Model	Cách hoạt động	Khi nào nên dùng
Linear Regression	❑ Từ tập dữ liệu numeric dataset ban đầu, sẽ tìm được phương trình tuyến tính để dự đoán đầu ra ở những điểm dữ liệu khác. ❑ Dữ liệu đầu vào và đầu ra có tính tuyến tính với nhau. ❑ Linear Regression là 1 dạng đặc biệt của Perceptron với activation function là f(x) = x. ❑ Công thức: $$y = \bar{\boldsymbol{X}} w^T$$ $\bar{\boldsymbol{X}} = [\bar{X}_1, \bar{X}_2, \bar{X}_3, \dots, \bar{X}_N] \in \mathbb{R}^{N \times D}$ $\bar{X}_i = [1, x_1, x_2, x_3, \dots, x_D] \in \mathbb{R}^{1 \times D}$ $w = [1, w_1, w_2, w_3, \dots, w_D] \in \mathbb{R}^{1 \times D}$ $y = [y_1, y_2, y_3, \dots, y_N] \in \mathbb{R}^{N \times 1}$	❑ Thường sử dụng cho bài toán **regression**. ❑ Dữ liệu có tính **linear**. ❑ Trong trường hợp dữ liệu không tuyến tính bậc 1, thì có thể cân nhắc Polynomial Regression tối đa là bậc 5 (cao hơn có thể gây overfitting). ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module **sklearn.linear_model**.
Perceptron	❑ Từ dữ liệu đầu vào có dán nhãn, model sẽ đi tìm 1 đường phẳng boundary để ngăn cách các điểm dữ liệu của 2 class ra 2 phía. ❑ Phương thức tạo ra model là từ bộ trọng số ban đầu với các phần từ gần với 0, xét từng điểm dữ liệu 1, nếu classified đúng thì không làm gì, còn nếu điểm đó bị misclassified thì sẽ cập nhật trọng số để có đường boundary mới tiến về phía làm cho điểm dữ liệu đó được classified đúng. Trong quá trình đó có thể làm những điểm dữ liệu khác bị misclassified nhưng sẽ hội tụ ở 1 thời điểm nào đó. ❑ Optimizer thường sẽ là SGD (Stochastic Gradient Descent). ❑ Trọng số weight được cập nhật như sau: $$w = w - \eta \nabla_w J(w, x_i, y_i)$$ ❑ Activation function là sgn function:	❑ Thường được sử dụng cho bài toán **binary classification** với dữ liệu đầu vào là **linearly separable**. ❑ Không thường được sử dụng phổ biến vì không phải lúc nào cũng sẽ hoàn toàn linearly separable nên sẽ ưu tiên dùng Logistic Regression hơn. ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module **sklearn.linear_model**.

	$$f(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases}$$ ❑ Perceptron có thể xem là model Artificial Neural Network (ANN) đơn giản nhất.	
Logistic Regression	❑ Tương tự như Perceptron, tìm 1 boundary (linear) để phân loại các điểm dữ liệu. Perceptron thì dựa vào kết quả của activation function là {-1,1} thì Logistic Regression sử dụng activation function là hàm sigmoid cho kết quả theo probability (0,1) để cho ra kết quả phân loại. ❑ Hàm sigmoid hoạt động đúng theo quy luật xác suất, với giá trị được rất nhỏ ban đầu thì tỉ lệ xác suất sẽ thấp, vượt qua 1 ngưỡng ban đầu thì sẽ tăng rất nhanh, đến 1 ngưỡng cao hơn nào đó thì sẽ tăng rất ít. Do đó hàm sigmoid sẽ dễ bão hòa ở 2 đầu, dễ xảy ra hiện tượng vanishing gradient. $$\sigma(x) = \frac{1}{1 + e^{-x}}$$ ❑ Dữ liệu đầu vào của Logistic Regression có thể không nhất thiết hoàn toàn linear separable. ❑ Logistic Regression là 1 dạng đặc biệt của Perceptron với activation function là sigmoid function. ❑ Thường nhạy cảm với dữ liệu của independent variable.	❑ Thường được sử dụng cho bài toán **binary classification** với dữ liệu đầu vào là **nearly linearly separable**. ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module **sklearn.linear_model**.
Softmax Regression	❑ Softmax Regression là sự kết hợp của nhiều output unit của Logistic Regression để đưa ra tỉ lệ dự đoán của các class, unit nào có probability cao nhất thì đầu ra cuối cùng sẽ kết luận rời vào class đó. ❑ Mỗi unit a_i trong activation function layer sẽ đại diện cho 1 class đưa ra tỉ lệ rơi vào class đó so với còn lại (One-vs-Rest). Activation	❑ Model được dùng cho bài toán **multi-class classification**. ❑ Softmax Regression thường được chọn là lớp **classification layer cuối cùng** của model Deep learning **Convolution**

