TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐẠI HỌC QUỐC GIA
THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

HỌC MÁY

HỌC KỲ 242 NĂM HỌC 2024-2025

CẢI THIỆN DECISION TREE VÀ LINEAR SVC BẰNG BAGGING VÀ BOOSTING

KHOA HỌC MÁY TÍNH

GVHD: Võ Thanh Hùng

——o0o——

SINH VIÊN 1: Nguyễn Hồng Minh 2212059

SINH VIÊN 2: Nguyễn Minh Triết 2213608

SINH VIÊN 3: Nguyễn Gia Nguyên 2212303

TP. Hồ Chí Minh, tháng 4 năm 2025

Contents

Chapter 1

Mở đầu

Học máy (machine learning) là một nhánh quan trọng của trí tuệ nhân tạo, tập trung vào việc phát triển các thuật toán có khả năng học hỏi từ dữ liệu để đưa ra dự đoán hoặc quyết định mà không cần lập trình rõ ràng. Trong học máy, nhiệm vụ phân loại là một trong những ứng dụng phổ biến nhất, với mục tiêu gán nhãn cho các đầu vào dựa trên các đặc trưng của chúng. Hai phương pháp phân loại được sử dụng rộng rãi là Cây quyết định (Decision Tree) và Máy vector hỗ trợ (Support Vector Machine - SVM), nhờ vào tính hiệu quả và khả năng áp dụng trong nhiều bài toán thực tiễn.

1.1 Động lực đề tài

Việc cải thiện hiệu suất của các mô hình học máy có ý nghĩa to lớn trong nhiều lĩnh vực ứng dụng, từ y tế, tài chính, đến phân tích hành vi, nơi mà độ chính xác và độ tin cậy của dự đoán có thể ảnh hưởng trực tiếp đến các quyết định quan trọng. Do đó, việc nghiên cứu và áp dụng các phương pháp ensemble như Bagging và Boosting là một hướng đi cần thiết để xây dựng các mô hình học máy mạnh mẽ và đáng tin cậy hơn.

1.2 Mục tiêu

Chúng tôi tập trung vào việc cải thiện hiệu suất của Cây quyết định và SVM thông qua việc áp dụng các phương pháp Bagging và Boosting. Để đánh giá hiệu quả của các phương pháp này, chúng tôi sử dụng một tập dữ liệu về hành vi bầy đàn của côn trùng

như một ví dụ thực nghiệm. Tập dữ liệu này đóng vai trò là công cụ kiểm nghiệm, cho phép chúng tôi so sánh hiệu suất của các mô hình ensemble với các mô hình cơ bản, từ đó đánh giá mức độ cải thiện về độ chính xác và độ ổn định. Ngoài ra, chúng tôi cũng thực hiện các kỹ thuật lựa chọn đặc trưng, bao gồm ngưỡng phương sai, thông tin tương hỗ, bộ lọc tương quan, và chính quy hóa L2 nhúng, để tối ưu hóa tập đặc trưng đầu vào cho các mô hình.

Mục tiêu chính của dự án là làm sáng tỏ cách mà các phương pháp ensemble có thể nâng cao hiệu quả của Cây quyết định và SVM, đồng thời cung cấp cái nhìn sâu sắc về vai trò của lựa chọn đặc trưng trong việc cải thiện hiệu suất mô hình. Thông qua việc so sánh các mô hình cơ bản và các phiên bản ensemble, chúng tôi hy vọng sẽ chứng minh được tiềm năng của Bagging và Boosting trong việc giải quyết các bài toán phân loại phức tạp.

1.3 Phạm vi nghiên cứu

Các phương pháp học máy như Cây quyết định và SVM được xem xét để phân tích và giải quyết các bài toán phân loại. Cây quyết định chia dữ liệu thành các tập con dựa trên giá trị đặc trưng, tạo cấu trúc cây nhằm hỗ trợ ra quyết định, nổi bật bởi tính dễ hiểu và khả năng diễn giải. Tuy nhiên, phương pháp này dễ gặp vấn đề quá khớp khi xử lý dữ liệu phức tạp, làm giảm hiệu quả trên dữ liệu mới. Trong khi đó, SVM tìm kiếm siêu phẳng tối ưu để phân tách các lớp trong không gian đặc trưng, đạt hiệu suất cao nhưng đòi hỏi tài nguyên tính toán lớn và phụ thuộc vào lựa chọn kernel, gây khó khăn trong một số ứng dụng.

Để nâng cao hiệu quả, nghiên cứu tập trung vào các phương pháp ensemble như Bagging và Boosting, vốn cải thiện độ chính xác và ổn định của mô hình. Bagging giảm phương sai thông qua huấn luyện nhiều mô hình trên các tập con dữ liệu ngẫu nhiên, kết hợp dự đoán như trong Random Forest. Ngược lại, Boosting giảm độ chệch bằng cách huấn luyện tuần tự các mô hình, tập trung sửa lỗi của mô hình trước, như trong AdaBoost hoặc Gradient Boosting. Các phương pháp này được đánh giá để xác định phạm vi áp dụng và hiệu quả trong các bài toán thực tiễn, đóng vai trò quan trọng trong việc tối ưu hóa mô hình học máy.

1.4 Cấu trúc đề tài

Báo cáo này được tổ chức như sau:

- Phần 2 trình bày các phương pháp lựa chọn đặc trưng, bao gồm ngưỡng phương sai, thông tin tương hỗ, bộ lọc tương quan, và L2 nhúng.
- Phần 3 thảo luận về quá trình lựa chọn, huấn luyện, và điều chỉnh các mô hình, bao gồm Cây quyết định, SVM, và các phiên bản ensemble.
- Phần 4 trình bày đánh giá thực nghiệm, so sánh hiệu suất của các mô hình dựa trên các chỉ số như độ chính xác, F1-score, và AUC.

Chapter 2

Cơ sở lý thuyết

Phần này cung cấp nền tảng lý thuyết cho các thuật toán và kỹ thuật được sử dụng trong dự án, bao gồm Cây quyết định (Decision Trees), Máy vector hỗ trợ (Support Vector Machines - SVM), và các phương pháp ensemble như Bagging và Boosting. Ngoài ra, chúng tôi cũng thảo luận về các phương pháp lựa chọn đặc trưng được áp dụng để tối ưu hóa dữ liệu đầu vào. Những khái niệm này là cơ sở để hiểu cách các mô hình được xây dựng, cải thiện, và đánh giá trong dự án.

