', k max)

#B4: Tạo lưới tham số với GridSearchCV
k_values = np.arange(start=1, stop = k_max + 1, dtype=int)
print('Các giá trị k: ', k_values)
params = {'n neighbors': k values}

#B5: Khởi tạo và huấn luyện mô hình
kNN = KNeighborsRegressor()
kNN_grid = GridSearchCV(kNN, params, cv=10)
kNN grid.fit(X train, y train)
```

```
#B6: Thông báo giá trị k tối ưu
print('Giá trị k tối ưu là: ', kNN_grid.best_params_)

#B7: Đánh giá hiệu năng trên tập test
y_hat = kNN_grid.predict(X_test)
print('Độ đo MSE là: ', mean_squared_error(y_hat, y_test))
```