	function sử dụng đó gọi là Softmax function. $$a_i = Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$ ❑ Hàm loss là cross-entropy function. Hàm sẽ trừng phạt dựa trên độ chênh lệch giữa 2 thành phần. Hàm sẽ trừng phạt nặng nhất với các điểm có độ chênh lệch cao nhất. Loss sẽ thấp nhất khi p = q. Vì thế sẽ giúp tối ưu để đạt được model cho kết quả dự đoán gần với groundtruth nhất có thể. $$H(p,q) = -\sum_{i=1}^{C} p_i \log(q_i)$$ ❑ Output layer của Softmax Regression được mã hóa theo dạng One-hot encoding. ❑ Boundary tạo ra bởi Softmax Regression là linear.	**Neural Network** (**CNN**). ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module **sklearn.linear_model**.
Multi-layer Perceptron	❑ MLP là 1 dạng ANN với 1 input layer, nhiều hidden layers và 1 output layer, trong khi Perceptron chỉ có 1 input layer và 1 output layer. ❑ Số lượng layers của MLP là tổng số hidden layers cộng 1 (output layer). ❑ Đầu vào của các unit trong hidden layer là $z_i^{(l)}$ và đầu ra là $a_i^{(l)}$ với output vector là $a^{(l)} \in \mathbb{R}^{d^{(l)}}$ (d: số units/layer). $$z^{(l)} = a^{(l-1)}\left(\boldsymbol{W}^{(l)}\right)^T + b^{(l)}$$ $$a^{(l)} = f(z^{(l)})$$ ❑ Trọng số $W^{(l)} \in \mathbb{R}^{d \times n^{(l)}}$ (n: số features của layer $l-1$) thể hiện	❑ MLP thường được dùng cho cả 2 dạng bài toán **classification** và **regression**. ❑ Có thể xử lý cho cả dữ liệu đầu vào **non-linear**. ❑ Thường được ưu sử dụng cho các **bài toán cơ bản không quá phức tạp**, vì các bài toán phức tạp đặc thù hơn đã có các Deep learning model tương ứng như RNN, CNN. ❑ Model sử dụng cho bài toán **Supervised learning.**

	kết nối từ layer $l-1$ đến layer l, phần từ $w_{ij}^{(l)}$ thể hiện kết nối từ node j của layer $l-1$ đến node i của layer l. ❑ Bias $b^{(l)} \in \mathbb{R}^{d^{(l)}}$ thể hiện các phần tử bias của layer l. ❑ MLP thực hiện cập nhật các trọng số weight **W** bằng backpropagation. Backpropagation là tính đạo hàm ngược từ cuối lên các trọng số. $$w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \nabla_w J(w_{ij}^{(l)})$$ ❑ Activation function là ReLU function. Hàm ReLU khắc phục được hiện tượng vanishing gradient của Sigmoid và Tanh, đồng thời có tốc độ hội tụ nhanh hơn. $$ReLU(x) = \max(0, x)$$ ❑ Hàm ReLU thường gặp hiện tượng "Dying ReLU" khi mà input đầu vào x < 0 hàm ReLU sẽ hoàn toàn bằng 0 dẫn đến không thể thực hiện tính gradient cũng như backpropagation để cập nhật trọng số ở các lớp tiếp. Do đó có biến thể khác Leaky ReLU để khắc phục hiện tượng đó. $$Leaky\ ReLU(x) = \max(\alpha x, x)\ (\alpha\ rất\ nhỏ)$$	❑ Có thể sử dụng từ module **sklearn.neural_network**.
Support Vector Machine	❑ Mục tiêu của model SVM là tìm đường boundary hoặc hyperplane trong không gian đặc trưng để phân chia dữ liệu vào các class riêng biệt, sao cho khoảng cách từ hyperplane đến các điểm dữ liệu gần nhất (margin) là cách đều và lớn nhất bằng cách tối ưu hàm loss và tối đa hóa khoảng cách đến các điểm gần nhất. ❑ Khoảng cách từ 1 điểm dữ liệu đến hyperplane cần được tối đa hóa:	❑ SVM có thể xử lý luôn cả dữ liệu **linear** và **non-linear**. ❑ SVM thường sử dụng cho bài toán **binary classifiation** và **regression**. ❑ SVM sẽ được ưu tiên sử dụng hơn Logistic Regression vì đạt được sự chính xác cao hơn.

$$distance = \frac{|xw^T + b|}{\|w\|_2}$$

- Bài toán tối ưu hóa có thể được chuyển thành bài toán tối ưu hóa lồi (convex optimization problem) bằng cách sử dụng các ràng buộc và hàm mục tiêu phù hợp:

$$w, b = \underset{w,b}{\operatorname{argmin}}(\frac{1}{2}\|w\|_2^2)$$
$$subject: y_i(x_i w^T + b) \geq 1$$

- Margin càng lớn hơn để dữ liệu chia ra được rạch ròi, phân loại tốt hơn các điểm dữ liệu mới, hạn chế tình trạng overfitting.
- Hàm loss là Hinge function. Nếu Cross-entropy loss trừng phạt những điểm ở xa biên thì, Hinge loss sẽ trừng phạt nặng những điểm bị misclassified hoặc nằm gần đường biên (boundary), việc này sẽ loại bỏ những điểm dù có classified đúng nhưng có độ chắc chắn thấp.

$$J(w, b) = \max(0, 1 - y_n z_n)$$

- Có 3 loại SVM là Hard Margin SVM, Soft Margin SVM, Kernel SVM.
- Hard Margin SVM được sử dụng trong trường hợp dữ liệu hoàn toàn linearly separable.
- Soft Margin SVM được sử dụng trong trường hợp dữ liệu không hoàn toàn linearly separable, cho phép 1 số điểm dữ liệu noise, outlier bị phân loại sai hoặc nằm gần boundary, mở rộng margin sao cho khoảng cách từ margin tới các outliers là nhỏ nhất, giúp model hoạt động linh hoạt hơn và khả năng tổng quát tốt hơn đối với dữ liệu mới.
- Kernel SVM là phương pháp mở rộng của SVM (được sử dụng phổ

- Kernel SVM thường sử dụng kernel poly cho dữ liệu tuyến tính và kernel RBF cho dữ liệu không tuyến tính.
- Model sử dụng cho bài toán **Supervised learning.**
- Có thể sử dụng từ module **sklearn.svm**.