2.1 Decision tree

Decision Tree [16][18] là một mô hình học máy thuộc nhóm mô hình học có giám sát (supervised learning), được sử dụng cho cả bài toán phân loại (classification) và hồi quy (regression). Mô hình này dựa trên việc phân tách dữ liệu đầu vào thành các nhánh (branches) theo giá trị của các thuộc tính (features), với mục tiêu tạo ra các nhóm dữ liệu có độ đồng nhất cao nhất có thể tại các nút lá (leaf nodes).

Giải thuật xây dựng cây quyết định được sử dụng là một phiên bản tối ưu hơn của giải thuật CART [4], và do thư viện scikit-learn từ Python hiện thực. Thư viện scikit-learn mô tả toán học giải thuật như bên dưới.

Cho các vector huấn luyện $\mathbf{x}_i \in \mathbb{R}^n, i = 1, \ldots, l$ và một vector nhãn $\mathbf{y} \in \mathbb{R}^l$, một cây quyết định sẽ phân vùng không gian đặc trưng một cách đệ quy sao cho các mẫu có cùng nhãn hoặc giá trị mục tiêu tương tự được nhóm lại với nhau.

Gọi dữ liệu tại nút m được biểu diễn bởi Q_m với n_m mẫu. Đối với mỗi phân chia ứng

viên $\theta = (j, t_m)$ bao gồm một đặc trưng j và ngưỡng t_m, phân chia dữ liệu thành các tập con $Q_m^{\text{left}}(\theta)$ và $Q_m^{\text{right}}(\theta)$

$$Q_m^{\text{left}}(\theta) = \{(\mathbf{x}, y) \mid x_j \leq t_m\}$$

$$Q_m^{\text{right}}(\theta) = Q_m \setminus Q_m^{\text{left}}(\theta)$$

Chất lượng của một phân chia ứng viên tại nút m sau đó được tính toán bằng cách sử dụng một hàm tạp chất hoặc hàm mất mát $H(\cdot)$, việc lựa chọn hàm này phụ thuộc vào nhiệm vụ cần giải quyết (phân loại hoặc hồi quy)

$$G(Q_m, \theta) = \frac{n_m^{\text{left}}}{n_m} H(Q_m^{\text{left}}(\theta)) + \frac{n_m^{\text{right}}}{n_m} H(Q_m^{\text{right}}(\theta))$$

Chọn các tham số sao cho tối thiểu hóa tạp chất

$$\theta^* = \arg\min_{\theta} G(Q_m, \theta)$$

Đệ quy cho các tập con $Q_m^{\text{left}}(\theta^*)$ và $Q_m^{\text{right}}(\theta^*)$ cho đến khi đạt độ sâu tối đa cho phép, $n_m <$ min_samples hoặc $n_m = 1$.

Nếu mục tiêu là một kết quả phân loại nhận các giá trị 0,1,…,K-1, đối với nút m, hãy để

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

là tỷ lệ của các quan sát thuộc lớp k trong nút m. Nếu m là một nút cuối, `predict_proba` cho vùng này được đặt thành p_m. Các thước đo tạp chất phổ biến bao gồm như sau:

- Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

- Log Loss hoặc Entropy:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

2.2 Support vector machine

Support vector machine [6] là một mô hình học máy thuộc nhóm học có giám sát (supervised learning), được sử dụng cho cả bài toán phân loại (classification) và hồi quy

(regression). SVM có nguyên lý cơ bản là tìm kiếm một siêu phẳng (hyperplane) tối ưu để phân chia dữ liệu thành các lớp khác nhau với khoảng cách lớn nhất (maximum margin). Về mặt trực giác, SVM cố gắng tìm một ranh giới phân chia dữ liệu sao cho:

- Khoảng cách từ siêu phẳng đến các điểm dữ liệu gần nhất của mỗi lớp là lớn nhất.
- Các điểm dữ liệu gần siêu phẳng nhất gọi là support vectors (vector hỗ trợ).

Giải thuật mà chúng tôi sử dụng đến từ hiện thực trong scikit-learn, gọi là LinearSVC [15] - phiên bản tuyến tính của SVM với ý tưởng giải là sử dụng các phương pháp như Liblinear [7]. Cụ thể hơn, scikit-learn miêu tả SVC (Support Vector Classifier) như bên dưới.

Cho các vector huấn luyện $\mathbf{x}_i \in \mathbb{R}^p, i = 1, \ldots, n$, thuộc hai lớp, và một vector $\mathbf{y} \in \{-1, 1\}^n$, mục tiêu của chúng ta là tìm $\mathbf{w} \in \mathbb{R}^p$ và $b \in \mathbb{R}$ sao cho dự đoán được đưa ra bởi $\text{sign}(\mathbf{w}^T\phi(\mathbf{x}) + b)$ là chính xác đối với hầu hết các mẫu.

SVC giải bài toán nguyên bản sau:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

với điều kiện $y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i,$

$$\xi_i \geq 0, i = 1, \ldots, n$$

Về mặt trực giác, chúng ta đang cố gắng tối đa hóa lề (bằng cách tối thiểu hóa $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$), đồng thời chịu một hình phạt khi một mẫu bị phân loại sai hoặc nằm trong ranh giới lề. Lý tưởng nhất, giá trị $y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)$ sẽ ≥ 1 cho tất cả các mẫu, điều này cho thấy một dự đoán hoàn hảo. Nhưng các bài toán thường không phải lúc nào cũng có thể phân tách hoàn hảo bằng một siêu phẳng, vì vậy chúng ta cho phép một số mẫu nằm ở khoảng cách ξ_i từ ranh giới lề chính xác của chúng. Hệ số phạt C kiểm soát mức độ của hình phạt này, và kết quả là, đóng vai trò như một tham số điều chuẩn nghịch đảo (xem ghi chú bên dưới).

Bài toán trên có thể được diễn đạt một cách tương đương như sau:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)),$$

Trong đó, có sử dụng hàm mất mát hinge. Đây là dạng được tối ưu hóa trực tiếp bởi `LinearSVC`, nhưng không giống như dạng đối ngẫu, dạng này không liên quan đến tích vô hướng giữa các mẫu, vì vậy thủ thuật *kernel* nổi tiếng không thể được áp dụng. Đây là lý do tại sao chỉ có *kernel* tuyến tính được hỗ trợ bởi `LinearSVC` (ϕ là hàm đồng nhất). Tại đây có hai phiên bản hàm mất mát có thể dùng: hinge [6] hoặc squared hinge [13].

