

# TRƯỜNG ĐH NGOẠI NGỮ - TIN HỌC TP.HỒ CHÍ MINH KHOA CÔNG NGHỆ THÔNG TIN

Hướng dẫn thực hiện đề tài môn **CSDL NC Bộ môn Hệ thống Thông tin**

BÀI BÁO CÁO KẾT THÚC HỌC PHẦN TRÍ TUỆ NHÂN TẠO

**TRÍ TUỆ NHÂN TẠO**

Giảng viên hướng dẫn: Dương Tuấn Anh

Sinh viên thực hiện:

1. Phạm Đình Anh Phương 22DH112900
2. Nguyễn Kim Ngân 22DH112315

Thành phố Hồ Chí Minh, tháng 07/2024

*Hình 1. Trang bìa*

**MỤC LỤC**

1. [GIỚI THIỆU K-NEAREST NEIGHBORS (K-NN) 1](#_TOC_250032)
   1. [GIẢI THUẬT PHÂN LỚP CÂY QUYẾT ĐỊNH LÀ GÌ ? 1](#_TOC_250031)
   2. [MÔ TẢ GIẢI THUẬT 1](#_TOC_250030)
2. [GIỚI THIỆU VỀ SCIKT - LEARN 3](#_TOC_250029)
   1. [Scikit – learn là gì 3](#_TOC_250028)
   2. [Các tính năng và ứng dụng chính 4](#_TOC_250027)
3. GIỚI THIỆU VỀ LENSES [4](#_TOC_250026)
4. [TÍNH TOÁN ĐỘ CHÍNH XÁC 4](#_TOC_250025)
5. [ÁP DỤNG K-NN VÀO PHÂN LỚP HOA IRIS 5](#_TOC_250024)
   1. [ĐỊNH NGHĨA BÀI TOÁN 5](#_TOC_250023)
   2. [THU THẬP VÀ TIỀN XỬ LÝ DỮ LIỆU 6](#_TOC_250022)
      1. [Làm sạch dữ liệu (data cleaning) 6](#_TOC_250021)
      2. [Chọn lọc dữ liệu (data selection) 7](#_TOC_250020)
      3. [Chuyển đổi dữ liệu (data transformation) 7](#_TOC_250019)
      4. [Chuẩn hoá dữ liệu (data normalization) 7](#_TOC_250018)
   3. [TRIỂN KHAI PHÂN LỚP HOA IRIS 10](#_TOC_250017)
      1. Cách thức đo độ chính xác của mô hình 13
   4. PHƯƠNG PHÁP CHỌN SỐ K THÍCH HỢP 13
6. [K-NN VỚI TRỌNG SỐ 13](#_TOC_250016)
7. [SO SÁNH KẾT QUẢ VỚI THƯ VIỆN SKLEARN 14](#_TOC_250015)
8. [KẾT LUẬN 15](#_TOC_250014)
9. [MÃ NGUỒN 16](#_TOC_250013)
   1. [CÀI ĐẶT CÁC THƯ VIỆN CẦN THIẾT 16](#_TOC_250012)
   2. [XÂY DỰNG LỚP MÔ HÌNH K-NN 16](#_TOC_250011)
   3. [ÁP DỤNG MÔ HÌNH K-NN VÀO BỘ DỮ LIỆU IRIS 20](#_TOC_250010)
      1. [Tải bộ dữ liệu 20](#_TOC_250009)
      2. [Tiền xử lý dữ liệu 20](#_TOC_250008)
      3. [Xây dựng mô hình KNN cho tập dữ liệu Iris 21](#_TOC_250007)
      4. [Thay đổi giá trị k tìm giá trị tốt nhất 21](#_TOC_250006)
      5. [Dùng mô hình K-NN có đánh trọng số 23](#_TOC_250005)
   4. [SO SÁNH VỚI KẾT QUẢ CỦA THƯ VIỆN SKLEARN 24](#_TOC_250004)
      1. [Dự đoán kết quả tập test dùng mô hình Knn 24](#_TOC_250003)
      2. [Thay đổi k tìm giá trị tốt nhất 25](#_TOC_250002)
      3. [Đánh trọng số cho các điểm lân cận 26](#_TOC_250001)
10. [TÀI LIỆU THAM KHẢO 28](#_TOC_250000)

Hình 1. Ảnh bìa 2

Hình 2. K-NN với giá trị nhiễu 3

Hình 3. Ba loài hoa Iris (Nguồn: Wikipedia) 5

Hình 4. Bài toán phần lớp hoa Iris 5

Hình 5. Giá trị mẫu 3 dòng dữ liệu Iris 10

Hình 6. Sự tương quan phân bố dữ liệu theo thuộc tính 12

Hình 7. Chọn số k thích hợp 13

Hình 8. Tìm số k thích hợp theo thư viện sklearn 14

Bảng 1. Giá trị trong tập dữ liệu Iris 6

Bảng 2. Dữ liệu minh hoạ trước khi chuẩn hoá 8

Bảng 3. Khoảng cách minh hoạ trước khi chuẩn hoá 8

Bảng 4. Dữ liệu minh hoạ sau khi chuẩn hoá 9

Bảng 5. Khoảng cách minh hoạ sau khi chuẩn hoá 9

Bảng 6. Giá trị đặc trưng chênh lệch 11

# GIỚI THIỆU GIẢI THUẬT PHÂN LỚP CÂY QUYẾT ĐỊNH (DECISION TREE)

### GIẢI THUẬT PHÂN LỚP CÂY QUYẾT ĐỊNH LÀ GÌ ?

**- Cây quyết định** (Decision Tree) là một thuật toán học máy được sử dụng rộng rãi trong các bài toán phân loại và hồi quy. Đây là một công cụ mạnh mẽ và trực quan giúp phân loại các đối tượng hoặc dự đoán giá trị của biến mục tiêu bằng cách chia dữ liệu thành các nhóm nhỏ hơn dựa trên các thuộc tính đầu vào (features). Cây quyết định có cấu trúc dạng cây, với các nút biểu thị các quyết định dựa trên các đặc điểm của dữ liệu và các nhánh biểu thị kết quả của các quyết định đó.