	biến nhất) cho phép phân loại dữ liệu không tuyến tính. Kernel SVM sẽ đi tìm 1 hàm kernel phù hợp để biến đổi không gian dữ liệu ban đầu không tuyến tính thành 1 không gian linearly separable có số chiều lớn hơn hoặc vô hạn. Việc biến đổi này gọi là kernel trick. Kernel function sẽ ánh xạ dữ liệu sang chiều không gian mới. Các kernel function phổ biến là linear, poly, RBF (Radial Basis Function), sigmoid.	
K-means Clustering	❑ Mục tiêu của model là tìm ra các trung tâm cụm (centroid) sao cho trung bình khoảng cách từ mỗi điểm dữ liệu đến trung tâm cụm tương ứng là nhỏ nhất. ❑ Quy trình thực hiện: 1. Chọn k điểm ngẫu nhiên làm centrer ban đầu. 2. Gán nhãn: gán mỗi điểm dữ liệu vào cluster gần nhất bằng cách tính trung bình khoảng cách của điểm đến các center và chọn center có khoảng cách nhỏ nhất. 3. Cập nhật trung tâm: tính toán lại center của mỗi cluster bằng cách lấy trung của tất cả các điểm dữ liệu thuộc cluster tương ứng. 4. Lặp lại đến khi không có sự thay đổi lớn khi cập nhật các center và gán nhãn. ❑ k-means clustering là 1 thuật toán tìm các điểm cục bộ, nên kết quả đạt được phụ thuộc vào k điểm center chọn ban đầu. ❑ Cần chọn k cho phù hợp để đạt kết quả tốt nhất. ❑ Có thể cải thiện k-means clustering bằng cách sử dụng k-means++ hoặc k-means local search để chọn k điểm center ban đầu cho phù hợp. Hoặc có thể sử dụng kỹ thuật Elbow method (hoạt động như biểu đồ Pareto) tìm điểm k sao cho mà ở đó khi tăng thêm k thì Distortion Score (tổng bình phương khoảng cách giữa các điểm trong clusters tới center của clusters đó) cũng không có thay đổi quá lớn, không nhỏ nhất vì sẽ dẫn đến overfitting.	❑ Model thường dùng cho bài toán **clustering** các nhóm dữ liệu dựa trên sự tương đồng, mở rộng hơn là **classification**. ❑ Model nên sử dụng cho dữ liệu có tính **linear**. ❑ Model sử dụng cho bài toán **Unsupervised learning.** ❑ Có thể sử dụng từ module **sklearn.cluster**.

K-nearest Neighbor	❑ Model hoạt động dựa trên nguyên tắc là các điểm dữ liệu có thuộc tính tương tự sẽ nằm gần nhau trong không gian. ❑ Nguyên tắc hoạt động là khi có 1 điểm dữ liệu mới cần phân loại, model sẽ tìm k điểm dữ liệu gần nhất (k-nearest neighbor) trong tập dữ liệu training dựa trên 1 phép đo khoảng cách nhất định. ❑ Các điểm dữ liệu sẽ có các trọng số, thì thường sẽ có 2 loại là uniform (các điểm dữ liệu có trọng số như nhau) hoặc distance (các điểm dữ liệu được xét có khoảng cách tới điểm dữ liệu chỉ định càng ngắn thì trọng số càng lớn, khoảng cách thường được tính theo Euclid). ❑ Đối với bài toán classification, k-NN sẽ xác định k điểm gần nhất với điểm dữ liệu mới và sử dụng đa số phiếu bầu để quyết định class của điểm dữ liệu đó. Đối với bài toán regression, k-NN sẽ tính mean hoặc median của các điểm dữ liệu gần nhất để dự đoán giá trị của điểm dữ liệu mới. ❑ k-NN không khả thi với các tập dữ liệu có số chiều cao vì sẽ giảm hiệu suất tính toán. ❑ k-NN sẽ lấy k điểm gần nhất mà không quan tâm đến các điểm đó có phải là noise hay không, sẽ ảnh hưởng trực tiếp đến kết quả cuối cùng.	❑ Model dùng cho cả bài toán **classification** và **regression**. ❑ Model sử dụng cho bài toán **Unsupervised learning.** ❑ Có thể sử dụng từ module **sklearn.neighbors**.
Decision Tree	❑ Model sử dụng cấu trúc cây để đưa ra quyết định dựa trên các quy tắc học được. ❑ Quy trình chi tiết: 1. Chọn thuộc tính để làm phép phân chia cho node cây dựa trên các phương pháp đo lường. 2. Phân chia các node con dựa trên các giá trị của thuộc tính đó. 3. Lặp lại quá trình đến khi không còn thuộc tính nào để chọn hoặc tất cả các điểm dữ liệu trong node là cùng 1 class. ❑ Decision Tree có 2 loại model phổ biến là ID3 (Iterative Dichotomiser 3) và CART (Classification and Regression Tree).	❑ Model sử dụng cho cả bài toán **classification** và **regression**. ❑ Thường dùng cho các bài toán có dữ liệu chứa số thuộc tính ít, để tránh xảy ra hiện tượng overfitting. Với tập dữ liệu phức tạp hơn thì sẽ phải cân nhắc sử dụng model Random Forest để đạt kết quả tốt hơn. ❑ Có thể làm việc trên cả dữ liệu rời rạc categorical và **numeric**.