2.3 Bagging

Trong các thuật toán ensemble, phương pháp bagging [2] tạo thành một lớp các thuật toán xây dựng nhiều phiên bản của một mô hình trên các tập con ngẫu nhiên của tập huấn luyện ban đầu, sau đó tổng hợp các dự đoán riêng lẻ của chúng để tạo ra một dự đoán cuối cùng. Các phương pháp này được sử dụng để giảm phương sai của mô hình ban đầu (ví dụ như cây quyết định), bằng cách tạo tập dữ liệu con ngẫu nhiên. Trong nhiều trường hợp, các phương pháp bagging là một cách rất đơn giản để cải thiện so với một mô hình đơn lẻ, mà không cần phải điều chỉnh giải thuật bên dưới. Vì chúng cung cấp một cách để giảm overfitting, các phương pháp bagging hoạt động tốt nhất với các mô hình mạnh và phức tạp (ví dụ như cây quyết định đầy đủ), trái ngược với các phương pháp boosting thường hoạt động tốt nhất với các mô hình yếu (ví dụ như cây quyết định nông). Các phương pháp bagging có nhiều biến thể nhưng chủ yếu khác nhau ở cách chúng lấy các tập con ngẫu nhiên của tập huấn luyện, xem thêm tại [2][3][12][14].

Trong scikit-learn, các phương pháp bagging được cung cấp dưới dạng class thống nhất BaggingClassifier [15]. Nhận đầu vào là một mô hình do người dùng chỉ định cùng với các tham số xác định chiến lược để lấy các tập con ngẫu nhiên. Cụ thể, max_samples và max_features kiểm soát kích thước của các tập con (về mặt mẫu và đặc trưng), trong khi bootstrap và bootstrap_features kiểm soát việc các mẫu và đặc trưng được lấy có hoàn lại hay không.

Một đặc điểm quan trọng của bagging là tính song song tính toán. Vì các mô hình ”yếu” được huấn luyện độc lập trên các tập dữ liệu bootstrap (tập con), việc huấn luyện chúng có thể diễn ra cùng lúc trên nhiều CPU hoặc GPU. Điều này giúp rút ngắn đáng kể thời gian training khi làm việc với tập dữ liệu lớn hoặc số lượng mô hình lớn.

2.4 Boosting

Trong học máy, khi một mô hình đơn lẻ quá yếu để dự đoán tốt, chúng ta cần một cách bổ sung dần dần sức mạnh cho mô hình. Boosting [9] ra đời từ nhu cầu đó với ý tưởng cốt lõi là: thay vì xây một mô hình mạnh ngay từ đầu, ta xây dựng nhiều mô hình yếu, mỗi mô hình học từ lỗi của cái trước, dần dần tạo nên một mô hình mạnh hơn.

Scikit-learn cung cấp giải thuật AdaBoost [15], một phương pháp Boosting phổ biến, ra đời vào năm 1995 bởi Freund và Schapire [9].

Nguyên tắc cốt lõi của AdaBoost là để phù hợp với một chuỗi các mô hình yếu (tức là, các mô hình chỉ nhỉnh hơn một chút so với dự đoán ngẫu nhiên, chẳng hạn như các cây quyết định nhỏ) trên các phiên bản được sửa đổi lặp đi lặp lại của dữ liệu. Các dự đoán từ tất cả các mô hình yếu này sau đó được kết hợp thông qua một cuộc bỏ phiếu đa số có trọng số (hoặc tổng) để tạo ra dự đoán cuối cùng. Các sửa đổi dữ liệu tại mỗi lần lặp tăng cường (gọi là boosting iteration) bao gồm việc áp dụng các trọng số $w_1, w_2, \ldots, w_N$ cho từng mẫu huấn luyện. Ban đầu, các trọng số này đều được đặt là $w_i = 1/N$, do đó bước đầu tiên đơn giản là huấn luyện một mô hình yếu trên dữ liệu gốc. Đối với mỗi lần lặp tiếp theo, các trọng số mẫu được cập nhật riêng lẻ và mô hình học được áp dụng lại cho dữ liệu có trọng số mới.

Tại một bước nhất định, những mẫu bị dự đoán sai bởi mô hình được tạo ra ở bước trước sẽ có trọng số tăng lên, trong khi trọng số giảm cho những mẫu mà dự đoán đúng. Khi các lần lặp tiếp tục, các mẫu khó dự đoán nhận được ảnh hưởng ngày càng lớn. Mỗi mô hình yếu tiếp theo do đó mà bị ép buộc phải tập trung vào các mẫu bị bỏ sót bởi các mô hình trước đó [11].

Chapter 3

Thử nghiệm

Trong phần này, ta tiến hành huấn luyện và kiểm thử các mô hình kết hợp Bagging và Boosting, so sánh chúng với các mô hình cơ bản như Decision tree và SVM. Toàn bộ mã nguồn thực nghiệm có thể tìm thấy tại: `https://github.com/Pinminh/ml-flock`.

3.1 Mô tả tập dữ liệu

Để thử nghiệm các mô hình, toàn bộ sẽ được huấn luyện và kiểm thử trên tập dữ liệu Swarm Behaviour (hành vi bầy đàn). Tập dữ liệu này được miễn phí truy cập tại: `https://archive.ics.uci.edu/dataset/524/swarm+behaviour`. Bên dưới là mô tả tổng quan về tập dữ liệu này.

Mỗi mẫu trong tập dữ liệu biểu diễn trạng thái của 200 thực thể (gọi là boid), với tổng cộng 2.400 đặc trưng. Với mỗi boid m, các đặc trưng bao gồm:

- Vị trí (x_m, y_m): Tọa độ cụ thể của biến ngẫu nhiên (X, Y) của mỗi boid.
- Vector vận tốc (x_{Veln}, y_{Veln}): Vận tốc của mỗi boid.
- Vector căn chỉnh (x_{Am}, y_{Am}): Hướng căn chỉnh với các boid lân cận.
- Vector tách biệt (x_{Sm}, y_{Sm}): Cho biết mức độ xa cách so với các boid lận cận.
- Vector hội tụ (x_{Cm}, y_{Cm}): Hướng di chuyển về trung tâm các boid lân cận.
- Số lượng boid trong bán kính căn chỉnh/hội tụ n_{ACm}.

- Số lượng boid trong bán kính tách biệt n_{Sm}.

Các thuộc tính này được lặp lại cho mỗi boid, với số chỉ boid m từ 1 đến 200, tức là tổng số có 2.400 đặc trưng (200 boid × 12 thuộc tính). Mỗi một mẫu trong tập dữ liệu được gán nhãn 0 (biểu thị không có hành vi bầy đàn) hoặc 1 (biểu thị có hành vi bầy đàn). Tập dữ liệu có đến 24.016 mẫu.