- Các Thành Phần Chính của Cây Quyết Định

+ Nút gốc (Root Node):

Là điểm bắt đầu của cây, chứa toàn bộ dữ liệu và biểu thị đặc điểm đầu tiên được dùng để phân chia dữ liệu.

+ Nút trong (Internal Nodes):

Các nút này đại diện cho các đặc điểm trong dữ liệu và điều kiện được sử dụng để tiếp tục phân chia dữ liệu.

+ Nhánh (Branches):

Các đường nối giữa các nút, thể hiện kết quả của một điều kiện hoặc quyết định.

+ Nút lá (Leaf Nodes):

Các nút cuối cùng của cây, nơi mà các quyết định cuối cùng được đưa ra, biểu thị các nhãn lớp (class labels) hoặc giá trị mục tiêu.

### Mô tả mô hình cây quyết định

Thuật toán cây quyết định xây dựng một mô hình quyết định bằng cách chia dữ liệu thành các tập con nhỏ hơn dựa trên các đặc điểm đầu vào, cho đến khi đạt được các điều kiện dừng cụ thể. Dưới đây là mô tả chi tiết về cách thức hoạt động của thuật toán này.

*Các Bước Của Thuật Toán Cây Quyết Định*

* Bắt Đầu Với Toàn Bộ Dữ Liệu:

+ Thuật toán bắt đầu với toàn bộ dữ liệu đào tạo tại nút gốc của cây.

* Tính Toán Độ Đo Để Chọn Đặc Điểm Phân Chia:

+ Đối với mỗi đặc điểm, tính toán một độ đo để đánh giá mức độ "tốt" của việc phân chia dữ liệu dựa trên đặc điểm đó. Các độ đo phổ biến bao gồm Entropy và Information Gain, hoặc Gini Index.

**+ Entropy** đo lường độ hỗn loạn của dữ liệu, và **Information Gain** đo lường mức giảm hỗn loạn sau khi phân chia.

**+ Gini Index** đo lường mức độ "thuần khiết" của các tập hợp con.

* Chọn Đặc Điểm Tốt Nhất Để Phân Chia:

+ Chọn đặc điểm có độ đo tốt nhất (ví dụ: Information Gain cao nhất hoặc Gini Index thấp nhất) để phân chia dữ liệu tại nút hiện tại.

* Phân Chia Dữ Liệu:

+ Phân chia dữ liệu thành các tập con dựa trên giá trị của đặc điểm đã chọn. Mỗi nhánh của cây sẽ đại diện cho một trong các giá trị có thể của đặc điểm đó.

* Lặp Lại Quy Trình Cho Từng Nhánh:

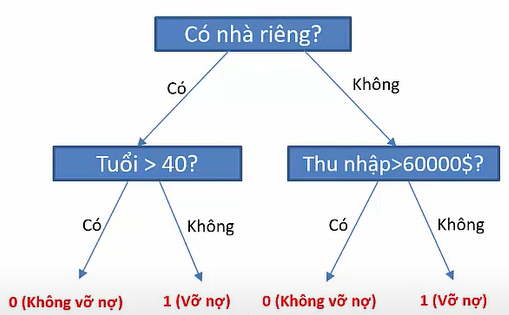
+ Lặp lại quá trình tính toán độ đo, chọn đặc điểm, và phân chia dữ liệu cho từng nhánh con, tạo thành các nút trong mới.

+ Quy trình này tiếp tục cho đến khi đạt một điều kiện dừng, chẳng hạn như:

* + - Tất cả các mẫu trong một nút đều thuộc cùng một lớp.
    - Không còn đặc điểm nào để phân chia.
    - Đạt đến độ sâu tối đa của cây.
* Gán Nhãn Lớp Cho Nút Lá**:**

+Khi đạt điều kiện dừng tại một nút, nút đó trở thành nút lá và được gán nhãn lớp dựa trên các mẫu trong nút đó.

Ví Dụ Đơn Giản Về Mô Hình Này :



* **Nút gốc (Root Node):** "Có nhà riêng?"

+ Đây là quyết định đầu tiên và quan trọng nhất. Nếu người đó có nhà riêng, ta chuyển

+ đến nút tiếp theo bên trái; nếu không, ta chuyển đến nút tiếp theo bên phải.

* **Nút bên trái (Có nhà riêng):** "Tuổi > 40?"

+ Nếu tuổi của người đó lớn hơn 40, họ có khả năng **không vỡ nợ** (0).

+ Nếu tuổi của người đó không lớn hơn 40, họ có khả năng **vỡ nợ** (1).

* **Nút bên phải (Không có nhà riêng):** "Thu nhập > 60000$?"

+ Nếu thu nhập của người đó lớn hơn 60000$, họ có khả năng **không vỡ nợ** (0).

+ Nếu thu nhập của người đó không lớn hơn 60000$, họ có khả năng **vỡ nợ** (1).

**Kết Luận :**

* Thuật toán cây quyết định là một phương pháp hiệu quả và dễ hiểu để phân loại dữ liệu. Bằng cách liên tục phân chia dữ liệu dựa trên các đặc điểm đầu vào, thuật toán này tạo ra một mô hình quyết định dễ dàng giải thích và trực quan. Tuy nhiên, để tránh overfitting, cần phải kiểm soát độ sâu và kích thước của cây quyết định.

# GIỚI THIỆU VỀ SCIKIT-LEARN

### Scikit-Learn là gì

**Scikit-Learn** là một thư viện mã nguồn mở mạnh mẽ cho học máy (machine learning) được xây dựng trên các thư viện NumPy, SciPy và matplotlib của Python. Được phát triển bởi cộng đồng các nhà nghiên cứu và kỹ sư, Scikit-Learn cung cấp một tập hợp phong phú các công cụ để học máy và khai thác dữ liệu.

Các đặc điểm nổi bật của Scikit-Learn:

-Dễ sử dụng**:** API của Scikit-Learn được thiết kế đơn giản và dễ hiểu, giúp người dùng nhanh chóng xây dựng và thử nghiệm các mô hình học máy.

-Hiệu quả cao**:** Scikit-Learn được tối ưu hóa để xử lý hiệu quả các tập dữ liệu lớn.