	❑ ID3 sử dụng Information Gain (Entropy Impurity) làm loss function để chọn các thuộc tính của node cây cho bài toán classification. Information Gain dễ dẫn đến tình trạng overfitting và ID3 cũng không có sử dụng biện pháp Pruning để giảm thiểu overfitting, do đó CART sẽ thường được ưu tiên sử dụng hơn. ❑ CART sử dụng Gini Gain (Gini Impurity) làm loss function để chọn các thuộc tính của node cây cho bài toán classification và MSE (Mean Square Error) cho bài toán regression. ❑ Trong bài toán classification, Entropy impurity và Gini impurity cùng là thước đo cho độ vẫn đục (impurity) hay mức độ không thuần khiết của phép phân chia. Trong đó Entropy thể hiện cho mức độ không chắc chắn trong 1 phép phân chia, còn Gini thể hiện cho mức độ không đồng nhất của phép phân chia. $$H(s) = -\sum_{i=1}^{C} \frac{N_i}{N} log \frac{N_i}{N}$$ $$H(x,s) = \sum_{k=1}^{K} \frac{m_k}{N} H(s_k)$$ $$Information\ Gain = H(s) - H(x,s)$$ $$Gini = 1 - \sum_{i=1}^{C} {p_i}^2$$ $$Gini\ Gain = Gini(p) - \sum_{k=1}^{K} \frac{m_k}{M} Gini(c_k)$$ (C: số lớp cần phân tách, K: số node con được phân tách ra, m_k: số phần tử của node con thứ k, M: số phần tử ở node p) ❑ Trong bài toán regression, MSE là thước đo cho việc chọn thuộc tính và giá trị ngưỡng để có được 1 phép chia dữ liệu. Phép chia có	❑ Model sử dụng cho bài toán **Supervised learning.** ❑ Có thể sử dụng từ module **sklearn.tree**.

	MSE nhỏ nhất sẽ được chọn cho các node cây. $$MSE = \frac{1}{N}(y - \hat{y})^2$$ ❑ Model CART có sử dụng Pruning (1 kỹ thuật Regularization) để cắt tỉa bớt hạn chế overfitting. Mục tiêu là chọn được số node tối ưu nhất và có thể dùng k-fold cross-validation (Grid Search, Randomized Search) để tìm ra tham số α phù hợp nhất. $$R_\alpha(T) = L + \alpha\|T\|$$	
Random Forest	❑ Model Random Forest là 1 tập hợp của các model Decision Tree, kết hợp kết quả từ các cây để có kết quả dự đoán cuối cùng. ❑ Quy trình thực hiện: 1. Lấy mẫu ngẫu nhiên theo quy tắc Bootstrapping: Từ tập training có kích thước N, lấy mẫu ngẫu nhiên từ tập dữ liệu để tạo ra các tập con (subset) có kích thước M (M < N). Việc này thực hiện bằng cách chọn ngẫu nhiên M điểm dữ liệu gốc mà có thể được lặp lại (sampling with replacement). 2. Xây dựng mode Decision Tree: tạo ra các model Decision Tree từ mỗi tập con với các feature ngẫu nhiên. 3. Dự đoán theo quy tắc Ensembling: khi có 1 điểm dữ liệu mới cần dự đoán, Random Forest sử dụng tất cả Decision Tree đã xây dựng trước đó để đưa ra dự đoán. Đối với bài toán classification, dự đoán được ra bằng cách thực hiện voting. Đối với bài toán regression, dự đoán được đưa ra bằng cách lấy giá trị mean của tất cả các Decision Tree. ❑ Random Forest giảm thiểu tình trạng overfitting của Decision Tree, có tính linh hoạt cao, hạn chế sự ảnh hưởng của các outliers và giảm được phương sai của kết quả dự đoán cuối cùng.	❑ Model sử dụng cho cả bài toán **classification** và **regression**. ❑ Có thể làm việc trên cả dữ liệu rời rạc categorical và **numeric**. ❑ Model sử dụng cho bài toán **Supervised learning.** ❑ Có thể sử dụng từ module **sklearn.ensemble**.

Singular Value Decomposition	❑ Model SVD (Model phân tích giá trị suy biến) sẽ chuyển 1 ma trận bất kỳ thành các thành phần chính giúp ta hiểu rõ hơn về cấu trúc và thông tin chứa đựng trong dữ liệu. $$A_{m\times n} = U_{m\times m} \times S_{m\times n} \times {V_{n\times n}}^T$$ U: ma trận trực giao, các cột trong ma trận là các vector riêng trái (left singular vectors) của ma trận A, thể hiện sự tương quan giữa các hàng của ma trận A. Cụ thể hơn sẽ thể hiện được sự tương quan giữa các mẫu hoặc thuộc tính trong dữ liệu gốc. $$U = eigenvector\ of\ (AA^T)$$ S: ma trận đường chéo, các phân tử trên đường chéo chính là các giá trị suy biến (singular values), với thứ tự giảm dần từ trên xuống. Cho biết mức độ quan trọng của các thành phần chính tương ứng. $$S = \sigma = \sqrt{eigenvalue}$$ V^T: ma trận trực giao và là ma trận chuyển vị của ma trận V, các cột trong ma trận là các vector riêng phải (right singular vectors) của ma trận A, thể hiện sự tương quan giữa các cột của ma trận A. Cụ thể hơn thể hiện thuộc tính hoặc từ trong dữ liệu. $$V = eigenvector\ of\ (A^TA)$$ Ma trận trực giao là ma trận mà các thành phần trong đó có độ lớn khác 0 và tích vô hướng giữa các thành phần là bằng 0. Quy trình tính toán có thể như sau (nếu m < n thì đổi $B = A^T$ và tính toán tương tự dựa trên A):	❑ Thường được dùng khi muốn phân tích ma trận dữ liệu để **trích xuất các đặc trưng quan trọng**. Ngoài ra còn dùng để giảm chiều dữ liệu, nén dữ liệu/ảnh, giảm nhiễu ảnh. ❑ Ma trận U thường được dùng để **giảm chiều dữ liệu**, **tái tạo lại dữ liệu xấp xỉ** dữ liệu gốc hoặc **tìm kiếm các mẫu tương tự**. ❑ Ma trận V thường được dùng khi làm việc với **dữ liệu văn bản** muốn **tìm kiếm các từ khóa quan trọng**. ❑ Thuộc về bài **Dimentionality Reduction**. ❑ Model sử dụng cho bài toán **Unsupervised learning.** ❑ Có thể sử dụng từ module **sklearn.decomposition** (Truncated SVD – SVD đã giảm k giá trị suy biến) hoặc **numpy.linalg** (SVD giữ lại toàn bộ giá trị suy biến).