3.2 Tiền xử lý dữ liệu

Mặc dù tập dữ liệu được báo cáo là không có giá trị thiếu, trong quá trình kiểm tra, chúng tôi đã phát hiện một số giá trị NaN ở một số hàng. Điều này có thể xảy ra do các vấn đề kỹ thuật trong quá trình tải hoặc xử lý dữ liệu. Để xử lý, xóa các hàng chứa giá trị NaN, vì số lượng giá trị thiếu không đáng kể và việc xóa không ảnh hưởng lớn đến kích thước tập dữ liệu (24.016 mẫu). Tập dữ liệu giảm còn 24.015 mẫu.

Để đảm bảo hiệu suất của các thuật toán máy học, đặc biệt là các mô hình dựa trên khoảng cách như K-Nearest Neighbors hoặc Support Vector Machines, chúng tôi đã sử dụng phương pháp StandardScaler từ thư viện scikit-learn. Phương pháp này chuẩn hóa dữ liệu sao cho mỗi đặc trưng có trung bình 0 và độ lệch chuẩn 1, giúp các đặc trưng đóng góp đồng đều vào mô hình.

Sau khi xử lý giá trị thiếu và chuẩn hóa, chúng tôi đã chia tập dữ liệu thành 3 phần: tập huấn luyện (train set), tập kiểm tra (test set), tập, với tỷ lệ 80% huấn luyện và 20% kiểm tra. Việc chia dữ liệu được thực hiện bằng hàm train_test_split từ thư viện scikit-learn, đảm bảo bảo toàn tỉ lệ nhãn rằng dữ liệu, giúp mỗi tập con đại diện cho toàn bộ tập dữ liệu.

Tỉ lệ nhãn (0 và 1) trong tập dữ liệu cân bằng nhau (xem hình bên dưới), nghĩa là một số độ đo như accuracy sẽ phản ánh chuẩn xác hơn khả năng của mô hình.

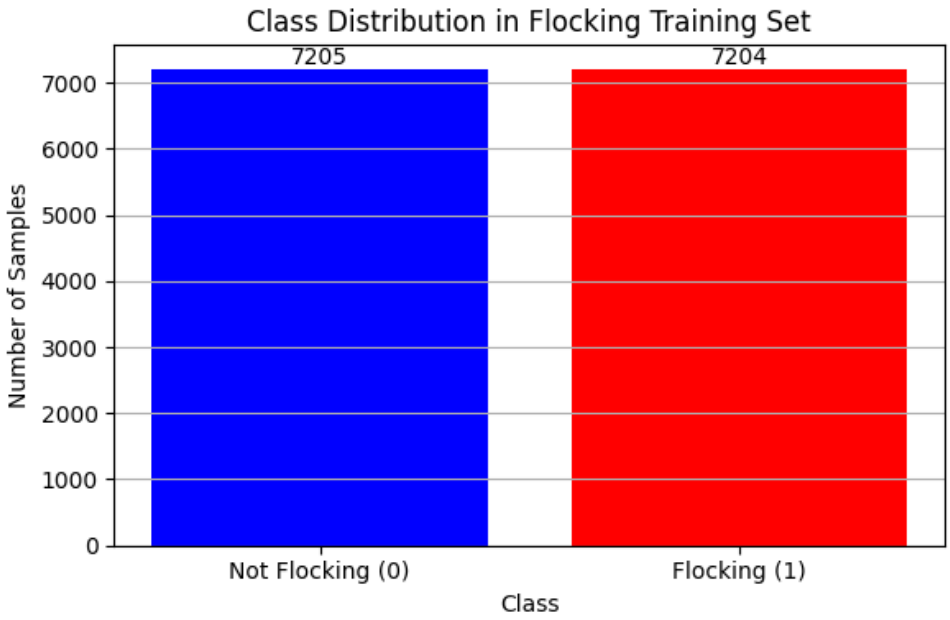

Figure 3.1: Phân bố nhãn trong tập dữ liệu huấn luyện

Bây giờ, hãy xem xét 4 features (cột) đầu tiên trong tập dữ liệu như hình dưới.

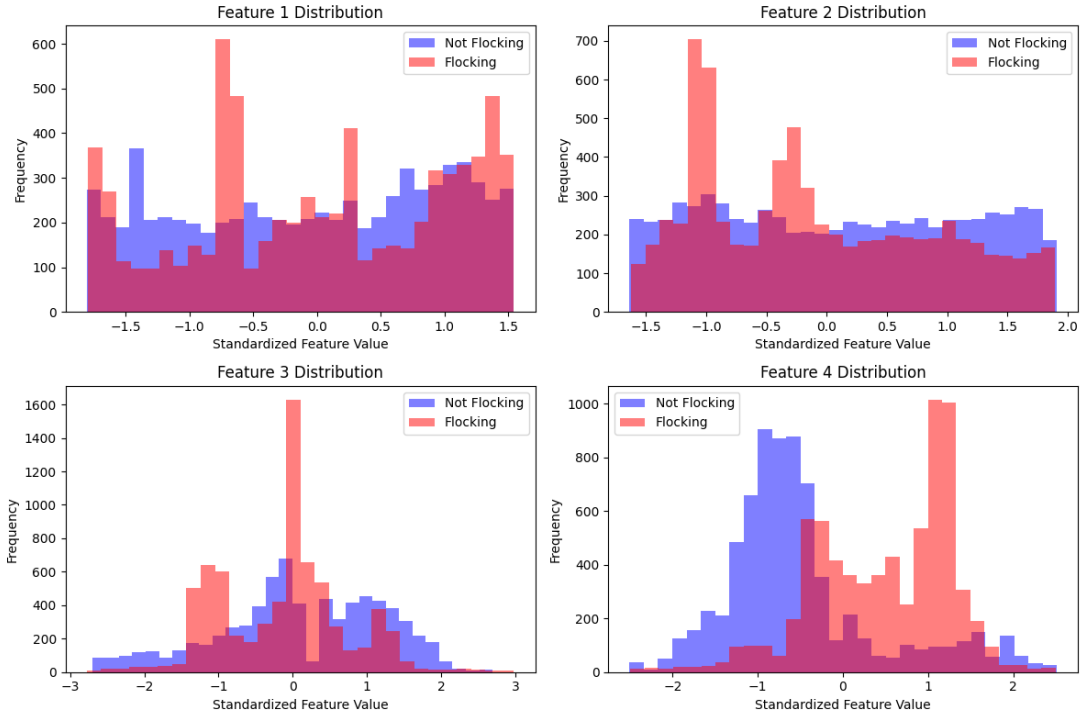

Figure 3.2: Tần suất nhãn trên mỗi giá trị của 4 features đầu tiên

Feature từ 1 đến 3 có phần diện tích chồng lên nhau nhiều, chứng tỏ khó mà dùng giá trị của các features này để phân biệt not flocking - flocking. Trong khi đó feature 4 ta thấy tần suất not flocking và flocking có hai phân bố ít chồng lắp lên nhau, nên ta có thể

dùng feature 4 để phân biệt not flocking và flocking. Như vậy, có những feature hữu ích và cũng có các feature không có nhiều ảnh hưởng. Bước tiếp theo cần thực hiện là lựa chọn các features có giá trị cao, tránh trùng lắp, gây giảm hiệu suất các mô hình.