-Tính mô-đun**:** Thư viện cung cấp các công cụ mô-đun để tiền xử lý dữ liệu, lựa chọn mô hình, huấn luyện và đánh giá.

-Tài liệu phong phú**:** Scikit-Learn có tài liệu chi tiết và nhiều ví dụ, giúp người dùng dễ dàng tiếp cận và học hỏi.

### Các tính năng và ứng dụng chính

Scikit-Learn hỗ trợ nhiều tính năng và thuật toán học máy, bao gồm:

* Tiền xử lý dữ liệu (Data Preprocessing):
* Chuẩn hóa (Normalization): Điều chỉnh các đặc điểm để có độ lớn tương đương.
* Mã hóa (Encoding): Biến đổi các đặc điểm phân loại thành dạng số.
* Xử lý giá trị thiếu (Imputation): Thay thế các giá trị thiếu trong dữ liệu.
* Phân lớp (Classification):
* Các thuật toán phổ biến như K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees (Cây quyết định), Random Forests, và Naive Bayes.
* Hồi quy (Regression):
* Các thuật toán như Linear Regression, Ridge Regression, Lasso, và Support Vector Regression (SVR).
* Cụm (Clustering):
* K-Means, DBSCAN, Agglomerative Clustering, và các thuật toán phân cụm khác.
* Giảm kích thước (Dimensionality Reduction):
* Principal Component Analysis (PCA), Singular Value Decomposition (SVD), và t-SNE.
* Đánh giá mô hình (Model Evaluation):
* Cross-validation, độ chính xác, độ nhạy, độ đặc hiệu, và các chỉ số đánh giá khác.
* Lựa chọn mô hình (Model Selection):
* Grid Search, Random Search, và các kỹ thuật lựa chọn tham số khác.

# GIỚI THIỆU VỀ LENSES

## 3.1 lenses là gì, nguồn gốc và mô tả bộ dữ liệu

**-** Bộ dữ liệu Lenses (Contact Lenses) là một tập dữ liệu mẫu được sử dụng rộng rãi trong các bài toán phân lớp. Bộ dữ liệu này được lấy từ UCI Machine Learning Repository, một kho dữ liệu học máy phổ biến.

- Nguồn gốc:

* Bộ dữ liệu được biên soạn bởi C. J. C. Burges từ Trung tâm Khoa học Máy tính và Trí tuệ Nhân tạo của MIT.
* Mô tả bộ dữ liệu:
* Bộ dữ liệu chứa thông tin về các đặc điểm của bệnh nhân và loại kính áp tròng được đề xuất.
* Số lượng mẫu: 24 mẫu.
* Số lượng đặc điểm: 4 đặc điểm đầu vào.
* Số lượng nhãn: 3 loại kính áp tròng (hard, soft, none).

## 3.2 Các đặc điểm và nhãn của bộ dữ liệu

Các đặc điểm **(Attributes):**

1. Age (Tuổi):
   * Trẻ (Young)
   * Trung niên (Pre-presbyopic)
   * Già (Presbyopic)
2. Spectacle Prescription (Đơn kính):
   * Cận (Myope)
   * Viễn (Hypermetrope)
3. Astigmatism (Loạn thị):
   * Có (Yes)
   * Không (No)
4. Tear Production Rate (Tốc độ sản xuất nước mắt):
   * Giảm (Reduced)
   * Bình thường (Normal)

**Nhãn (Labels):**

* Hard Contact Lenses (Kính áp tròng cứng): Loại kính áp tròng cứng được đề xuất cho những người có các điều kiện cụ thể về mắt.
* Soft Contact Lenses (Kính áp tròng mềm): Loại kính áp tròng mềm được đề xuất cho những người có các điều kiện khác.
* No Contact Lenses (Không có kính áp tròng): Không đề xuất kính áp tròng cho những người không đủ điều kiện.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Age | Spectacle Prescription | Astigmatism | Tear Production Rate | Lenses |
| Young | Myope | No | Normal | Soft |
| Presbyopic | Hypermetrope | Yes | Reduced | None |
| Pre-presbyopic | Myope | No | |  | | --- | |  |  |  | | --- | | Normal | | Hard |
| ... | ... | ... | ... | ... |

Bộ dữ liệu Lenses là một ví dụ điển hình và đơn giản để minh họa cách hoạt động của các thuật toán phân lớp, đặc biệt là cây quyết định. Sự phân loại dựa trên các đặc điểm đầu vào cụ thể giúp xây dựng các mô hình học máy có thể đưa ra quyết định chính xác và dễ hiểu. Scikit-Learn cung cấp các công cụ mạnh mẽ để thực hiện và đánh giá các mô hình này, từ đó giúp người dùng có cái nhìn sâu sắc và toàn diện hơn về dữ liệu và các thuật toán học máy. q

# Xử lý và Phân tích Dữ liệu

## Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là bước quan trọng trong việc chuẩn bị dữ liệu để xây dựng mô hình. Các bước tiền xử lý thường bao gồm:

* Làm sạch dữ liệu**:** Xử lý các giá trị thiếu (missing values), loại bỏ các giá trị ngoại lệ (outliers), và sửa chữa các lỗi trong dữ liệu.
* Mã hóa các giá trị danh mục (Categorical Encoding): Chuyển đổi các giá trị danh mục (categorical values) thành các giá trị số. Các phương pháp phổ biến bao gồm:
  + One-hot encoding: Tạo ra các cột nhị phân riêng lẻ cho mỗi giá trị danh mục.
  + Label encoding: Gán các giá trị danh mục vào các số nguyên.

- Chuẩn hóa dữ liệu (Normalization): Điều chỉnh các giá trị của dữ liệu về một thang đo chung, thường trong khoảng [0, 1] hoặc [-1, 1], để giúp mô hình hoạt động hiệu quả hơn.