	1. Tính eigenvector và eigenvalue của A^TA (right): $$det(A^TA - \lambda I) = 0 \rightarrow \lambda \rightarrow \sigma$$ $$A^TAv_i = \lambda_i v_i \rightarrow v$$ 2. Tính eigenvector của AA^T (left): $$Av_i = \sigma_i u_i \ (\sigma_i = \sqrt{\lambda_i}) \rightarrow u$$ ❑ Khi ma trận A lớn thì việc tính toán SVD sẽ tốn nhiều thời gian.	
Principal Component Analysis	❑ Mục tiêu của PCA là đi tìm kiếm các thành phần chính quan trọng trong dữ liệu để biểu diễn các dữ liệu ban đầu thành 1 số thành phần mới có chiều thấp hơn, hay nói cách khác là đi tìm 1 hệ trực giao **U** sao cho trong hệ này, các thành phần quan trọng nhất trong K thành phần đầu tiên. $$X = U_K Z + \overline{U}_K Y$$ Ta sẽ lượt bỏ đi phần $\overline{U}_K Y$ không quan trọng. ❑ Quy trình thực hiện: 1. Chuẩn hóa dữ liệu đầu vào: Trừ dữ liệu đầu vào với vector kỳ vọng. $$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$ $$\hat{x} = x - \bar{x}$$ 2. Tính toán ma trận covariance: ma trận covariance cho biết độ tương quan giữa các điểm dữ liệu.	❑ Thường được dùng khi chỉ muốn tập trung vào **việc giảm chiều dữ liệu** hoặc **trực quan hóa** dữ liệu. ❑ PCA cũng có thể ứng dụng để tính **góc xoay 2D** của vật thể với đầu vào là các điểm dữ liệu (contour) trên 2 trục x-y dựa trên eigenvector và eigenvalue. ❑ Thuộc về bài **Dimentionality Reduction**. ❑ Model sử dụng cho bài toán **Unsupervised learning**. ❑ Có thể sử dụng từ module **sklearn.decomposition**.

$$C = \frac{1}{N}\hat{X}^T\hat{X}$$

3. Tính eigenvector và eigenvalue của ma trận covariance: eigenvector cho biết hướng (phương) của dữ liệu trong không gian, eigenvalue cho biết độ biến đổi của dữ liệu.

$$Cv = \lambda v$$

4. Sắp xếp các eigenvector và eigenvalue: sắp xếp theo thứ tự giảm dần, đảm bảo các thành phần chính quan trọng nhất được xác định trước.
5. Lựa chọn số thành phần chính: dựa trên eigenvalue để chọn số thành phần chính, thường thì sẽ chọn K sao tổng eigenvalue được giữ lại 90%.
6. Xác định ma trận trọng số: lấy các eigenvector tương ứng với số thành phần chính đã chọn và xếp thành các cột của ma trận W.

$$T = XW$$

Với T là "score" – kết quả của việc chiếu dữ liệu ban đầu X vào không gian thành phần chính, mỗi hàng tương ứng với 1 điểm dữ liệu mới, mỗi cột là 1 thành phần chính. W là "loadings" – các eigenvectors tương ứng với các thành phần chính, thể hiện mức độ mà mỗi biến trong dữ liệu ban đầu đóng góp vào thành phần chính.

- Ngoài ra PCA có thể được thực hiện dựa trên SVD để bỏ qua bước tính toán ma trận Covariance:
  1. Tính toán SVD của ma trận dữ liệu X để tìm ma trận U,S,V.

	$$X = U \times S \times V^T$$ 2. Lựa chọn K giá trị suy biến (singular values) muốn giữ lại. 3. Các cột trong ma trận U tương ứng với các thành phần chính. $$T = XW = (U \times S \times V^T) \times V = U \times S$$ Ma trận W có thể coi như tương tự với ma trận V vì W là eigenvector của ma trận $C = A^T A$ và V là singular vector của $A^T A$, singular value là căn bậc 2 của eigenvalue, tỉ lệ đó tương tự với eigenvector và singular vector. ❑ Các thành phần chính tìm được PCA không có sự tương quan tuyến tính lớn để có thể tách riêng và xem xét chúng 1 cách độc lập, từ đó có thể phân cụm (clustering) để ứng dụng trong việc trực quan hóa dữ liệu trong đồ thị 2D. ❑ Thành phần chính lớn nhất là thành phần có covariance lớn nhất, và cứ thế giảm dần.	
Linear Discriminant Analysis	❑ Mục tiêu của LDA là đi tìm phép biến đổi tuyến tính sao cho dữ liệu được biểu diễn trong không gian mới có độ tách rời giữa các lớp (between-class variance) là lớn nhất và đồng thời giữ lại được càng nhiều thông tin phân biệt giữa các lớp càng tốt (within-class variance nhỏ nhất). ❑ Số chiều dữ liệu mới nhỏ hơn $C - 1$ (C: là số lượng class). ❑ Quy trình thực hiện: 1. Chuẩn bị dữ liệu: sắp xếp dữ liệu thành các class riêng biệt và tính vector kỳ vọng của các class. $$e_k = \frac{1}{N_k} \sum_{n \epsilon C_k} x_n$$	❑ LDA là phương pháp **giảm chiều dữ liệu** cho bài toán **classification**, tức là giảm chiều sao cho việc phân lớp hiệu quả nhất. ❑ Thường được sử dụng với dữ liệu **linearly separable**. ❑ Thuộc về bài **Dimentionality Reduction**. ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module

sklearn.discriminant_analysis.