3.3 Lựa chọn feature

Để thực hiện lựa chọn đặc trưng, chúng tôi đã sử dụng ba phương pháp: VarianceThreshold, Mutual Information, và Correlation Filter. Các phương pháp này được chọn vì chúng bổ sung cho nhau, giúp loại bỏ các đặc trưng không thông tin, xác định các đặc trưng quan trọng đối với biến mục tiêu, và giảm thiểu redundancy giữa các đặc trưng. Quá trình này được thực hiện bằng cách sử dụng các công cụ từ thư viện scikit-learn và pandas.

Variance Threshold [15] là kỹ thuật đơn giản áp dụng đầu tiên trong quá trình chọn features. Nó hiệu quả trong loại bỏ các đặc trưng có độ lệch chuẩn thấp. Độ lệch chuẩn (variance) của một đặc trưng phản ánh mức độ thay đổi của nó trong tập dữ liệu. Các đặc trưng có variance thấp thường gần như không thay đổi giữa các mẫu, do đó không cung cấp thông tin hữu ích để phân biệt các lớp. Chúng tôi đặt threshold là 0.1 (khá lớn so với thang giá trị đã chuẩn hóa). Kết quả thu được là không có loại bỏ features nào cả, sau bước này 2.400 features vẫn được giữ nguyên.

Mutual Information [19] là một phương pháp tiếp nối sau Variance Threshold, đo lường mức độ phụ thuộc giữa một đặc trưng và biến mục tiêu. Nó đánh giá lượng thông tin mà một đặc trưng cung cấp về biến mục tiêu, bao gồm cả các mối quan hệ phi tuyến tính, điều này đặc biệt quan trọng trong tập dữ liệu phức tạp như Hành vi bầy đàn, nơi các đặc trưng như vị trí, vận tốc, hoặc vector căn chỉnh có thể có mối quan hệ không đơn giản với hành vi. Chúng tôi chọn k features có mutual information cao nhất, giá trị k sẽ thông qua fine-tuning (tinh chỉnh) mà biết được.

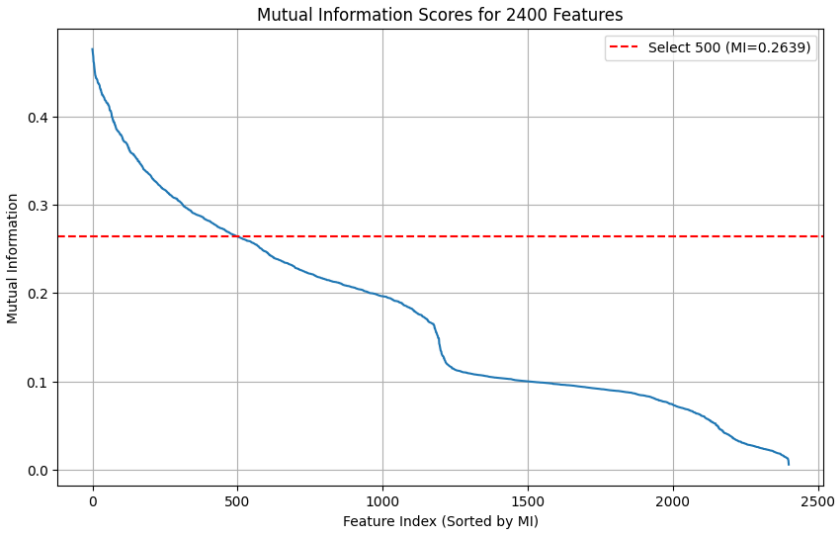

Figure 3.3: Mutual information của mỗi feature xếp từ cao đến thấp

Correlation Filter [10][5] được áp dụng sau Mutual Information (lượng features đã giảm). Kỹ thuật này tập trung vào việc loại bỏ các đặc trưng có tương quan cao hơn threshold ct (số ct được tìm qua fine-tuning), vì các đặc trưng tương quan cao thường mang thông tin dư thừa, có thể gây ra hiện tượng multicollinearity và làm giảm hiệu suất của mô hình. Trong tập dữ liệu hành vi bầy đàn, có khả năng tồn tại nhiều đặc trưng tương quan cao, chẳng hạn như vị trí hoặc vận tốc của các boid gần nhau.

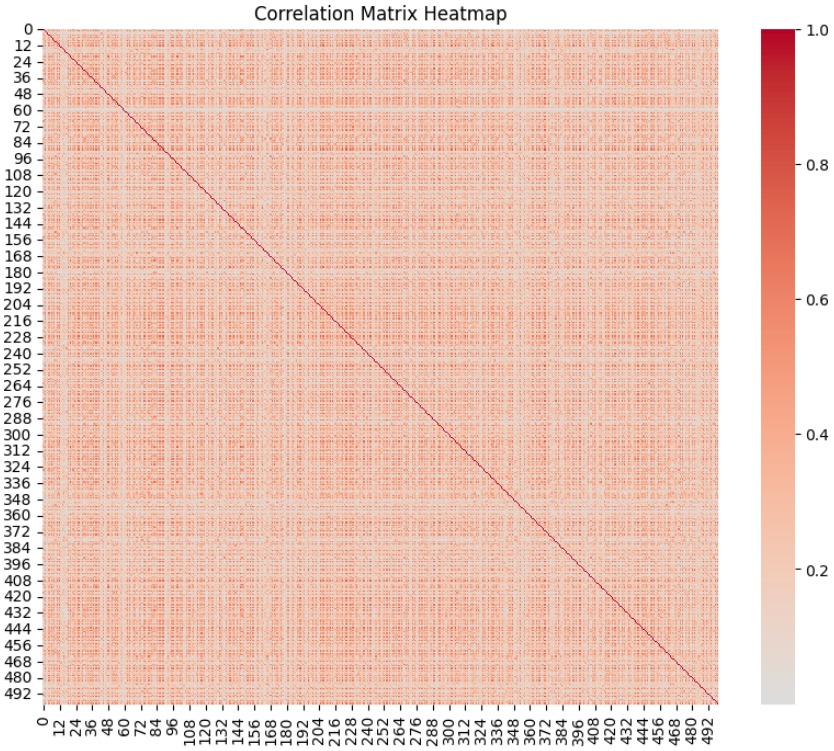

Figure 3.4: Biểu đồ nhiệt cho thấy độ tương quan giữa 300 features

3.4 Huấn luyện và tinh chỉnh

Quá trình này được thực hiện chủ yếu trên tập train và tập validation (cho tinh chỉnh fine-tuning). Chúng tôi sử dụng một Pipeline bao gồm các bước theo thứ tự: Standard Scaler $\rightarrow$ Variance Threshold $\rightarrow$ Select k best Mutual Information $\rightarrow$ Correlation Threshold Filter $\rightarrow$ Models.