## 4.2 Khám phá dữ liệu (Exploratory Data Analysis)

Khám phá dữ liệu giúp chúng ta hiểu rõ hơn về cấu trúc của dữ liệu và mối quan hệ giữa các đặc trưng. Các bước khám phá dữ liệu bao gồm:

* Thống kê mô tả (Descriptive Statistics): Tính toán các giá trị trung bình, độ lệch chuẩn, giá trị lớn nhất và nhỏ nhất, v.v.
* Trực quan hóa dữ liệu (Data Visualization): Sử dụng các biểu đồ như histogram, scatter plot, box plot, heatmap, v.v., để hình dung mối quan hệ giữa các đặc trưng và phát hiện các mẫu trong dữ liệu.
* Phân tích mối quan hệ (Correlation Analysis): Xác định mối quan hệ giữa các đặc trưng, sử dụng ma trận tương quan (correlation matrix) để đánh giá mức độ liên quan giữa các đặc trưng.

## Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra

* Chia dữ liệu thành hai phần: tập huấn luyện (training set) và tập kiểm tra (test set), giúp đánh giá hiệu suất của mô hình một cách chính xác. Thông thường, tỷ lệ chia dữ liệu là 70-80% cho tập huấn luyện và 20-30% cho tập kiểm tra. Cách chia này giúp đảm bảo rằng mô hình được kiểm tra trên một tập dữ liệu chưa từng được nhìn thấy trong quá trình huấn luyện.

Ví dụ sử dụng Scikit-Learn để chia tập dữ liệu:

# Xây dựng Mô Hình Cây Quyết Định

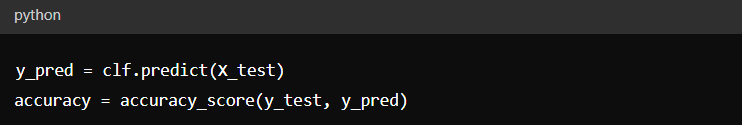
## 5.1. Huấn luyện mô hình

Để huấn luyện mô hình, chúng ta sử dụng phương thức fit của Scikit-Learn:



## 5.2. Đánh giá hiệu suất mô hình

Chúng ta có thể sử dụng độ chính xác (accuracy) để đánh giá hiệu suất của mô hình:



## 5.3. Tối ưu hóa và tinh chỉnh mô hình

Tinh chỉnh các tham số của mô hình để đạt được hiệu suất tốt nhất. Ví dụ: tinh chỉnh độ sâu tối đa của cây (max\_depth), tiêu chí chia (criterion),

# SO SÁNH KẾT QUẢ

(chưa làm )

# KẾT LUẬN

* **Ưu điểm** của mô hình cây quyết định :

1. **Dễ Hiểu và Giải Thích**: Cây quyết định rất trực quan và dễ hiểu. Các quy tắc phân loại được biểu diễn dưới dạng cây, giúp người dùng dễ dàng theo dõi và giải thích quá trình ra quyết định của mô hình.
2. **Ít Yêu Cầu Tiền Xử Lý Dữ Liệu**: Cây quyết định không yêu cầu dữ liệu phải được chuẩn hóa hay biến đổi đặc biệt nào trước khi huấn luyện. Chúng có thể xử lý cả dữ liệu số và dữ liệu phân loại.
3. **Xử Lý Tốt Các Tính Năng Không Tuyến Tính**: Mô hình cây quyết định có khả năng nắm bắt các mối quan hệ phi tuyến tính giữa các biến đầu vào và đầu ra mà không cần biến đổi đặc biệt.
4. **Không Đòi Hỏi Thông Số**: Cây quyết định không yêu cầu người dùng phải chọn trước các thông số phức tạp như các mô hình học máy khác (ví dụ: mạng nơ-ron hay SVM).
5. **Tính Mạnh**: Có thể xử lý dữ liệu thiếu hụt và không yêu cầu phải loại bỏ các quan sát có giá trị thiếu.

* **Nhược Điểm** Của Mô Hình Cây Quyết Định:

1. **Quá Khớp Dữ Liệu:** Cây quyết định dễ dàng bị quá khớp nếu không được cắt tỉa đúng cách. Điều này có thể dẫn đến hiệu suất kém trên tập dữ liệu kiểm tra.
2. **Nhạy Cảm Với Sự Thay Đổi Trong Dữ Liệu**: Một thay đổi nhỏ trong dữ liệu có thể dẫn đến một cây quyết định hoàn toàn khác. Điều này làm cho cây quyết định kém ổn định so với một số mô hình học máy khác.
3. **Hiệu Suất Kém Trên Dữ Liệu Lớn**: Khi số lượng mẫu và các tính năng tăng lên, cây quyết định có thể trở nên phức tạp và khó hiểu. Điều này cũng có thể làm giảm hiệu suất của mô hình.
4. **Thiếu Khả Năng Tổng Quát Hóa**: Cây quyết định có thể tạo ra các quyết định dựa trên các đặc điểm rất cụ thể của tập huấn luyện, dẫn đến khả năng tổng quát hóa kém khi áp dụng cho dữ liệu mới.
5. **Không Hữu Hiệu Với Dữ Liệu Mất Cân Bằng**: Trong trường hợp dữ liệu có sự mất cân bằng giữa các lớp, cây quyết định có thể tạo ra các mô hình không chính xác, thường ưu tiên lớp có số lượng mẫu lớn hơn.

* **Tổng Kết**

Mô hình cây quyết định có nhiều ưu điểm về tính trực quan và dễ hiểu, nhưng cũng có nhược điểm về khả năng quá khớp và nhạy cảm với thay đổi trong dữ liệu. Để khắc phục một số nhược điểm này, có thể sử dụng các phương pháp kết hợp như Random Forest hoặc Gradient Boosting để cải thiện hiệu suất và độ ổn định của mô hình.v

# MÃ NGUỒN (chưa làm)

Giải thuật KNN được nhóm triển khai trên môi trường Google Colab.