2. Tính within-class covariance matrix: các điểm sau khi chiếu trong không gian mới có phương sai đến điểm kỳ vọng của class tương ứng là nhỏ thể hiện dữ liệu ít bị phân tán, có xu hướng giống nhau trong cùng 1 class.

$$\sigma_k^2 = \sum_{n \epsilon C_k} \|y_n - e_k\|_F^2 = \|Y_k - E_k\|_2^2$$

$$s_w = \sum_{k=1}^{C} \sigma_k^2 = Trace(\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W})$$

$$\boldsymbol{S}_W = \sum_{k=1}^{C} \sum_{n \epsilon C_k} (x_n - m_k)(x_n - m_k)^T$$

Với σ_k^2 và $\boldsymbol{S}_W$ là within-class covariance của class k và within-class covariance matrix.

3. Tính between-class covariance matrix: để các class tách rời nhau riêng biệt lớn thì kỳ vọng của mỗi class xa vector kỳ vọng chung là lớn.

$$s_b = \sum_{k=1}^{C} N_k \|e_n - e\|_F^2 = Trace(\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W})$$

$$\boldsymbol{S}_B = \sum_{k=1}^{C} (m_k - m)(m_k - m)^T$$

Với $\boldsymbol{S}_B$ là between-class covariance matrix.

4. Tối ưu hàm: tìm 1 phép biến đổi tuyến tính W sao cho hàm tối ưu J(W) là lớn nhất (between-class variance lớn nhất và

	within-class variance nhỏ nhất). $$J(W) = \frac{s_b}{s_w} = \frac{Trace(\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W})}{Trace(\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W})}$$ $$\rightarrow \boldsymbol{W} = \underset{\boldsymbol{W}}{\text{argmax}} \frac{Trace(\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W})}{Trace(\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W})}$$ $$\nabla_{\boldsymbol{W}} J(\boldsymbol{W}) = 0$$ $$\rightarrow \boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{W} = J\boldsymbol{W}$$ Từ đó có thể thấy được **W** là eigenvector của $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$ ứng với eigenvalue lớn nhất bằng với J. Các cột trong ma trận **W** là phải độc lập tuyến tính. 5. Giảm chiều dữ liệu: chọn K thành phần chính (linear discriminants) tương ứng với K giá trị lớn nhất của J(W). Chiếu dữ liệu vào không gian mới có K chiều để thực hiện phân loại hoặc trích xuất đặc trưng. $$\boldsymbol{y} = X\boldsymbol{W}^T$$ ❑ LDA có giả sử ngầm rằng dữ liệu của các class đều tuân theo phân phối chuẩn và covariance matrix của các classes là gần nhau. ❑ LDA hoạt động tốt với dữ liệu là linearly separable và sẽ giảm hiệu quả rõ rệt với dữ liệu không linearly separable.	
Naive Bayes Classifier	❑ NBC hoạt động dựa trên quy tắc xác suất Bayes để dự đoán phân loại class. ❑ Mục tiêu của model là tính xác suất có điều kiện (Condition Probability) của một class dựa trên các đặc trưng của mẫu dữ liệu. Các đặc trưng được thuật toán giả định có tính độc lập với nhau, không phụ thuộc.	❑ Thường được dùng trong các bài toán **classification**, chủ yếu là **phân loại văn bản** và **lọc email spam**. ❑ Thường sử dụng khi dữ liệu chứa các **đặc trưng có tính độc**

$$P(class|Feature) = \frac{P(Features|Class) \times P(Class)}{P(Features)}$$

- Các loại phân phối thường dùng:
  - Gaussian Naive Bayes: thường được sử dụng cho dữ liệu có tính liên tục.

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

Bộ tham số $\theta = \{\mu_y, \sigma_y^2\}$ được xác định theo quy tắc Maximum Likelihood estimation (tìm giá trị tham số sao cho xác suất xảy ra của 1 class là cao nhất):

$$\{\mu_y, \sigma_y^2\} = \underset{\mu_y, \sigma_y^2}{\operatorname{argmax}} \prod_{n=1}^{N} p(x_i|\mu_y, \sigma_y^2)$$

  - Multinomial Naive Bayes: thường sử dụng cho dữ liệu văn bản với feature vector theo quy tắc Bag of Words, với vector có độ dài n là số từ trong dictionary, từng giá trị trong vector là thể hiện số lần xuất của mỗi từ trong văn bản. Trong trường hợp, xuất hiện từ không có sẵn trong dict từ trước thì để hạn chế trường hợp xác suất bằng 0 (tử số bằng 0) thì sẽ thêm tham số α (thường là bằng 1) trên tử và αn ở dưới mẫu để đảm bảo tổng xác suất các từ (feature) trong 1 class là bằng 1.

$$P(x_i|y) = \frac{N_{y_i} + \alpha}{N_y + \alpha n}$$

lập với nhau (độ phụ thuộc thấp).
- Hoạt động tốt hơn với tập dữ liệu **nhỏ** so với model **SVM**.
- Model sử dụng cho bài toán **Supervised learning**.
- Có thể sử dụng từ module **sklearn.naive_bayes**.