Chúng tôi sử dụng **Randomized Search CV** [1] để đi qua một số tổ hợp hyperparameters (siêu tham số) và kiểm thử với tập validation bằng các độ đo như accuracy [17], F1-score [20], AUC ROC [8]. Tuy nhiên, trong các độ đo đó, các tổ hợp hyperparameters sẽ được xếp hạng dựa trên F1-score. Randomized Search CV được sử dụng khoảng từ 5 đến 50 lần lặp. Bên dưới là các đồ thị cho liệt kê từ tổ hợp tốt nhất đến xấu nhất.

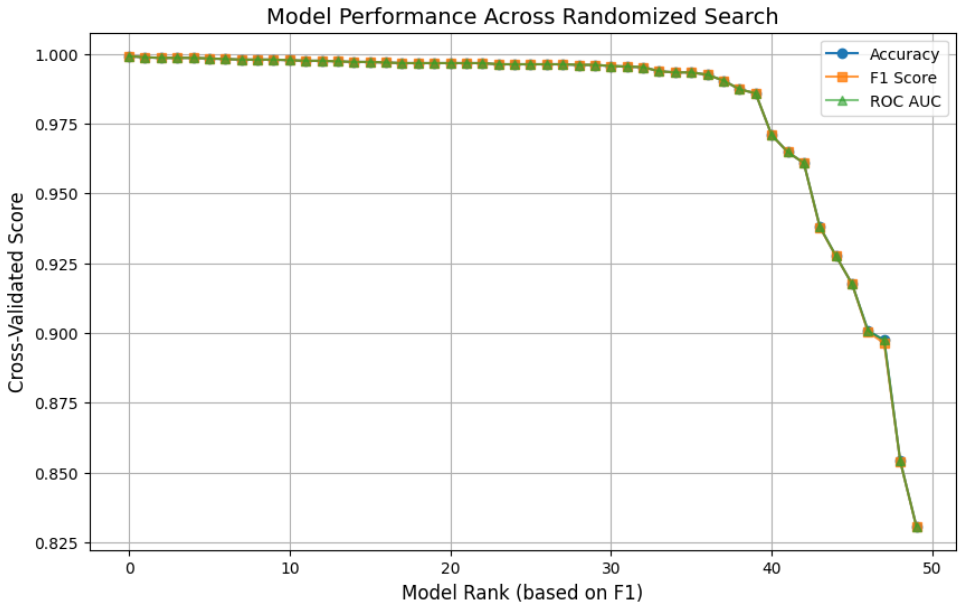

Figure 3.5: Độ đo thay đổi theo các tổ hợp hyperparameters cho Decision Tree

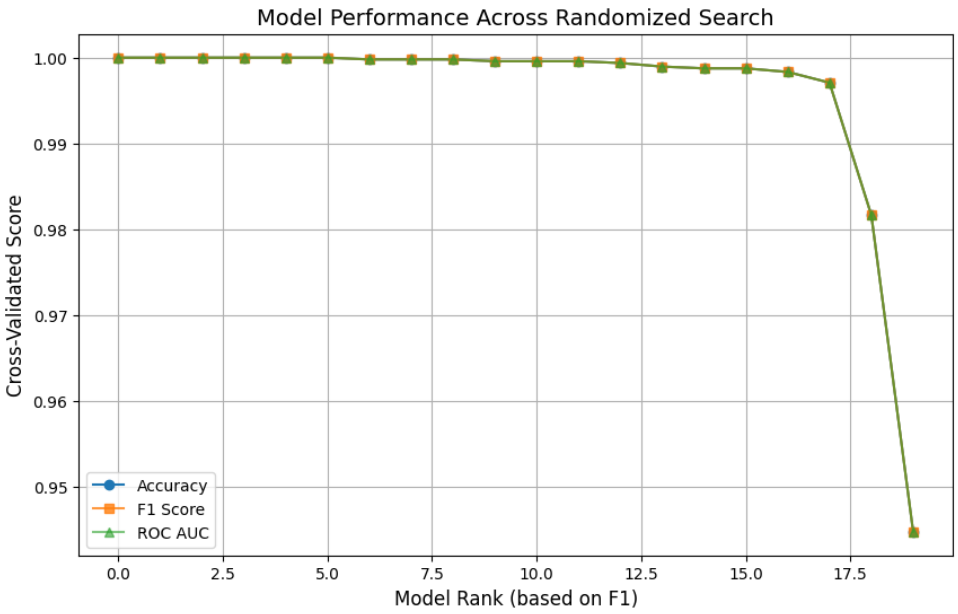

Figure 3.6: Độ đo thay đổi theo các tổ hợp hyperparameters cho Linear SVC

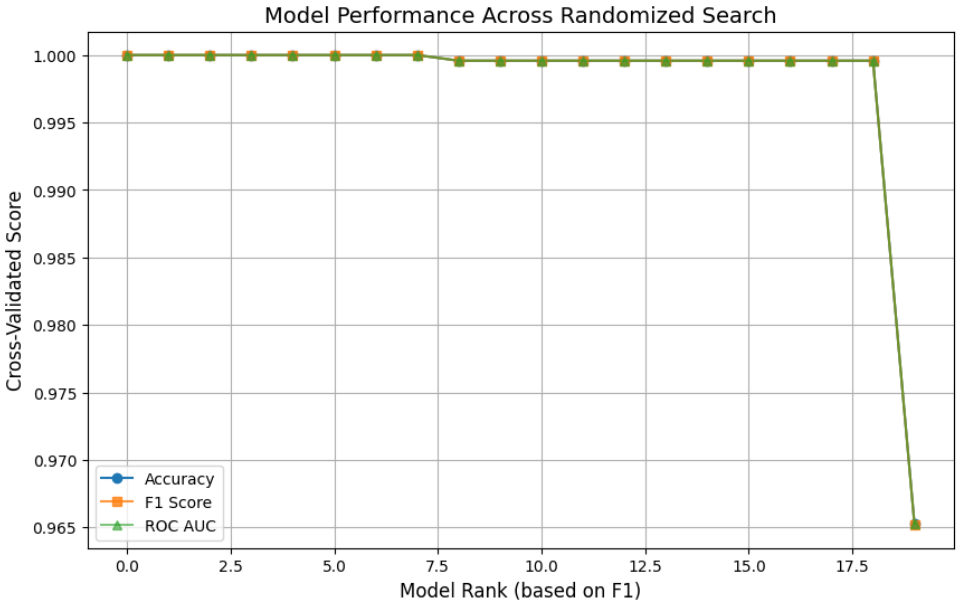

Figure 3.7: Độ đo thay đổi theo các tổ hợp hyperparameters cho Bagging Decision Tree