***Link mã nguồn:*** Tham khảo tại đây

### Cài đặt các thư viện cần thiết

Một số thư viện cần thiết được dùng trong bài gồm:

* Thư viện pandas dùng để load dữ liệu dưới dạng bảng (csv).
* Thư viện numpy dùng để hỗ trợ tính toán ma trận.
* Thư viện sklearn dùng để kiểm tra so sánh kết quả với giải thuật hiện thực.
* Thư viện matplotlib và seaborn cho trực quan dữ liệu.

import pandas as pd import numpy as np import operator

import matplotlib.pyplot as plt from collections import Counter from sklearn.utils import shuffle

import seaborn as sns

### Xây dựng lớp mô hình K-NN

Nhóm đã xây dựng một lớp *KnearestNeighborsClassifier* với các tham số khởi tạo là số lượng điểm lân cận k (*n\_neighbors*) và độ đo khoảng cách (*metric*) sẽ sử dụng ('euclidean', 'manhattan'), phương pháp đánh trọng số (*weights*) sẽ sử dụng ('uniform', 'distance', custum\_weight\_function).

Việc xây dựng lớp mô hình sẽ giúp việc khởi tạo và tái sử dụng mô hình dễ dàng hơn.

* Hàm fit(*self*, *X*, *y*): Giúp đưa dữ liệu huấn luyện vào mô hình. Với X là mảng các giá trị đặc trưng của điểm dữ liệu với mỗi hàng là một điểm dữ liệu. Với y là vector nhãn dữ liệu. Dữ liệu sẽ được chuyển đổi thành từng cặp giá trị đặc trưng và nhãn dữ liệu.
* Hàm \_calculate\_distances(*self*, *X*): Tính toán khoảng cách của các điểm dữ liệu cần dự đoán với các điểm dữ liệu trong tập huấn luyện.
* Hàm \_get\_k\_neighbors(*self*, *X*): Trả về khoảng cách và chỉ số thứ tự trong tập dữ liệu huấn luyện của k điểm dữ liệu gần nhất với dữ liệu cần dự đoán.
* Hàm \_get\_weights(*self*, *dist*): Tính trọng số của khoảng cách.
* Hàm \_weighted\_mode(*self*, *a*, *w*, \*, *axis*=0): Giúp bầu chọn nhãn cho dữ liệu cần dự đoán dựa theo phương pháp đánh trọng số lựa chọn.
* Hàm score(*self*, *X*, *y*): Trả về trực tiếp giá trị độ chính xác khi đưa dữ liệu cần kiểm tra.

*class* KNearestNeighborsClassifier():

*def*  init (*self*, *n\_neighbors*=5, *metric*='euclidean', *weights*='uniform'):

self.n\_neighbors = n\_neighbors self.metric = metric self.weights = weights

*def* fit(*self*, *X*, *y*): self.X\_fit\_ = np.asarray(X) self.y\_fit\_ = np.asarray(y)

*def* \_calculate\_distances(*self*, *X*): if self.metric == 'euclidean':

X = np.expand\_dims(X, 1)

distances = np.linalg.norm(X - self.X\_fit\_, axis=-1) elif self.metric == 'manhattan':

distances = np.sum(np.abs(X - self.X\_fit\_), axis=-1) return distances

*def* \_get\_k\_neighbors(*self*, *X*):

distances = self.\_calculate\_distances(X)

neigh\_ind = np.argsort(distances)[:,:self.n\_neighbors] neigh\_dist = np.asarray([np.take(distances[i], neigh\_ind[i])

for i in range(neigh\_ind.shape[0])]) return neigh\_dist, neigh\_ind

*def* \_get\_weights(*self*, *dist*):

if self.weights in (None, 'uniform'): return None

elif self.weights == 'distance':

with np.errstate(divide='ignore'): dist = 1. / dist

inf\_mask = np.isinf(dist)

inf\_row = np.any(inf\_mask, axis=1) dist[inf\_row] = inf\_mask[inf\_row] return dist

elif callable(self.weights): return weights(dist)

*def* \_weighted\_mode(*self*, *a*, *w*, \*, *axis*=0): if axis is None:

a = np.ravel(a) w = np.ravel(w) axis = 0

else:

a = np.asarray(a) w = np.asarray(w)

if a.shape != w.shape:

w = np.full(a.shape, w, dtype=w.dtype)

scores = np.unique(np.ravel(a)) # get ALL unique values testshape = list(a.shape)

testshape[axis] = 1

oldmostfreq = np.zeros(testshape) oldcounts = np.zeros(testshape) for score in scores:

template = np.zeros(a.shape) ind = (a == score) template[ind] = w[ind]

counts = np.expand\_dims(np.sum(template, axis), axis) mostfrequent = np.where(counts > oldcounts, score,

oldmostfreq)

oldcounts = np.maximum(counts, oldcounts) oldmostfreq = mostfrequent

return mostfrequent, oldcounts

*def* predict(*self*, *X*): X\_ = np.asarray(X)

neigh\_dist, neigh\_ind = self.\_get\_k\_neighbors(X\_) weights = self.\_get\_weights(neigh\_dist)

if weights is None:

mode, \_ = stats.mode(self.y\_fit\_[neigh\_ind], axis=1) else:

mode, \_ = self.\_weighted\_mode(self.y\_fit\_[neigh\_ind], weights, axis=1)

mode = np.asarray(mode.ravel(), dtype=np.intp)

return mode

*def* score(*self*, *X*, *y*): predictions = self.predict(X) correct = 0

for i in range(len(y)):

if y[i] == predictions[i]: correct = correct + 1

return correct / float(len(y))

* Hàm hỗ trợ:
  + get\_accuracy :hỗ trợ tính toán độ chính xác của mô hình bằng cách tính tỉ lệ dự đoán đúng của mô hình so với nhãn biết trước
  + draw\_varying\_neighbor : Giúp tính toán độ chính xác của tập dữ liệu huấn luyện và tập kiểm tra với sự thay đổi của giá trị k lân cận và vẽ biểu đồ sự thay đổi.