	▪ Bernoulli Naive Bayes: thường áp dụng cho dữ liệu dạng binary (0 hoặc 1) thì thay vì với Multinomial là số lần xuất hiện của mỗi từ thì sẽ là từ đó có xuất hiện hay không. $$P(x_i\|y) = p(i\|y)^{x_i}(1 - p(i\|y))^{1-x_i}$$ ❑ NBC có thời gian training và testing rất nhanh, do việc giả sử trước về tính độc lập của dữ liệu. Nếu tính độc lập thật sự được thõa mãn thì có thể đạt được kết quả tốt hơn SVM và Logistic Regression với tập dữ liệu nhỏ.	
eXtreme Gradient Boosting	❑ XGBoost là 1 model cải tiến của thuật toán Gradient Boosting. ❑ Model hoạt động theo cơ chế boosting (học tăng cường) bằng cách xây dựng các Decision Tree yếu (weak Decision Tree) tuần tự, với từng cây sẽ được tạo ra để cải thiện dựa trên prediction error của cây trước. ❑ Quy trình thực hiện tạo ra 1 cây: 1. Khởi tạo cây ban đầu chỉ chứa 1 node gốc với prediction ban đầu thường là 0.5 dù là classification hay regression. 2. Chọn thuộc tính phù hợp: model thực hiện tính toán Gain dựa Similarity Score của từng node. Với node gốc ban đầu, thì sẽ là giá trị trung bình (regression) hoặc 0.5 (classification). Thuộc tính có Gain cao nhất sẽ cho phép chia tốt nhất. ▪ Regression: $$Similarity\ Score = \frac{(\sum residual)^2}{N + \lambda}$$ ▪ Classification: $$Similarity\ Score = \frac{(\sum residual)^2}{\sum Previous\ P_i \times (1 - Previous\ P_i) + \lambda}$$	❑ Có thể sử dụng cho cả bài toán **classification** (**binary** và **multi-class**) và **regression**. ❑ Thường được ưu tiên sử dụng phổ biến vì đạt được độ chính xác cao với khả năng hạn chế overfitting. ❑ Model sử dụng cho bài toán **Supervised learning**. ❑ Có thể sử dụng từ module **xgboost**.

3. Tạo các nhánh con: thực hiện kiểm tra giá trị Gain đã thõa điều kiện ngưỡng (gamma hoặc cover value) để có thể thực hiện chia các nhánh con.
4. Kết quả của các leaf node:

- Regression:

$$Output\ value = \frac{\sum residual}{N + \lambda}$$

- Classification:

$$Output\ value = \frac{\sum residual}{\sum Previous\ P_i \times (1 - Previous\ P_i) + \lambda}$$

5. Đưa ra dự đoán:

- Regression:

$$New\ prediction = Origin\ prediction + \eta \times Output\ \ value$$

- Classification:

$$Log(odds) = \log\left(\frac{p}{1-p}\right)$$

$$Log(odds)\ prediction = Log(odds)\ origin + \eta \times Output\ \ value$$

$$New\ prediction = \frac{e^{\log(odds) prediction}}{1 + e^{\log(odds) prediction}}$$

6. Tính residual (error) mới để tạo ra cây mới tiếp theo để cải thiện residual mới có được.

- XGBoost sử dụng regularization (L1 và L2) cho các trọng số của cây để hạn chế overfitting, ngoài ra còn có tham số learning rate η, việc điều chỉnh tham số này sẽ ảnh hưởng khả năng đóng góp của mỗi cây trong quá trình training, cụ thể là khả năng điều chỉnh cải thiện áp dụng đối với mỗi cây, learning rate thấp sẽ giảm tốc độ hội tụ lại, tạo ra các cây đơn giản hơn.
- Đối với việc pruning, với bài toán regression, XGBoost còn có tham số gamma để xác định mức tối thiểu để pruning cây, khi Gain của hàm loss tại mỗi node không đạt được mức độ tối thiểu (gamma γ) sẽ bị cắt tỉa (prune) để hạn chế sự phức tạp của cây. Cộng hưởng với tham số Regularization (lambda λ), thì sẽ giúp dễ cắt tỉa cây vì nó sẽ giảm Gain đi và dưới giới hạn ngưỡng. Với bài toán classification, thì giá trị Cover sẽ được tính toán, nếu Gain thấp hơn ngưỡng Cover thì sẽ bị cắt tỉa đi.

$$Cover\ value = \sum Previous\ P_i \times (1 - Previous\ P_i)$$

- Các model Decision Tree của XGBoost được xây dựng dựa trên thuật toán Approximate Greedy Algorithm khi chọn thuộc tính hay ngưỡng cho node. Khi gặp tập dữ liệu dataset lớn, thay vì xem xét trên toàn bộ các điểm chia có thể có của tập training, XGBoost chỉ xem xét trên tập con (quantiles) các điểm chia tiềm năng. Cụ thể, dataset được tính toán biểu diễn thành biểu đồ histogram xấp xỉ, sau đó được phân chia thành các quantiles xấp xỉ, chứa đồng đều nhau các điểm dữ liệu có gắn trọng số (weighted quantiles). Điều này giúp tăng tốc quá trình training với dataset lớn.
- XGBoost hỗ trợ training song song trên nhiều máy tính, CPU và cả GPU, do đó có tốc độ training rất nhanh so với các model khác.