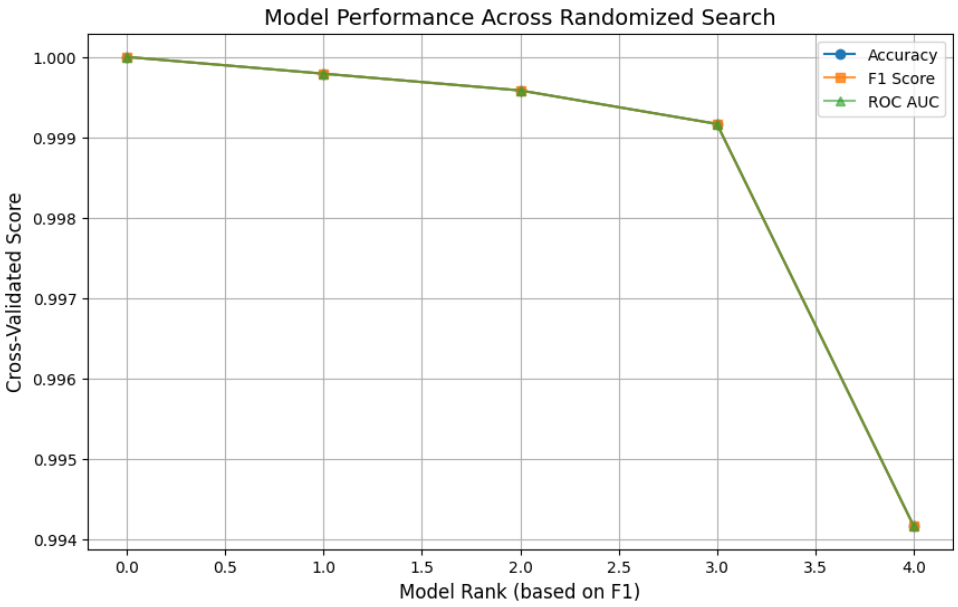

Figure 3.8: Độ đo thay đổi theo các tổ hợp hyperparameters cho Bagging Linear SVC

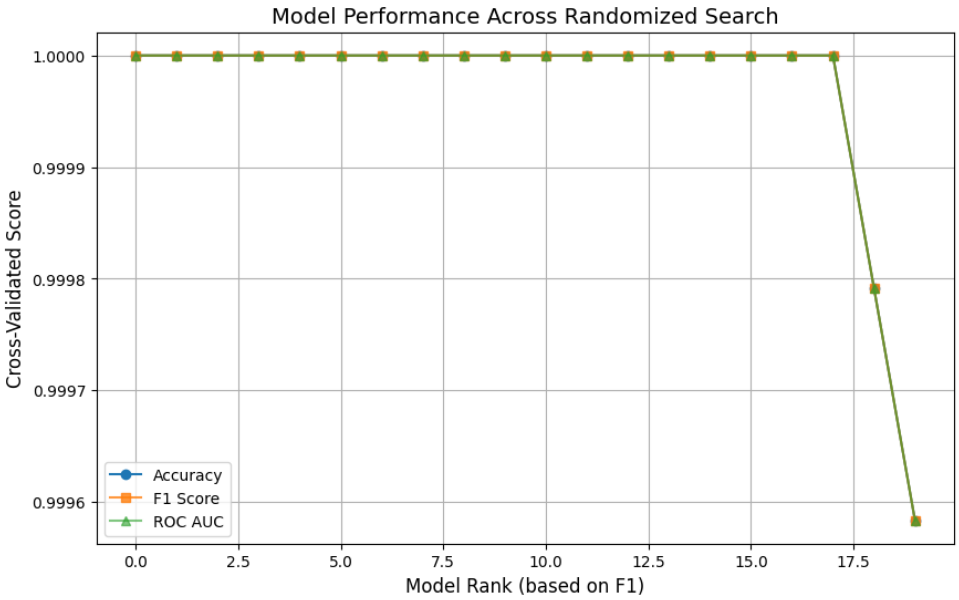

Figure 3.9: Độ đo thay đổi theo các tổ hợp hyperparameters cho AdaBoost Decision Tree

Và các kết quả tổ hợp hyperparameters tốt nhất cho từng mô hình được liệt kê như trong bảng dưới đây:

Table 3.1: Tổ hợp hyperparameters tốt nhất cho từng mô hình

Classifier	**Best Hyperparameters**
DecisionTree	{criterion: gini, max_depth: 20, max_features: 0.3, min_samples_leaf: 5, min_samples_split: 14, ct_threshold: 0.8, mi_k: 1500}
LinearSVC	{mi_k: 600, ct_threshold: 0.7, penalty: l2, loss: squared_hinge, C: 100}
BaggingDT	{bootstrap: False, bootstrap_features: True, max_features: 0.55, max_samples: 0.73, n_estimators: 50, ct_threshold: 0.5, mi_k: 800}
BaggingLSVC	{bootstrap: True, bootstrap_features: False, max_features: 0.98, max_samples: 0.87, n_estimators: 50, ct_threshold: 0.8, mi_k: 700}
AdaBoostDT	{learning_rate: 0.38, n_estimators: 50, ct_threshold: 0.6, mi_k: 1100}

Các tên biến được giải thích như sau:

- **clf__criterion**: Hàm được sử dụng để đo lường chất lượng của việc chia nhánh trong cây quyết định. Các giá trị phổ biến là `gini` hoặc `entropy`.
- **clf__max_depth**: Độ sâu tối đa của cây quyết định. Giới hạn số tầng của cây nhằm kiểm soát việc overfitting.
- **clf__max_features**: Tỷ lệ hoặc số lượng đặc trưng sẽ được xem xét khi tìm kiếm cách chia tốt nhất.
- **clf__min_samples_leaf**: Số lượng mẫu tối thiểu cần có tại một nút lá.
- **clf__min_samples_split**: Số lượng mẫu tối thiểu cần thiết để tách một nút bên trong.
- **ct__threshold**: Ngưỡng dùng trong bộ lọc tương quan. Các đặc trưng có độ tương quan cao hơn giá trị này sẽ bị loại bỏ.