*def* get\_accuracy(*labels*, *predictions*): correct = 0

for i in range(len(labels)):

if labels[i] == predictions[i]: correct = correct + 1

return (correct / float(len(labels))) \* 100.0

*def* draw\_varying\_neighbor(*X\_train*, *y\_train*, *X\_test*, *y\_test*, *model\_class*=KNearestNeighborsClassifier, *weights*='uniform'):

#Setup arrays to store training and test accuracies neighbors = np.arange(1,20)

train\_accuracy =np.empty(len(neighbors)) test\_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):

#Setup a knn classifier with k neighbors

knn = model\_class(n\_neighbors=k, weights=weights)

#Fit the model knn.fit(X\_train, y\_train)

#Compute accuracy on the training set train\_accuracy[i] = knn.score(X\_train, y\_train)

#Compute accuracy on the test set test\_accuracy[i] = knn.score(X\_test, y\_test)

#Generate plot plt.figure(figsize=(15,7))

plt.title('k-NN Varying number of neighbors') plt.plot(neighbors, test\_accuracy, label='Testing Accuracy') plt.plot(neighbors, train\_accuracy, label='Training accuracy') plt.legend()

plt.xlabel('Number of neighbors') plt.ylabel('Accuracy') plt.show()

### Áp dụng mô hình K-NN vào bộ dữ liệu Iris

#### Tải bộ dữ liệu

dataset\_filename = "/content/drive/MyDrive/MasterBK\_2020/Machine

Learning/data/Iris.csv"

dataset = pd.read\_csv(dataset\_filename)

print(*f*"[INFO] Dataset shape: {dataset.shape}") dataset.head()

dataset.info()

dataset['Species'].value\_counts().plot(kind='bar')

dataset.describe()

sns.pairplot(dataset.drop(columns='Id'), hue = 'Species',

kind='scatter')

plt.show()

#### Tiền xử lý dữ liệu

X = dataset.drop(['Id', 'Species'], axis=1)

y = dataset['Species']

* Mã hoá dữ liệu

Nhãn dữ liệu thuộc loại kiểu dữ liệu phân loại. Do đó cần phải chuyển đổi nhãn từ kiểu kí tự sang số nguyên trước khi đưa vào mô hình.

Iris-setosa tương ứng 0 Iris-versicolor tương 1 Iris-virginica tương ứng 2

from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

y = le.fit\_transform(y)

* Chia tập dữ liệu thành tập train và test tỉ lệ test\_size = 20%

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=2)

* Chuẩn hoá dữ liệu

X\_train\_norm = scaler.fit\_transform(X\_train)

X\_test\_norm = scaler.fit\_transform(X\_test)

#### Xây dựng mô hình KNN cho tập dữ liệu Iris

* Không có chuẩn hoá dữ liệu

knn\_model = KNearestNeighborsClassifier(n\_neighbors=5) knn\_model.fit(X\_train, y\_train)

predictions = knn\_model.predict(X\_test) print(y\_test)

print(predictions)

print("Accuracy: ", get\_accuracy(y\_test, predictions), "%")

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

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

Accuracy: 100.0 %

* Có chuẩn hoá dữ liệu

knn\_model.fit(X\_train\_norm, y\_train) predictions = knn\_model.predict(X\_test\_norm) print(y\_test)

print(predictions)

print("Accuracy: ", get\_accuracy(y\_test, predictions), "%")

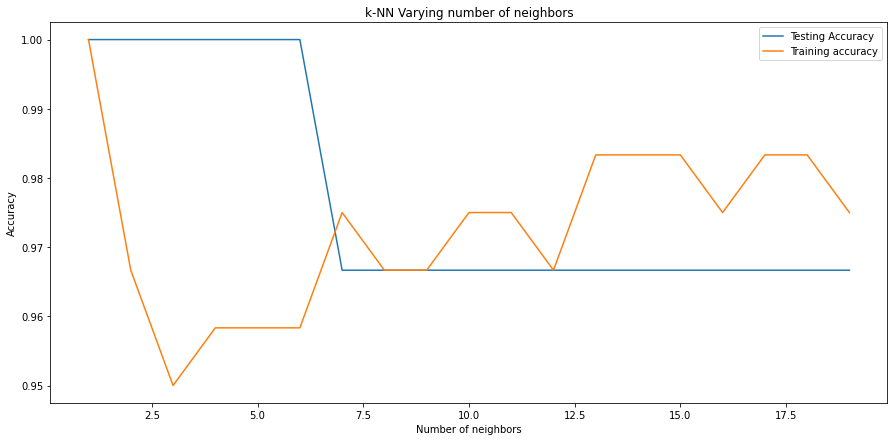
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [0 0 2 0 0 | 2 0 2 2 0 | 0 0 0 0 | 1 1 0 1 2 1 1 | 1 2 1 1 0 0 2 0 2] |
| [0 0 2 0 0 | 2 0 2 2 0 | 0 0 0 0 | 1 1 0 1 2 1 2 | 1 2 1 1 0 0 2 0 2] |

Accuracy: 96.66666666666667 %

#### Thay đổi giá trị k tìm giá trị tốt nhất

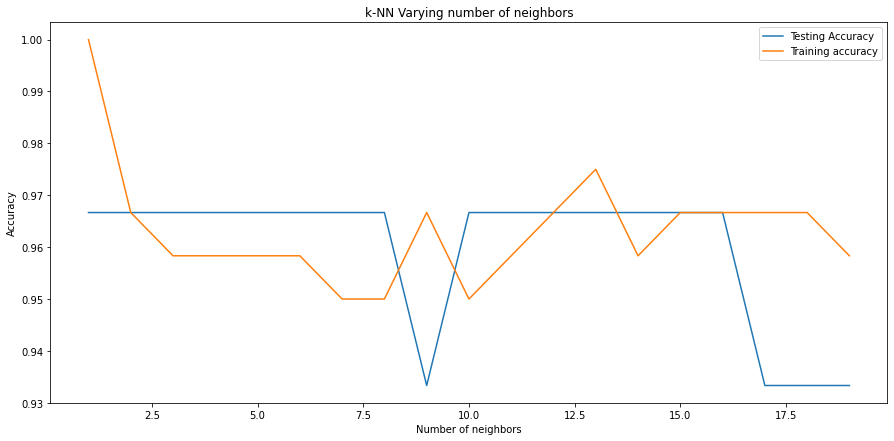
* Không có chuẩn hoá dữ liệu

draw\_varying\_neighbor(X\_train, y\_train, X\_test, y\_test)



* Có chuẩn hoá dữ liệu

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test)



* Chọn k tốt nhất k=1

# Tạo model (k = 1)

knn\_best = KNearestNeighborsClassifier(n\_neighbors=1)