Q-Learning	❑ Q-learning là 1 thuật toán on-policy learning. ❑ Mục tiêu của model là hoàn thành Q table chứa các giá trị tính được từ Q-function tương ứng với các cặp state-action (s,a) trong quá trình tương tác với môi trường, để sau đó agency có thể sử dụng Q table để đưa ra action phù hợp nhất theo 1 policy nào đó trước 1 state nhất định. ❑ Q table trong Q learning là 1 bảng có kích thước $m \times n$, với m, n lần lượt là kích thước của không gian trạng thái (observation space) và không gian hành động (action space) của môi trường, tương ứng với số state và action có được trong không gian. Mỗi 1 ô đại diện cho giá trị Q-function của 1 cặp state-action. Giá trị ban đầu của Q-table có thể là ngẫu nhiên hoặc khởi tạo cố định, các giá trị trong đó sẽ được cập nhật trong quá trình agency tương tác với môi trường dựa trên các điểm thưởng/phạt (reward) tương ứng với action đã thực hiện theo công thức của Q-function dựa trên thuật toán Temporal difference learning (TD): $$Q(a,s) = Q(a,s) \times (1-\alpha) + \alpha \times (r + \gamma \times Q(s',a))$$ $Q(a,s)$: Giá trị Q-function của cặp (s,a). $Q(s',a)$: Giá trị Q-function của cặp (s',a). α: learning rate – mức độ cập nhật giá trị Q-function. r: reward – điểm thưởng/phạt nhận được từ mỗi action. γ: discount factor – mức độ ưu tiên của việc đạt được kết quả hay reward cao hơn ở tương lai hay hiện tại hơn. ❑ Quá trình thực hiện: 1. Khởi tạo Q table ban đầu với các giá trị ngẫu nhiên toàn bộ cặp state-action có thể trong môi trường. 2. Chọn 1 action a ở state s hiện tại dựa trên policy bất kỳ. 3. Thực hiện action a và nhận reward r tương ứng từ môi trường. 4. Chuyển đến state mới s'.	❑ Thường được sử dụng cho các vấn đề khi có 1 hệ thống môi trường để tương tác và học như **game**, **robotics**, **tài chính**, **xe tự hành**. ❑ Thường sử dụng cho môi trường **không biết trước thông tin**, có không gian **trạng thái nhỏ đến vừa**, đồng thời **rời rạc**, không liên tục. ❑ Model sử dụng cho bài toán **Reinforcement learning**.

5. Cập nhật giá trị Q-function với cặp state-action (s, a).
6. Thực hiện vòng lặp tương tự cho các state tiếp theo đến khi kết thúc số vòng lặp chỉ định hoặc đạt được mục tiêu của môi trường.

- Action của state có thể được chọn dựa trên Greedy policy hoặc Epsilon-greedy policy. Với Greedy policy sẽ chọn action có giá trị Q-function cao nhất ở state hiện tại, đồng nghĩa là luôn chọn hành động tối ưu với mục đích tối đa hóa phần thưởng. Với Epsilon-greedy policy thay vì chỉ tập trung vào việc chọn hành động tối ưu thì sẽ có thêm tỉ lệ ε để thực hiện ngẫu nhiên khác để khám phá thêm để có thể có được các action mới tối ưu hơn.

$$Chosen\ action = \begin{cases} random\ action\ a', & p \le \varepsilon \\ argmax\big(Q(s,a)\big), & p > \varepsilon \end{cases}$$

Với p là 1 xác suất ngẫu nhiên có giá trị (0,1) ở mỗi state.

- Hệ số epsilon ε thường sẽ không cố định mà sẽ cao ban đầu để kích thích khám phá (exploration) và giảm dần về sau khi đã khám phá đủ để thực hiện khai thác (exploitation) dựa trên những gì đã học được (prior experience). Thường thì có thể thực hiện dựa trên các phương pháp Decaying epsilon khác nhau.

$$\varepsilon_{t+1} = \varepsilon_{min} + e^{-decay\_rate \times t} \times (\varepsilon_{max} - \varepsilon_{min})$$

$$\varepsilon_{t+1} = \frac{\varepsilon_t}{1 + decay\_rate^t}$$

$$\varepsilon_{t+1} = \varepsilon_t \times decay\_rate^t$$

Deep Q-Network	❑ DQN là 1 thuật toán off-policy learning. ❑ DQN có cách thức hoạt động giống như thuật toán Q-learning dựa vào state hiện tại để đưa ra action tối ưu nhất nhưng thay vì sử dụng Q-table chứa các giá trị Q-value tương ứng với các cặp state-action thì sẽ sử dụng mạng nueral sâu (Deep Neural Network) với đầu vào là state để dự đoán xấp xỉ giá trị Q-value tương ứng với từng action. ❑ Quy trình thực hiện: 1. Cho agency tương tác với môi trường và lưu trữ 1 số lượng nhất định các tập hợp experience {current state, reward, current action, next state} vào replay memory. 2. Sau đó cứ sau mỗi 1 số lượng step nhất định sẽ lấy ra n-batch từ replay memory để cập nhật main model dựa trên công thức: $$y_i = \begin{cases} r_i, & terminate \\ r + \gamma \times arg\max_{a'} Q(s', a', \theta_{i-1}), & non-terminate \end{cases}$$ $$Loss = (y_i - Q(s, a, \theta_i))^2$$ Trong đó θ_{i-1} là dùng bởi target model và θ_i là dùng bởi main model. 3. Và cứ sau mỗi 1 lượng rất lớn step sẽ thực hiện cập nhật target model bằng main model.	❑ Thường sử dụng cho môi trường **không biết trước thông tin**, có không gian **trạng thái lớn** (khắc phục được khuyết điểm của Q-learning), đồng thời **rời rạc**, hoặc liên tục. ❑ Model sử dụng cho bài toán **Deep Reinforcement learning**.