- **mi__k**: Số lượng đặc trưng hàng đầu được chọn dựa trên điểm số Thông tin Tương hỗ (Mutual Information).
- **clf__estimator__penalty**: Chuẩn được sử dụng trong việc phạt mô hình LinearSVC (ví dụ: `l1`, `l2`).
- **clf__estimator__loss**: Hàm mất mát được sử dụng bởi LinearSVC (ví dụ: `squared_hinge`).
- **clf__estimator__C**: Độ mạnh của việc regularization trong LinearSVC. Giá trị nhỏ hơn tương ứng với việc regularization mạnh hơn.
- **clf__bootstrap**: Có sử dụng lấy mẫu bootstrap khi xây dựng các bộ ước lượng trong Bagging hay không.
- **clf__bootstrap_features**: Có lấy mẫu bootstrap đối với các đặc trưng khi tạo tập huấn luyện cho Bagging hay không.
- **clf__max_samples**: Tỷ lệ mẫu được chọn từ tập dữ liệu gốc để huấn luyện mỗi bộ ước lượng cơ sở trong Bagging.
- **clf__n_estimators**: Số lượng bộ ước lượng cơ sở (như cây quyết định hoặc mô hình khác) trong các phương pháp ensemble như Bagging và AdaBoost.
- **clf__learning_rate**: Hệ số làm giảm đóng góp của mỗi bộ phân loại trong AdaBoost nhằm tránh overfitting.

3.5 Kiểm thử

Sau khi đã có các tổ hợp tốt nhất (dựa trên F1-score), ta tiếp tục thực hiện kiểm thử các mô hình thông qua tập kiểm thử (test set). Kết quả sau khi kiểm thử toàn bộ mô hình bằng test set được viết lại trong bảng dưới.

Table 3.2: Các độ đo cho các mô hình

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC AUC**
DecisionTree	0.998751	0.998752	0.998751	0.998751	0.998751
LinearSVC	1.000000	1.000000	1.000000	1.000000	1.000000
BaggingDT	1.000000	1.000000	1.000000	1.000000	1.000000
BaggingLSVC	1.000000	1.000000	1.000000	1.000000	1.000000
AdaBoostDT	1.000000	1.000000	1.000000	1.000000	1.000000

Chapter 4

Kết luận

Báo cáo này trình bày quá trình phân tích toàn diện tập dữ liệu Hành vi bầy đàn (UCI Machine Learning Repository), một nguồn tài nguyên quan trọng để nghiên cứu hành vi tập thể như đàn đúm, căn chỉnh, và nhóm họp thông qua mô phỏng. Tập dữ liệu bao gồm ba phần con (Aligned, Flocking, và Grouped), mỗi phần chứa 24.017 mẫu và 2.400 đặc trưng, tạo ra một nền tảng phong phú nhưng đầy thách thức cho các ứng dụng học máy, đặc biệt trong các nhiệm vụ phân loại nhị phân.

Trong giai đoạn huấn luyện và tinh chỉnh, chúng tôi đã xây dựng một chuỗi xử lý tích hợp các bước lựa chọn đặc trưng với các mô hình phân loại, bao gồm Cây quyết định (Decision Tree - DT) và Bộ phân loại vector hỗ trợ tuyến tính (Linear Support Vector Classifier - LSVC). Các tham số của Mutual Information (số lượng đặc trưng được chọn) và Correlation Filter (ngưỡng tương quan) được tinh chỉnh thông qua tìm kiếm lưới và tìm kiếm ngẫu nhiên, kết hợp với chéo kiểm tra 5-fold để tối ưu hóa hiệu suất. Các chỉ số đánh giá như độ chính xác (accuracy), độ chính xác (precision), độ nhạy (recall), điểm F1, và AUC-ROC được sử dụng để so sánh hiệu suất của các mô hình.

Kết quả đánh giá cho thấy cả DT và LSVC đều đạt được hiệu suất đáng kể trên các chỉ số đánh giá. Tuy nhiên, LSVC thể hiện khả năng tổng quát hóa tốt hơn, với sự chênh lệch nhỏ giữa hiệu suất trên tập huấn luyện và tập kiểm tra, trong khi DT có dấu hiệu overfitting, đặc biệt khi hiệu suất trên tập huấn luyện cao hơn đáng kể so với tập kiểm tra. Những phát hiện này nhấn mạnh tầm quan trọng của việc lựa chọn mô hình phù hợp với đặc điểm của dữ liệu, đặc biệt khi dữ liệu có tính tuyến tính mạnh, nơi LSVC có lợi thế.

References

[1] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS-12)*. 2012, pp. 1–8.

[2] Leo Breiman. “Bagging predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140.

[3] Leo Breiman. “Pasting small votes for classification in large databases and online”. In: *Machine Learning* 36.1 (1999), pp. 85–103.

[4] Leo Breiman et al. *Classification and Regression Trees*. CRC Press, 1984.

[5] Wei Chu et al. “Feature selection using mutual information and correlation-based methods”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.2 (2013), pp. 415–426.

[6] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[7] Rong-En Fan et al. “LIBLINEAR: A library for large linear classification”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[8] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874.

[9] Yoav Freund and Robert E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504.

[10] Mark A. Hall. “Correlation-based feature selection for discrete and numeric class machine learning”. In: *Proceedings of the Seventeenth International Conference on Machine Learning* (1999), pp. 359–366.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer, 2009. ISBN: 978-0-387-84857-0.

[12] Tin Kam Ho. “The random subspace method for constructing decision forests”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8 (1998), pp. 832–844.

[13] Chih-Chung Lee and Chih-Jen Lin. “A Study on L2-Loss (Squared Hinge-Loss) Multiclass SVM”. In: *Neural Computation* 25.5 (2013), pp. 1302–1323. DOI: 10.1162/NECO_a_00434.

[14] Gilles Louppe and Pierre Geurts. “Ensembles on Random Patches”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 346–361.

[15] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[16] Vili Podgorelec et al. “Decision Trees: An Overview and Their Use in Medicine”. In: *Journal of Medical Systems* 26.5 (2002), pp. 445–463. DOI: 10.1023/A:1016409317640.

[17] David M. W. Powers. “Evaluation: From precision, recall and F-measure to ROC, Informedness, Markedness and Correlation”. In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63.

[18] J. R. Quinlan. “Induction of Decision Trees”. In: *Machine Learning* 1.1 (1986), pp. 81–106. DOI: 10.1023/A:1022643204877.

[19] Claude E. Shannon. “A Mathematical Theory of Communication”. In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[20] C.J. Van Rijsbergen. “The use of the F1 score to evaluate classification systems in the presence of imbalanced classes”. In: *Information Retrieval* 3.1 (1979), pp. 57–61.