# Fitting the model knn\_best.fit(X\_train\_norm, y\_train) predictions = knn\_best.predict(X\_test) print(y\_test)

print(predictions)

print("Accuracy: ", get\_accuracy(y\_test, predictions), "%")

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

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

Accuracy: 96.66666666666667 %

#### Dùng mô hình K-NN có đánh trọng số

* Đánh trọng số “distance” cho các điểm lân cận

knn\_model = KNearestNeighborsClassifier(n\_neighbors=3,

weights='distance') knn\_model.fit(X\_train\_norm, y\_train) predictions = knn\_model.predict(X\_test\_norm) print(y\_test)

print(predictions)

print("Accuracy with weight sigma: ", get\_accuracy(y\_test,

predictions), "%")

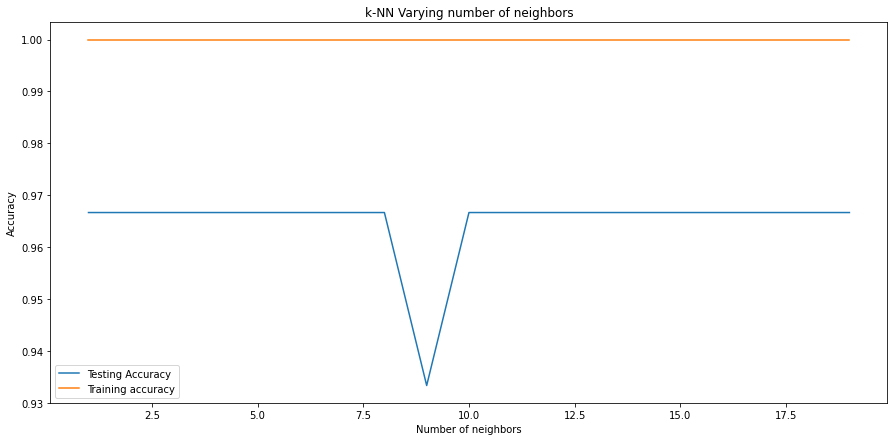
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

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

Accuracy with weight sigma: 96.66666666666667 %

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test,

weights='distance')



* Đánh trọng số "sigma" cho các điểm lân cận

*def* myweight(*distances*):

sigma2 = .5 # we can change this number return np.exp(-distances\*\*2/sigma2)

knn\_model = KNearestNeighborsClassifier(n\_neighbors=3, weights=myweight)

knn\_model.fit(X\_train\_norm, y\_train) predictions = knn\_model.predict(X\_test\_norm) print(y\_test)

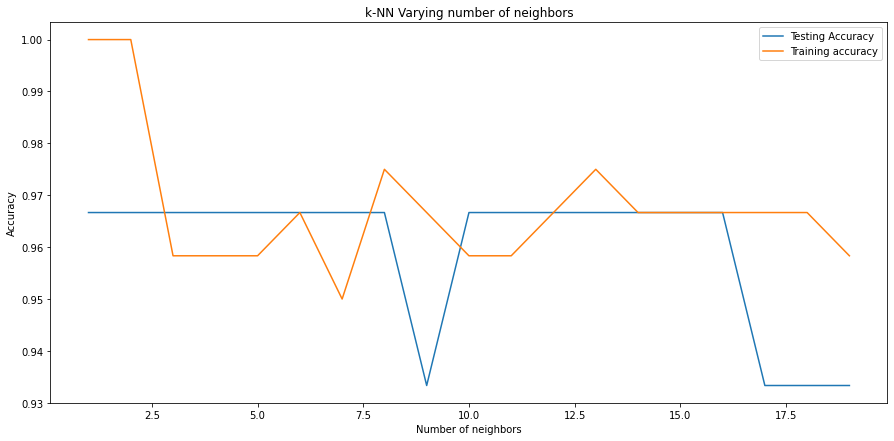
print(predictions)

print("Accuracy with weight sigma: ", get\_accuracy(y\_test, predictions), "%")

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

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

Accuracy with weight sigma: 96.66666666666667 %



weights=myweight)

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test,

### So sánh với kết quả của thư viện sklearn

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score, plot\_confusion\_matrix

#### Dự đoán kết quả tập test dùng mô hình Knn

knn\_model\_sk = KNeighborsClassifier(n\_neighbors=14) knn\_model\_sk.fit(X\_train\_norm, y\_train) predictions\_sk = knn\_model\_sk.predict(X\_test) print(y\_test)

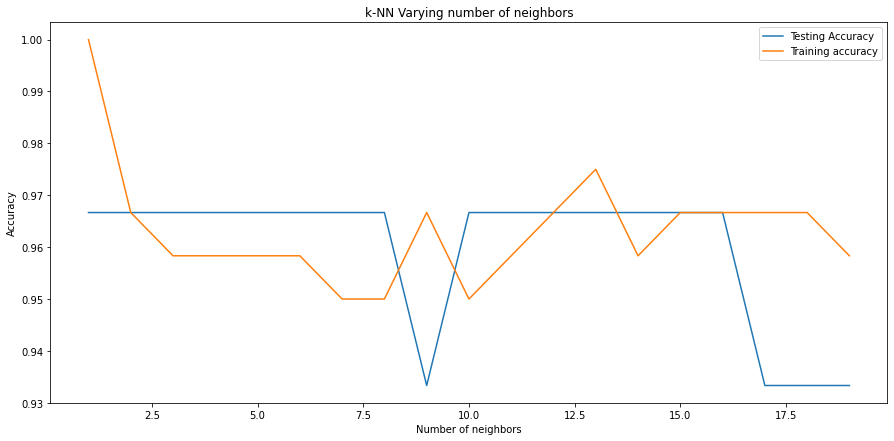
print(predictions\_sk)

print("Accuracy: ", get\_accuracy(y\_test, predictions\_sk), "%")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [0 0 2 0 0 | 2 0 2 2 0 | 0 0 0 0 | 1 1 0 1 2 1 1 | 1 2 1 1 0 0 2 0 2] |
| [0 0 2 0 0 | 2 0 2 2 0 | 0 0 0 0 | 1 1 0 1 2 1 2 | 1 2 1 1 0 0 2 0 2] |

Accuracy: 96.66666666666667 %

#### Thay đổi k tìm giá trị tốt nhất



model\_class=KNeighborsClassifier)

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test,

* Chọn k tốt nhất k = 1

# Tạo model (k = 1)

classifier = KNeighborsClassifier(n\_neighbors=1)

# Fitting the model classifier.fit(X\_train\_norm, y\_train)

# Predicting on the test set

y\_pred = classifier.predict(X\_test) print(*f*"Labels: {y\_test}") print(*f*"Predictions: {y\_pred}")

print(*f*"Accuracy is: {accuracy\_score(y\_test, y\_pred)\*100}%")

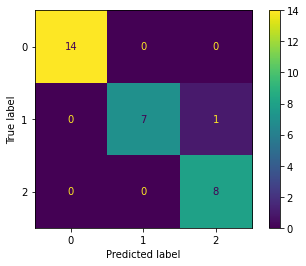
Labels: [0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

Predictions:[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]

Accuracy is: 96.66666666666667% Confusion matrix

cm = plot\_confusion\_matrix(classifier, X\_test\_norm, y\_test)

plt.show()



#### Đánh trọng số cho các điểm lân cận

* Đánh trọng số “distance” cho các điểm lân cận

knn\_model\_weight = KNeighborsClassifier(n\_neighbors = 7, p = 2,

weights 'distance') knn\_model\_weight.fit(X\_train\_norm, y\_train) y\_pred = knn\_model\_weight.predict(X\_test)

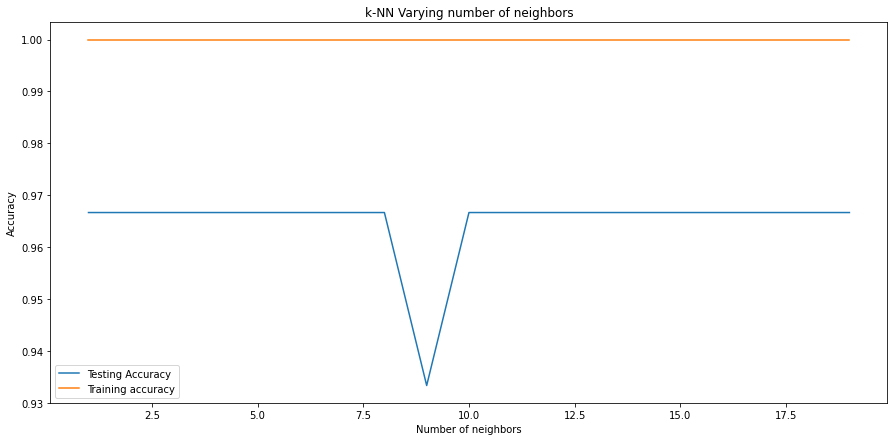
print("Accuracy of 10NN (1/distance weights): %.2f %%"

%(100\*accuracy\_score(y\_test, y\_pred)))

Accuracy of 10NN (1/distance weights): 96.67 %

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test,

model\_class=KNeighborsClassifier, weights = 'distance')



* Đánh trọng số tự chọn

*def* myweight(*distances*):

sigma2 = .5 # we can change this number return np.exp(-distances\*\*2/sigma2)

knn\_model\_custom\_weight = KNeighborsClassifier(n\_neighbors = 5, p = 2, weights = myweight)

knn\_model\_custom\_weight.fit(X\_train\_norm, y\_train) y\_pred = knn\_model\_custom\_weight.predict(X\_test\_norm)

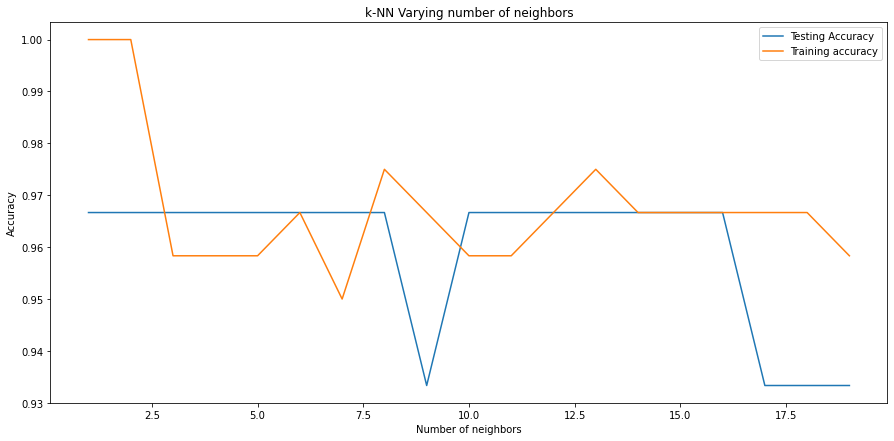
print("Accuracy of 10NN (customized weights): %.2f %%"

%(100\*accuracy\_score(y\_test, y\_pred)))

Accuracy of 10NN (customized weights): 96.67 %

draw\_varying\_neighbor(X\_train\_norm, y\_train, X\_test\_norm, y\_test,

model\_class=KNeighborsClassifier, weights = myweight)



* **Nhận xét:**

Kết quả hiện thực của nhóm giống với kết quả của thư viện sklearn.

# 10.TÀI LIỆU THAM KHẢO

1. PGS.TS Dương Tuấn Anh, Chương 3 – Phân lớp dựa trên lân cận gần nhất, Học Máy và Ứng Dụng, Khoa Khoa Học và Kỹ Thuật Máy Tính, Đại Học Bách Khoa TPHCM.
2. Vũ Hữu Tiệp, Blog Machine Learning Cơ Bản, <https://machinelearningcoban.com/2017/01/08/knn/>
3. Machine learning cơ bản (Học máy thống kê)

<https://www.youtube.com/playlist?list=PLpDNYPX7w1RYeDSr3q0EJA978jjuMz4TX>

1. Cây quyết định - Decision Tree

<https://www.youtube.com/watch?v=dc_YhOmxZ4A&list=PLpDNYPX7w1RaHzTQiqGmjatmwOuM31CnX&index=1>